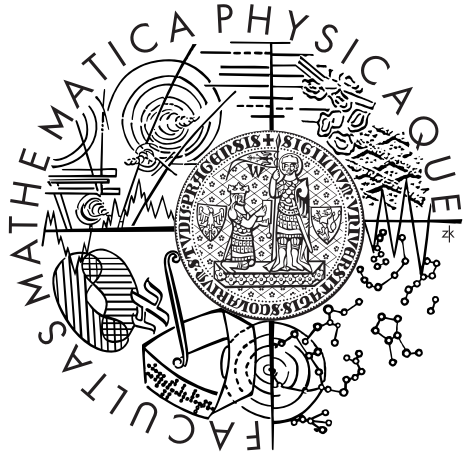
Charles University in Prague
Faculty of Mathematics and Physics

DOCTORAL THESIS



Jan Bulánek

The Online Labeling Problem

Department of Theoretical Computer Science
and Mathematical Logic,
and
Institute of Mathematics of the Academy of Sciences
of the Czech Republic

Supervisor of the doctoral thesis: Michal Koucký

Study programme: Computer Science (4I1)

Prague 2014

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature of the author

Název práce: Problém Online Labelingu

Autor: Mgr. Jan Bulánek

Katedra: Katedra teoretické informatiky a matematické logiky a
Matematický ústav Akademie věd České republiky

Vedoucí disertační práce: Doc. Mgr. Michal Koucký, Ph.D.,
Informatický ústav Univerzity Karlovy

Abstrakt: Setříděné pole je zásadní algoritmický koncept, jehož on-line varianta je základem pro problém online labelingu. Problém online labelingu je definován následovně. Vstupem je pole velikosti m a posloupnost celých čísel z univerza $\{1, \ldots, r\}$ v libovolném pořadí délky n. Naším úkolem je udržovat všechna přijatá čísla setříděná v poli. Mezi vloženými čísly mohou být mezery. Protože závěrečné pořadí čísel nelze určit, dokud nejsou všechna vložena, je povoleno čísla přesouvat. Cílem je minimalizovat počet přesunů.

Ukážeme dva algoritmy, které společně poskytují optimální řešení pro téměř všechny možné hodnoty m coby funkce n. Dokážeme těsné dolní odhady pro téměř všechny hodnoty m. Zavedeme notaci omezeného univerza vstupní množiny a dokážeme dolní odhady i pro tuto variantu. Dokážeme dolní odhady i pro případ randomizovaných algoritmů.

Klíčová slova: problém online labelingu, problém file maintenance, dolní odhady, horní odhady

Title: The Online Labeling Problem

Author: Mgr. Jan Bulánek

Department: Department of Theoretical Computer Science and Mathematical Logic, and
Institute of Mathematics of the Academy of Sciences of the Czech Republic

Supervisor: Doc. Mgr. Michal Koucký, Ph.D.,
Computer Science Institute of Charles University

A sorted array is a fundamental algorithmic concept. Its on-line variant gives rise to the *online labeling problem*. In the online labeling problem we are given an array of size m and a stream of n integers from the universe $\{1, \ldots, r\}$ coming in an arbitrary order. Our task is to maintain all received items in the array in sorted order. The inserted items do not have to be stored consecutively in the array. Since the final order of the items is not known until we see all the items, moves of already inserted items are allowed but should be minimized.

We present two algorithms which together provide an optimal solution for almost all values of m as a function of n. We provide tight lower bounds for almost all ranges of m. We introduce a notion of the limited universe and prove lower bounds also in that setting. Some of our lower bounds also apply to randomized algorithms.

Keywords: online labeling problem, file maintenance problem, lower bounds, upper bounds

Acknowledgments

This thesis could not have been written without the support of many people. The most important person was my advisor Michal Koucký who not only was a great teacher but also supported me and helped me on many occasions going far beyond our work. I am very grateful for everything he has taught me and for being a true friend to me.

I am also grateful to all I have worked with, besides Michal, they were Mike Saks, who is a co-author of all new results in this thesis and my fellow students Martin Babka and Vladimír Čunát who are co-authors of results in Chapters 6 and 8. It is impossible not to mention Vašek Koubek here. His comments to result in Chapter 8 were invaluable and he pointed out to several weak points of the proof. Finally, I had a pleasure to work with my friend Zbyněk Falt on his interesting projects.

Also, I would like to thank my previous advisor Daniel Král', for introducing me to Michal.

I could not have been writing these lines without the continuous support of my parents, my wife Petra and our son Honzík.

During my studies, I was partially supported by GAUK project no. 344711, grant GA ČR P202/10/0854, grant IAA100190902 of GA AV ČR and project No. 1M0021620808 of MŠMT ČR.

Finally, I would like to thank Department of Theoretical Computer Science and Mathematical Logic for the opportunity of being a doctoral student under Michal's supervision. I am also very grateful to Institute of Mathematics of the Academy of Sciences of the Czech Republic for providing me an office and other support for my research.

Contents

1. Introduction

The construction of effective algorithms and data structures is the central focus of computer science since its very beginning. As the amount of data we need to process increases this is an important topic even though the computational power of our devices is increasing. Therefore, we need to design the best algorithms or data structures possible. To do so we need to be able to recognize what is the best possible solution. Thus, we need to prove lower bounds on the best possible cost of algorithms and data structures.

In this thesis we do both. We design algorithms and prove lower bounds on their cost. For a particular problem called the *online labeling problem* we not only present a new algorithm which is superior to known algorithms for certain input sizes, but we also prove matching general lower bounds.

The online labeling problem is tightly connected to sorting. Sorting is no doubt one of the fundamental problems of computer science. We know optimal time complexity of sorting. Sorted arrays are easy to use, provide asymptotically optimal search time and utilize caches well during scan query.

A natural question is, whether inserts of additional items in a sorted array can be implemented efficiently. There are immediate applications of such a structure in algorithm and data structure design, e.g., dynamic sorted arrays, priority queues, dictionaries etc. First, we formulate the problem.

You are given an array of size m and a stream of n integers coming in arbitrary order where $n \leq m$. Your task is to maintain all already received items in the array in sorted order. The inserted items do not have to be stored consecutively in the array. Since the final order of the items is not known until we see all the items, moves of already inserted items are allowed but should be minimized. The obvious solution moves $O(n)$ items after each insert with the total complexity $O(n^2)$. This, however, does not benefit from the fact that the size of the array may be significantly larger than n.

This is one of several possible formulations of online labeling problem which is also known as the *file maintenance problem*. In the general formulation, items from a universe U of size r are assigned *labels* from a given linearly ordered universe of size m. The ordering of their labels must respect the ordering of items. Relabeling of items (which is equivalent to moving items in the array) is allowed, but should be minimized.

The first non-trivial algorithm for the online labeling problem was given at the beginning of the eighties by Itai, Konheim and Rodeh [17]. The amortized time complexity was $\Omega(\log^2(n))$ per inserted item, while the size of the array is only cn for any constant $c > 1$. This surprising result was achieved by a clever utilization of empty space in the arrays. Since then, several other algorithms were designed,

but it remained open whether whether any of them is asymptotically optimal.

In this thesis we resolve this question by proving lower bounds matching the complexity of these algorithms. Indeed, we prove optimal lower bounds for almost all possible array sizes, in the case of linear size arrays we prove such lower bounds even under the assumption of limited universe ($r \in O(n)$ while previous results assume $r \geq 2^n$), and we also prove the first lower bound for randomized algorithms for the online labeling problem.

Thus, we essentially completely determine the optimal complexity of deterministic algorithms for the online labeling problem. There are two possible further directions how to extend our results: Are there asymptotically better randomized algorithms for the online labeling problem? We know that in the case of polynomial size arrays the answer is negative. In the other direction one can ask whether it is possible to obtain better algorithms when the size of the universe from which the items are selected is comparable to the size of the array. We know that in the case of linear size arrays the answer is negative as well but what about larger arrays?

1.1 Applications

The very first algorithm for the online labeling problem was given by Itai, Konheim and Rodeh [17] who used it as a tool for an implementation of a priority queue. This is a natural application as one can see the online labeling problem as a dynamic maintenance of sorted array. Unfortunately, the best algorithm for the online labeling problem in arrays of linear size is worse than binary trees by a logarithmic factor which makes approach in [17] inferior to them.

However, in recent years there has been renewed interest in the online labeling problem problem because of its applications in the design of cache-oblivious algorithms, e.g., design of cache-oblivious B-trees [4, 9], cache-oblivious dynamic dictionaries [5], and packed arrays [6]. In these results, algorithms for the online labeling problem are used to implement buffers that utilize caches well.

Recently, Emek and Korman [15] established a connection between the online labeling and the distributed controller problem introduced in [1]. In distributed controller problem, nodes in an asynchronous distributed network receive requests from outside the network for units of a limited resource, and issue usage permits in response to the requests. The number of permits issued may not exceed the total resource supply, and the protocol must also ensure that no request is declined until the number of permits committed exceeds a $1 - \varepsilon$ fraction of the supply. Protocols with message complexity $O(n \log^2(n))$ on n-node networks are known (e.g. [1, 19]). Emek and Korman showed that the online labeling problem can be reduced to distributed controller problem. They noted that, using their reduction,

a matching $\Omega(n \log^2(n))$ lower bound would follow from an $\Omega(n \log^2(n))$ lower bound on the online labeling problem. We provide such lower bound thus implying the optimality of known upper bounds.

1.2 Definitions

We first define the deterministic version of online labeling problem. We have parameters $n \leq m < r$, and are given a sequence of n numbers from the set $U = \{1, \ldots, r\}$ and must assign to each of them a label in the range $\{1, \ldots, m\}$. This is the only meaningful setting of parameters, as for $n > m$ we have insufficient number of labels for all inserted items and for $r \leq m$ the problem is trivial.

A *deterministic* online labeling algorithm $\mathcal{A}$ with parameters (n, m, r) is an algorithm that on input sequence $(y_1, y_2, \ldots, y_t)$ with $t \leq n$ of distinct elements from U outputs a *labeling* $\mathbf{f}_{\mathcal{A}} : \{y_1, y_2, \ldots, y_t\} \to [m]$ that respects the natural ordering of $y_1, \ldots, y_t$, that is for any $x, y \in \{y_1, y_2, \ldots, y_t\}$, $\mathbf{f}_{\mathcal{A}}(x) < \mathbf{f}_{\mathcal{A}}(y)$ if and only if $x < y$. We refer to $y_1, y_2, \ldots, y_t$ as *items*. We write Y_t for the set $\{y_1, y_2, \ldots, y_t\}$.

Fix an algorithm $\mathcal{A}$. Any item sequence $y_1, \ldots, y_n$ determines a sequence $\mathbf{f}_{\mathcal{A},0}, \mathbf{f}_{\mathcal{A},1}, \ldots, \mathbf{f}_{\mathcal{A},n}$ of labelings (or allocations in case of file maintenance problem formulation) where $\mathbf{f}_{\mathcal{A},t}$ is the labeling of $(y_1, \ldots, y_t)$ determined by $\mathcal{A}$ immediately after y_t was presented. When the algorithm $\mathcal{A}$ is fixed we omit the subscript $\mathcal{A}$. We say that $\mathcal{A}$ *relabels* $y \in Y_t$ *at step* t if $\mathbf{f}_{t-1}(y) \neq \mathbf{f}_t(y)$. In particular, y_t is relabeled at step t. $\mathbf{Rel}_{\mathcal{A},t}$ denotes the set of items relabeled at step t. The cost of $\mathcal{A}$ on $y_1, y_2, \ldots, y_n$ is $\chi_{\mathcal{A}}((y_1, \ldots, y_n), m, r) = \sum_{t=1}^{n} |\mathbf{Rel}_t|$. If $r \geq 2^n - 1$ parameter r is omitted as it does not influence the cost of the algorithm.

A *randomized* online labeling algorithm $\mathcal{A}_r$ is a probability distribution on deterministic online labeling algorithms. Given an item sequence $y_1, \ldots, y_n$, the algorithm $\mathcal{A}_r$ determines a probability distribution over sequences of labelings $\mathbf{f}_0, \ldots, \mathbf{f}_n$. The set $\mathbf{Rel}_{\mathcal{A}_r,t}$ is a random variable whose value is a subset of $y_1, \ldots, y_t$. The cost of $\mathcal{A}_r$ on $y_1, y_2, \ldots, y_n \in U$ is the expected cost $\chi_{\mathcal{A}_r}((y_1, \ldots, y_n), m) = \mathbf{E}[\chi_{\mathcal{A}}((y_1, \ldots, y_n), m)]$.

The maximum cost $\chi_{\mathcal{A}_r}((y_1, \ldots, y_n), m)$ over all sequences $y_1, \ldots, y_n$ is denoted $\chi_{\mathcal{A}_r}(n, m)$. We write $\chi(n, m)$ for the smallest cost $\chi_{\mathcal{A}_r}(n, m)$ that can be achieved by any algorithm $\mathcal{A}_r$ with range m.

Thorough the thesis we work with various vectors and sequences that change over time. We use subscript to denote the time step and we use $(\cdot)$ notation to denote a particular coordinate of such a vector or sequence at that step.

1.3 Label Space vs. Array Notation

It is illuminating to think of the online labeling problem in terms of the file maintenance problem mentioned earlier. In this reformulation the label space $\{1, \ldots, m\}$ is associated to an array indexed by $\{1, \ldots, m\}$ and an item labeled by j is viewed as stored in location j.

With the array picture in mind, we call an interval of labels a *segment*, and say that a label is *occupied* if there is a item assigned to it. By segment of the array of size m (or the label space) we denote the subinterval of $\{1, \ldots, m\}$ (i.e. the indices of cells of the array). Let S be an arbitrary segment in array of size m. By $\min(S)$ and $\max(S)$ we denote the index of leftmost and rightmost cell of the segment. By the $|S|$ we denote the length of the segment i.e. $\max(S) - \min(S) + 1$.

1.4 Organization of the Thesis

The rest of the thesis consists of two parts.

Part I focuses on the upper bounds for the online labeling problem. We provide description of two deterministic algorithms which together cover almost all possible array sizes and achieve asymptotically optimal time complexity. Our main contribution in Part I is the optimal algorithm for arrays of superpolynomial size which was a joint work with Michal Koucký and Michael Saks.

In Part II we describe tight lower bounds for the online labeling problem. This part is mainly based on our results which were a joint work with Michal Koucký, Michael Saks, Martin Babka and Vladimír Čunát. With a small exception (see Chapter 6), these results are superior to previous work. We prove tight lower bounds for the deterministic version of the online labeling problem for arbitrary m assuming that the size of the universe of items is exponential. We provide the first nontrivial lower bound for arrays of linear size under the assumption of limited universe of items (tight). Finally we prove the first nontrivial lower bound for the randomized algorithms which is tight for arrays of size $n^{1+\varepsilon}$ where ε is a constant greater than zero.

Part I

Online Labeling Problem Upper Bounds

2. Introduction

In this part we present two algorithms. The first one is optimal for arrays of small size, and the second one is optimal for arrays of large size. Together they cover almost all array sizes except for arrays of size $m \in [n^{\omega(1)}, n^{\log n})$ where no non-trivial upper bound is known.

A trivial algorithm can use the same approach as the Insertion sort [18]. First we determine a position of the new item. Then we move all items inserted so far which are behind its position by one cell for the asymptotic cost $O(n)$. Thus we obtain one empty cell into which the new item can be inserted.

Notice, that such approach does not take any advantage of arrays of size larger than n. To make better (or any) use of the additional space, our algorithms follow two basic concepts. First we maintain gaps (free space) between already inserted items and second we try to keep these gaps spread as evenly as possible.

Both our algorithms work in rounds. At the very beginning of the round the algorithm is provided with the next item y to be inserted. Then the algorithm finds out where y should be inserted. Then it finds the *suitable* part of the array (*segment*), which contains this point and finally it rearranges the items in this segment (including y) as evenly as possible.

2.1 Previous work

The very first non-trivial algorithm for solving the online labeling problem was given by Itai, Konheim and Rodeh [17] who developed this algorithm to solve the priority queue problem. They achieved $O(n \log^2(n))$ time complexity for arrays of size $O(n)$ in the amortized setting. This algorithm was simplified by Itai and Katriel [16]. However the analysis of the algorithm became more complicated. Another improvement was done by Willard [20], who achieved the same time complexity but in the worst case settings (i.e. each insert has time complexity at most $\log^2(n)$). His result was then simplified by Bender et al. [3]. In the special case of $m = n$, an algorithm with cost $O(n \log^3(n))$ in amortized settings was first developed by Zhang [21] and then it was simplified by Bird and Sadnicky [7]. This result is surprising since when you insert last few items the array contains almost no gaps.

It is also interesting that none of these works considers the case of limited universe. As we already mentioned, if $r \leq m$ the online labeling problem is trivial. A natural question is whether for r say less than $10m$ the problem is significantly easier. We will show in Chapter 9 that in the case of linear array size you cannot benefit from the small universe.

3. Online Labeling Problem in Small Arrays

3.1 Introduction

In this chapter, we describe an algorithm $\mathcal{A}^{small}$ for the online labeling problem which achieves an asymptotically optimal time complexity $O(n \cdot \frac{\log^2(m)}{\log(m)-\log(n)})$ for arrays of size $m \in [\frac{4}{3}n, n^c]$ where c is a constant greater than one and its modification, algorithm $\mathcal{A}^{tiny}$, which achieves an asymptotically optimal time complexity $O(n \cdot \log^2(m) \log\left(\frac{m}{m-n}\right))$ for arrays of size $m \in [n, \frac{4}{3}n]$. This matches (up to a constant factor) the lower bounds given in Chapters 8 and 9.

The algorithm $\mathcal{A}^{small}$ is a modification of the algorithm by Itai, Konheim and Rodeh [17]. However, unlike the original algorithm, $\mathcal{A}^{small}$ is optimal even for arrays larger than linear size. The existence of such modification was known as a folklore, however, to our knowledge it was not published before. Furthermore, we provide a more detailed analysis which handles rounding issues during the rearrange of items. The result by Itai and Katriel [16] provides slightly simpler algorithm with the same time complexity, however, its analysis is more complicated and also we are not aware of its extension for superlinear arrays. Finally, there is the result by Willard [20] (and consequently by Bender et al. [3]) who provides an algorithm for the linear size arrays. Willard's algorithm works even with the worst case guarantee while $\mathcal{A}^{small}$ achieves the optimal cost per insert only in the amortized settings.

The algorithm $\mathcal{A}^{tiny}$ is a reformulation of the result by Zhang [21]. We iteratively use the $\mathcal{A}^{small}$ on the groups of items instead of items itself.

This chapter is organized as follows. In Section 3.3 we state the main theorem of the chapter, which will be an immediate consequence of Lemmas 3.6.1 and 3.8.1. Lemma 3.6.1 is proved by the construction of algorithm $\mathcal{A}^{small}$ from Figure 3.1 which is done in Sections 3.2 to 3.7. The modification of $\mathcal{A}^{small}$, algorithm $\mathcal{A}^{tiny}$, and associated Lemma 3.8.1 is in Section 3.8.

3.2 Algorithm Outline

Let us recall that each online labeling algorithm works in rounds. During each round the algorithm is given a next item y to be inserted, then it moves some of the already inserted items and finally it inserts y. Thus we only have to decide which items will be moved during each round. From the high level perspective we make this decision as follows. After we obtain y we determine where it should be

inserted. Then we find the smallest segment of the array which contains this point and whose density of items in it is smaller than a certain threshold defined for segments of various sizes. Finally we rearrange items in this segment (including y) as evenly as possible.

3.3 Main Result

In this section we state the main result of this chapter.

Theorem 3.3.1 (Main Theorem)**.** *Let n, m be integers such that $n < m$. Then there exist constants C_0 and C_1 and algorithms $\mathcal{A}^{small}$ and $\mathcal{A}^{tiny}$ such that*

1. $\chi_{\mathcal{A}^{small}}(n, m) \leq C_0 \cdot n \cdot \frac{\log^2(m)}{\log(m)-\log(n)}$ *if* $n < \frac{3}{4}m$*, and*
2. $\chi_{\mathcal{A}^{tiny}}(n, m) \leq C_1 \cdot n \cdot \log^2(m) \log\left(\frac{m}{m-n}\right)$ *otherwise.*

This theorem is an immediate implication of Lemma 3.6.1 in which we show the complexity of $\mathcal{A}^{small}$ and Lemma 3.8.1 in which we show the complexity of $\mathcal{A}^{tiny}$.

Notice that for $m \in [n, n^2]$ this matches the lower bounds given in Chapters 8 and 9 as $\log(n) \in \Theta(\log(m))$ for such m.

3.4 Definitions

Let S be a segment of the array. Using the notation from Section 1.2 and given an algorithm $\mathcal{A}$ we denote by $\mathbf{items}_{\mathcal{A},t}(S)$ the set of $y \in Y_t$ such that $\mathbf{f}_{\mathcal{A},t}(y) \in S$. The *density* of the segment S is defined as $\rho_{\mathcal{A},t}(S) = \frac{|\mathbf{items}_{\mathcal{A},t}(S)|}{|S|}$. Recall, that $Y_t = \{y_1, y_2, \ldots, y_t\}$ and for each segment of array S its length $|S|$ is defined as $|S| = \max(S) - \min(S) + 1$.

If the algorithm $\mathcal{A}$ is obvious from the context we omit it from the subscript.

3.5 Segment Hierarchy

Recall that we insert n items to the array of size m. Let m' be the greatest power of 2 such that $m' \leq m$. Let $\mathbf{height} = \log(m') = \lfloor \log(m) \rfloor$.

The segment hierarchy is a hierarchy of segments $S_\ell(i)$ for $\ell \in \{0, \ldots, \mathbf{height}\}$ and $i \in \left\{0, \ldots, 2^\ell - 1\right\}$. We say that $S_\ell(i)$ is the i-th segment of the ℓ-th level. We first define segments $S_{\mathbf{height}}(i)$ as follows:

1. if $i < m - m'$ then we set $S_t(i) = \{2i, 2i + 1\}$

2. otherwise we set $S_t(i) = \{i + m - m'\}$

This ensures that the number of segments in the last level is power of two which simplifies some proofs later. Let us note that the way we choose which segments are of size two and which are of size one can be arbitrary (as long as we preserved their numbers).

The remaining levels of segment hierarchy are defined as follows. Let us assume that we have already defined all segments of the level $\ell + 1$. Then for $i \in \{0, \ldots, 2^\ell - 1\}$ we set $S_\ell(i) = S_{\ell+1}(2i) \cup S_{\ell+1}(2i+1)$.

Notice that the segment hierarchy can be seen as a binary tree where each segment (except those on the last level) has two subsegments and each segment has one parent (except the one on the 0-th level).

Now we show that two consecutive segments on one level of segment hierarchy cannot differ too much in their length.

Claim 3.5.1. *For each $\ell \in \{0, \ldots, \mathbf{height}\}$ and arbitrary $i \in \{0, \ldots, 2^\ell - 1\}$ it holds that*

$$\frac{3}{2} \cdot |S_{\ell+1}(2i)| \leq |S_\ell(i)| \leq 3 \cdot |S_{\ell+1}(2i)|, \text{ and}$$
$$\frac{3}{2} \cdot |S_{\ell+1}(2i+1)| \leq |S_\ell(i)| \leq 3 \cdot |S_{\ell+1}(2i+1)|.$$

Proof. It follows from the definition of the segment hierarchy that

1. $|S_{\ell+1}(2i)| \geq |S_{\ell+1}(2i+1)|$
2. $2 \geq \frac{|S_{\ell+1}(2i)|}{|S_{\ell+1}(2i+1)|}$.

Finally since $|S_\ell(i)| = |S_{\ell+1}(2i)| + |S_{\ell+1}(2i+1)|$ the proof follows. □

Finally, let us emphasize that the segment hierarchy is not built explicitly by the algorithm $\mathcal{A}^{small}$ and is only used for the exposition of the algorithm $\mathcal{A}^{small}$. It is easy to observe that borders of each segment of the segment hierarchy may be calculated in constant time if ℓ and i are given.

3.6 Algorithm Construction

In this section we define the algorithm $\mathcal{A}^{small}$ with the properties we described above in Theorem 3.3.1, part 1, and we state Lemma 3.6.1 which implies the Theorem 3.3.1.

The algorithm $\mathcal{A}^{small}$ determines for each inserted item y_t a segment in which the items are distributed as evenly as possible (including y_t). To achieve this, we define a *level threshold density* for $\ell \in \{0, \ldots, \mathbf{height}\}$ as follows:

$$\hat{\rho}(\ell) = \frac{n}{m} \cdot \left(\frac{m}{n}\right)^{\frac{\ell}{\mathbf{height}}}.$$

Now we can find so called *critical segment* Q_t which is the smallest segment in the segment hierarchy which contains the position where we want to insert y_t and whose density (with respect to t) is smaller than the threshold density of its level. Finally, we rearrange all the items in the critical segment (including y_t) as evenly as possible.

We say that segment S was *rebuilt* at step t if it holds that $S \subset Q_t$ (i.e. not Q_t itself). All segments are considered to be rebuilt at step 0.

Prior to an algorithm pseudocode exhibition, we recall that $\mathbf{items}_t(S)$ denotes the set of items stored in segment S at t-th step and $\rho_t(S)$ denotes the density of items in S.

The pseudocode of the algorithm is in Figure 3.1. Now we can state the main lemma for the algorithm $\mathcal{A}^{small}$.

Lemma 3.6.1. *Let n, m be integers such that $n < m$. Let $\mathcal{A}^{small}$ be an algorithm from Figure 3.1 with the parameters n, m.*

Then

$$\chi_{\mathcal{A}^{small}}(n, m) \leq 6 \cdot n \cdot \frac{\log^2(m)}{\ln(m) - \ln(n)}.$$

Prior to proving the lemma itself we state a couple of useful claims. The proof then can be found on page 16.

3.7 Proof of Lemma 3.6.1

The first claim shows the relation between critical segments in different steps. It is an immediate consequence of the fact that critical segments are chosen from the segment hierarchy.

Claim 3.7.1. *For each $t, t' \in \{1, \ldots, n\}$ only one of the following cases is possible:*

1. $Q_t \subseteq Q_{t'}$,
2. $Q_{t'} \subset Q_t$,
3. $Q_t \cap Q_{t'} = \emptyset$.

Algorithm $\mathcal{A}^{small}(n, m)$

- $m' \longleftarrow$ the greatest power of 2 which is at most m
- For $t = 1 \ldots n$ do:
  - $y_t \longleftarrow$ next item to be inserted
  - $pos \longleftarrow \mathbf{f}_{t-1}(y)$ where y is the smallest item from Y_{t-1} such that $y \geq y_t$ or m if such y does not exist.

  {***Critical Segment Choice***}
  - $\ell \longleftarrow$ **height**
  - $i \longleftarrow j$ such that $S_{\mathbf{height}}(j)$ contains pos
  - do
    - $\ell \longleftarrow \ell - 1$
    - $i \longleftarrow \lfloor i/2 \rfloor$
  - while $\left(\frac{|\mathbf{items}_{t-1}(S_\ell(i))|+1}{|S_\ell(i)|} > \hat{\rho}(\ell)\right)$
  - $Q_t \longleftarrow S_\ell(i)$
  - $\mathbb{S} \longleftarrow \mathbf{items}_{t-1}(Q_t) + \{y_t\}$

  {***Preserve Labels Outside of*** Q_t}
  - foreach y in $Y_t \setminus \mathbb{S}$ do
    - $\mathbf{f}_t(y) = \mathbf{f}_{t-1}(y)$

  {***Rearrange Evenly Items in*** Q_t}
  - $\rho \longleftarrow \frac{|\mathbb{S}|}{|Q_t|}$
  - foreach y in $\mathbb{S}$ do
    - $ord \longleftarrow$ order of y in $\mathbb{S}$
    - $\mathbf{f}_t(y) \longleftarrow \left\lfloor \frac{ord-1}{\rho} \right\rfloor + \min(Q_t)$
- Output: $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_n$

Figure 3.1: Pseudocode for the algorithm $\mathcal{A}^{small}$

A next claim is also an immediate consequence of the definition of algorithm $\mathcal{A}^{small}$. It upper bounds the cost of a single step t using the size of the critical segment Q_t and the threshold densities.

Claim 3.7.2. *For each $t \in \{1, \ldots, n\}$, if i, ℓ are integers such that $S_\ell(i) = Q_t$ then*

$$\mathbf{Rel}_t \leq \hat{\rho}(\ell) \cdot |S_\ell(i)| = \hat{\rho}(\ell) \cdot |Q_t|.$$

Before we state a next claim we define a *paying set at step t* which is the set of items inserted to the subsegment S of Q_t (the greatest one which would contain y_t) since S was rebuilt last time. We use them to define the items among which we distribute the cost of step t.

More formally, consider an arbitrary step $t \in \{1, \ldots, n\}$ and the corresponding critical segment Q_t. Let ℓ and i be chosen so that $S_\ell(i) = Q_t$. Let $S_{\ell+1}(j)$ be chosen among $S_{\ell+1}(2i)$ and $S_{\ell+1}(2i+1)$ using the one which would contain y_t unless the rebuild occurred. Let $t' \in \{0, \ldots, t-1\}$ be the greatest such that $S_{\ell+1}(j)$ was rebuilt at t'. By the paying set at time t, $\mathbb{P}_t$, we denote a subset of $\{y_{t'+1}, y_{t'+2}, \ldots, y_t\}$ such that for each $y \in \mathbb{P}_t$ it holds that $\mathbf{f}_{t-1}(y) \in S_{\ell+1}(j)$ or $y = y_t$.

The following claims show that each inserted item y can only take place in limited number of paying sets.

Claim 3.7.3. *For each $t, t' \in \{1, \ldots, n\}$ such that $t' < t$ and item y such that it was inserted during the steps $\{1, \ldots, t'\}$, if $y \in \mathbb{P}_{t'}$ and $y \in \mathbb{P}_t$, then $Q_{t'} \subset Q_t$.*

Proof. First notice that each subset of $Q_{t'}$ was rebuilt (by definition). Thus the smallest segment of segment hierarchy which was not rebuilt and which contains y is $Q_{t'}$. Thus if $y \in \mathbb{P}_t$ it follows that $Q_t \not\subseteq Q_{t'}$ since otherwise $\mathbb{P}_t$ would be subset of $y_{t'+1}, y_{t'+2}, \ldots, y_t$ but y was inserted prior to $y_{t'+1}$.

Now, let us assume for the sake of contradiction that $Q_t \cap Q_{t'} = \emptyset$. Then since $y \notin \mathbf{items}_t(Q_t)$ we could infer that $y \notin \mathbb{P}_t$.

Thus the only remaining possibility is $Q_{t'} \subset Q_t$ (Claim 3.7.1) which implies the claim. □

A next claim shows, that just after the segment is rebuilt at step t the density of this segment is limited by the density of Q_t.

Claim 3.7.4. *Let S be a segment which was rebuilt at step t. Then*

$$\rho_t(S) \leq \rho_t(Q_t) + \frac{1}{|S|}.$$

Proof. Let S' be a subsegment of Q_t such that $\min(S') = \min(Q_t)$ (i.e. their left borders are equal). It is easy to show that

$$\rho_t(Q_t) \leq \rho_t(S') \leq \rho_t(Q_t) + \frac{1}{|S'|}.$$

It follows immediately from the way the positions of items in Q_t are chosen during the rearrange (i.e., as evenly as possible) by $\mathcal{A}^{small}$.

In case $\min(S) = \min(Q_t)$ above mentioned fact implies the claim immediately. Thus let us assume that $\min(S) > \min(Q_t)$. Let us choose S' so that $S' = \{\min(Q_t), \ldots, \min(S) - 1\}$ and S'' so that $S'' = \{\min(Q_t), \ldots, \max(S)\}$. From the inequalities from the previous paragraph we can infer the following:

$$\begin{aligned}
\rho_t(S) &= \frac{|\mathbf{items}_t(S'')| - |\mathbf{items}_t(S')|}{|S|} \\
&\leq \frac{\left(\rho_t(Q_t) + \frac{1}{|S''|}\right) \cdot |S''| - \rho_t(Q_t) \cdot |S'|}{|S|} \\
&= \rho_t(Q_t) \cdot \frac{|S''| - |S'|}{|S|} + \frac{1}{|S|} = \rho_t(Q_t) + \frac{1}{|S|}
\end{aligned}$$

□

Now we use this claim to lower bound the size of the paying sets.

Claim 3.7.5. *For each $t \in \{1, \ldots, n\}$ let ℓ, j be chosen so that $S_{\ell-1}(j) = Q_t$. Let i be chosen so that $\mathbb{P}_t \setminus \{y_t\} \subseteq \mathbf{items}_{t-1}(S_\ell(i))$. Then*

$$|\mathbb{P}_t| \geq \frac{1}{2} \cdot (\hat{\rho}(\ell) - \hat{\rho}(\ell - 1)) \cdot |S_\ell(i)|.$$

Proof. Let t' be the greatest such that $t' < t$ and $S_\ell(i)$ was rebuilt at step t'. First notice that

$$|\mathbb{P}_t| = |\mathbf{items}_{t-1}(S_\ell(i))| + 1 - |\mathbf{items}_{t'}(S_\ell(i))|.$$

Next we limit $|\mathbf{items}_{t'}(S_\ell(i))|$. Notice that Claim 3.7.4 implies that

$$\begin{aligned}
|\mathbf{items}_{t'}(S_\ell(i))| &\leq |S_\ell(i)| \cdot \left(\rho_{t'}(Q_{t'}) + \frac{1}{|S_\ell(i)|}\right) \\
&\leq |S_\ell(i)| \cdot \left(\hat{\rho}(\ell - 1) + \frac{1}{|S_\ell(i)|}\right),
\end{aligned}$$

which follows from the fact that $Q_{t'}$ is at most at the level $\ell - 1$. From Critical Segment Choice we can infer that

$$|\mathbf{items}_{t-1}(S_\ell(i))| + 1 > |S_\ell(i)| \cdot \hat{\rho}(i),$$

otherwise the segment $S_\ell(i)$ would not be rebuilt at the step t.

Thus we can derive the following chain of inequalities:

$$|\mathbf{items}_{t-1}(S_\ell(i))| + 1 - |\mathbf{items}_{t'}(S_\ell(i))| \geq |S_\ell(i)| \cdot \left(\hat{\rho}(\ell) - \hat{\rho}(\ell-1) - \frac{1}{|S_\ell(i)|}\right).$$

$$|\mathbf{items}_{t-1}(S_\ell(i))| + 2 - |\mathbf{items}_{t'}(S_\ell(i))| \geq |S_\ell(i)| \cdot (\hat{\rho}(\ell) - \hat{\rho}(\ell-1))$$

And since $S_\ell(i)$ was not rebuilt from t' to t it follows that $|\mathbf{items}_{t-1}(S_\ell(i))| \geq |\mathbf{items}_{t'}(S_\ell(i))|$ and thus we obtain:

$$2 \cdot |\mathbf{items}_{t-1}(S_\ell(i))| + 2 - 2 \cdot |\mathbf{items}_{t'}(S_\ell(i))| \geq |S_\ell(i)| \cdot (\hat{\rho}(\ell) - \rho_{t'}(\hat{\rho}(\ell-1))),$$

which implies the claim immediately. □

Having all these claims we are ready to prove Lemma 3.6.1. In the next proof we first define how we distribute the cost of each step t among the items from the paying set at step t. Then we show that the overall sum of costs assigned to one item is at most $O\left(\frac{\log(n)^2}{\log(m)-\log(n)}\right)$.

Proof of Lemma 3.6.1. Let y be an arbitrary item from Y_n. By $c(y)$ we denote the cumulative cost assigned to the item y during the whole algorithm run. At the very beginning we set $c(y) = 0$ for each $y \in Y_n$. Then at each step $t \in \{1, \ldots, n\}$, if $y \in \mathbb{P}_t$ we set $c(y) = c(y) + \frac{\mathbf{Rel}_t}{|\mathbb{P}_t|}$ and we left it unchanged otherwise. Recall that $\mathbf{Rel}_t$ denotes the set of relabeled items at step t.

It is easy to verify that $\chi(n, m) = \sum_{y \in Y_t} c(y)$. Thus if we limit $c(y)$ for each y we are done.

Let ℓ, i be chosen so that $S_\ell(i) = Q_t$. Let j be chosen so that $\mathbb{P}_t \setminus \{y_t\} \subseteq \mathbf{items}_{t-1}(S_{\ell+1}(j))$. First we compute the maximal possible increase of $c(y)$at one step by the following chain of inequalities:

$$\begin{aligned}
\frac{\mathbf{Rel}_t}{|\mathbb{P}_t|} &\leq \frac{\hat{\rho}(\ell)|S_\ell(i)|}{\frac{1}{2} \cdot (\hat{\rho}(\ell+1) - \hat{\rho}(\ell)) \cdot |S_{\ell+1}(j)|} \text{ (Claim 3.7.2, Claim 3.7.5)} \\
&\leq \frac{2 \cdot 3\hat{\rho}(\ell)}{(\hat{\rho}(\ell+1) - \hat{\rho}(\ell))} \text{ (Claim 3.5.1)} \\
&= \frac{6 \cdot \frac{n}{m} \cdot \left(\frac{m}{n}\right)^{\frac{\ell}{\mathbf{height}}}}{\frac{n}{m} \cdot \left(\frac{m}{n}\right)^{\frac{\ell+1}{\mathbf{height}}} - \frac{n}{m} \cdot \left(\frac{m}{n}\right)^{\frac{\ell}{\mathbf{height}}}} = \frac{6}{\left(\frac{m}{n}\right)^{\frac{1}{\mathbf{height}}} - 1} \\
&= \frac{6}{e^{\ln\left(\frac{m}{n}\right) \cdot \frac{1}{\mathbf{height}}} - 1} \leq \frac{6 \cdot \mathbf{height}}{\ln m - \ln n},
\end{aligned}$$

where in the last inequality we used the fact that $e^x - 1 \geq x$.

Now it remains to limit the number of increases for each item. Claim 3.7.3 implies that the maximal number of paying sets in which each item can take place is limited by **height**. This however implies that number of increases for each item is limited also by **height**. The lemma follows. □

3.8 Modification of Algorithm $\mathcal{A}^{small}$ for Very Small Arrays

In this section we focus on the case when $n > \frac{3}{4}m$, i.e., the case when the array is very small in comparison to the number of inserted items. We present a reformulation of the approach introduced by Zhang [21] in his thesis. Our reformulation will however use a notation consistent with the rest of thesis.

The basic idea is to use algorithm $\mathcal{A}^{small}$ (Figure 3.1) for arrays of linear size iteratively. In each round we fill one half of the remaining empty cells. Then we show that the cost of each round is roughly $O(n \log^2(n))$ while it is obvious that number of rounds is at most $\log(n)$.

Let $\mathcal{A}^{small}$ denote the **Algorithm** from Figure 3.1. We use algorithm $\mathcal{A}^{small}$ as follows. Let n be the number of items we want to insert in current round and n_0 be the number of already inserted items. First we choose $2n$ items among the n_0 already inserted items as evenly as possible. We denote this set as Y'. Then we simulate $\mathcal{A}^{small}$ with the parameters $(2n, 4n)$ using the set Y' as an input (in arbitrary order) so we obtain an allocation $\mathbf{f}_{\mathcal{A}^{small},2n}$. We rearrange the items in the original array so that for each pair of consecutive items from Y' there is the same number of empty cells between them as in the allocation $\mathbf{f}_{\mathcal{A}^{small},2n}$ (placing the remaining items arbitrarily between them so that the ordering is preserved). Notice that after this step the number of items stored between each pair of consecutive items from Y' is roughly $\frac{n_0}{2n}$.

Then after each consequent insert we add the newly arriving item y_t to the Y' (in the end there will be $3n$ items in Y'). We simulate insert of y_t by $\mathcal{A}^{small}$ and we rearrange the items in the original array so that for each pair of consecutive items from Y' there is same number of empty cells between them as in the allocation $\mathbf{f}_{\mathcal{A}^{small},t}$. Notice that the number of items between each pair of consecutive items from Y' is still at most $\frac{n_0}{2n}$. Thus the cost of the newly constructed algorithm is at most $\frac{n_0}{2n}$ times greater than the cost of $\mathcal{A}^{small}$, while the number items inserted during each round decreases exponentially.

Let us start with the main lemma of the section which together with Lemma 3.6.1 implies Theorem 3.3.1.

Lemma 3.8.1. *Let n, m be integers such that $n \geq \frac{3}{4}m$ and $n \leq m$. Then there exists algorithm $\mathcal{A}^{tiny}$ such that*

$$\chi_{\mathcal{A}^{tiny}}(n, m) \leq 62 \cdot m \log^2(m) \left\lceil \log\left(\frac{m}{m-n}\right) \right\rceil.$$

The constants are chosen for ease of exposition and can certainly be improved. Notice that this is asymptotically equal to $O\left(n \log^2(n) \log\left(\frac{m}{m-n}\right)\right)$. For n almost equal to m this simplifies to $O(n \log^3(n))$.

To prove this lemma with first prove a claim which deals with one round of the algorithm $\mathcal{A}^{tiny}$. Let $\chi_{\mathcal{A}^{tiny}}(n, m|n_0)$ denote the maximal cost of the algorithm $\mathcal{A}^{tiny}$ which inserts n items to the array of size m assuming that n_0 items was already inserted.

Claim 3.8.2. *Let n_0, m be integers such that $n_0 \geq \frac{3}{4}m$ and $n_0 < m$. Then there exists Algorithm $\mathcal{B}$ such that*

$$\chi_{\mathcal{B}}\left(\left\lfloor \frac{m-n_0}{2} \right\rfloor, m|n_0\right) \leq 62 \cdot m \log^2(m)$$

Proof. To proof this claim we construct algorithm $\mathcal{B}$. Recall, that $\mathcal{A}^{small}$ is the algorithm from Figure 3.1. For the sake of analysis we define $\min_U = 0$ and $\max_U = 2^m + 1$ i.e. the items which are smaller and greater than any item in the universe. Without loss of generality we assume that $\mathcal{A}^{small}$ assigns $\mathbf{f}_{\mathcal{A}^{small},t}(\min_U) = 0$ and $\mathbf{f}_{\mathcal{A}^{small},t}(\max_U) = m + 1$ and $\mathbf{f}_{\mathcal{B},t}(\min_U) = 0$ and $\mathbf{f}_{\mathcal{B},t}(\max_U) = m + 1$ for each step t. Let $n = \left\lfloor \frac{m-n_0}{2} \right\rfloor$ denote the actual number of inserted items. Let $(y, y')-$subinterval of set Y be a subinterval of Y which contains all $y'' \in Y$ such that $y \leq y'' \leq y$. Let $y_{n_0+1}, y_{n_0+2}, \ldots, y_{n_0+n}$ be the sequence of inserted items. Let $Y_{\mathcal{B},n_0}$ denote the set of items inserted in the array before the algorithm starts. Let $Y_{\mathcal{B},t}$ be defined as $Y_{\mathcal{B},n_0} \cup \{y_{n_0+1}, y_{n_0+2}, \ldots, y_t\}$. Let $Y_{\mathcal{A}^{small},t}$ be defined as follows:

- for $t = n_0$ we define $Y_{\mathcal{A}^{small},t}$ to be a subset of $Y_{\mathcal{B},n_0}$ of size $m - n_0$ which is chosen as evenly as possible among $Y_{\mathcal{B},n_0}$. In other words for each pair $y, y' \in Y_{\mathcal{A}^{small},t} \cup \{\min_U, \max_U\}$ such that $y < y'$ and no items is between them in $Y_{\mathcal{A}^{small},t} \cup \{\min_U, \max_U\}$ let I be $(y, y')-$subinterval of $Y_{\mathcal{B},t} \cup \{\min_U, \max_U\}$. Then $\left\lfloor \frac{n_0}{m-n_0+1} \right\rfloor \leq |I| \leq \left\lceil \frac{n_0}{m-n_0+1} \right\rceil$.
- for $t > n_0$ we set $Y_{\mathcal{A}^{small},t} = Y_{\mathcal{A}^{small},n_0} \cup \{y_{n_0+1}, y_{n_0+2}, \ldots, y_t\}$.

We define the allocation $\mathbf{f}_{\mathcal{B},n_0}$ to be an allocation obtained by application of Rearranging Procedure (Figure 3.2) with parameters $I = Y_{\mathcal{A}^{small},n_0}$ and $y^{min} = \min_U$. The cost of the procedure with such parameters is at most n_0. Consider the sequence of allocations $\mathbf{f}_{\mathcal{A}^{small},t}$ obtained by $\mathcal{A}^{small}(m - n_0 + n, m - n_0 + 2n)$ on input $(y_1, y_2, \ldots y_{m-n_0}) \cup (y_{n_0}, \ldots, y_{n_0+n})$ where sequence $(y_1, y_2, \ldots y_{m-n_0})$ is an arbitrary sequence of all items from $Y_{\mathcal{A}^{small},n_0}$.

Then the allocations $\mathbf{f}_{\mathcal{B},n_0}, \mathbf{f}_{\mathcal{B},n_0+1}, \ldots, \mathbf{f}_{\mathcal{B},n_0+n}$ are obtained as follows. Let y^{min} be the greatest from $Y_{\mathcal{A}^{small},t} \cup \{\min_U, \max_U\}$ such that $y^{min} < \min(\mathbf{Rel}_{\mathcal{A}^{small},t})$. Similarly we define y^{max} to be the smallest from $Y_{\mathcal{A}^{small},t} \cup \{\min_U, \max_U\}$ such that $y^{max} > \max(\mathbf{Rel}_{\mathcal{A}^{small},t})$. Let $I = (y^{min}, y^{max})-$subinterval of $Y_{\mathcal{B},t}$. Then for each $y \in (Y_{\mathcal{B},t} \setminus I) \cup \{y^{min}, y^{max}\}$ we set $\mathbf{f}_{\mathcal{B},t}(y) = \mathbf{f}_{\mathcal{B},t-1}(y)$. For the items in I we use a Rearranging Procedure from Figure 3.2 with parameters (I, t, y^{min}).

Rearranging Procedure(I, t, y^{min})

- $pos \longleftarrow \mathbf{f}_{\mathcal{B},t}(y^{min})$
- $y^{last} \longleftarrow y^{min}$
- **for** $i \in \{2, \ldots, |I| - 1\}$ **do**
  - **y** $\longleftarrow$ i**-th smallest from** I
  - **if** $y \in Y_{\mathcal{A}^{small},t}$ **then**
    - $pos \longleftarrow pos + \mathbf{f}_{\mathcal{A}^{small},t}(y) - \mathbf{f}_{\mathcal{A}^{small},t}(y^{last}) + 1$
    - $y^{last} \longleftarrow y$
  - **else**
    - $pos \longleftarrow pos + 1$
  - $\mathbf{f}_{\mathcal{A}^{tiny},t}(y) \longleftarrow pos$

Figure 3.2: Pseudocode for Rearranging Procedure

Since it is easy to see that such defined allocation is correct it remains to compare sizes $|\mathbf{Rel}_{\mathcal{A}^{small},t}|$ and $|\mathbf{Rel}_{\mathcal{B},t}|$. Let for each $t \in \{n_0, \ldots, n_0 + n\}$ be y, y' items from $Y_{\mathcal{A}^{small},t} \cup \{\min_U, \max_U\}$ such that no item is between them in $Y_{\mathcal{A}^{small},t} \cup \{\min_U, \max_U\}$. Let I be a (y, y')-subinterval of $Y_{\mathcal{B},t} \cup \{\min_U, \max_U\}$. Then it is obviously true that $|I| \leq \left\lceil \frac{n_0}{m - n_0 + 1} \right\rceil$. Thus Rearranging Procedure implies that

$$|\mathbf{Rel}_{\mathcal{B},t}| \leq \left\lceil \frac{n_0}{m - n_0 + 1} \right\rceil \cdot (|\mathbf{Rel}_{\mathcal{A}^{small},t}| + 1).$$

Combining these facts we infer the following chain of inequalities:

$$\begin{aligned}
n_0 + \sum_{t=n_0+1}^{n+n_0} \mathbf{Rel}_{\mathcal{B},t} &\leq \left\lceil \frac{n_0}{m - n_0 + 1} \right\rceil \cdot \sum_{t=n_0+1}^{n+n_0} (|\mathbf{Rel}_{\mathcal{A}^{small},t}| + 1) + n_0 \\
&\leq 4 \cdot \frac{n_0}{m - n_0} \cdot \sum_{t=n_0+1}^{n+n_0} |\mathbf{Rel}_{\mathcal{A}^{small},t}| + n_0 \\
&\leq 4 \cdot \frac{n_0}{m - n_0} \cdot (\chi_{\mathcal{A}^{small}}(m - n_0 + n, m - n_0 + 2n) - m + n_0) + n_0 \\
&\leq 4 \cdot \frac{n_0}{m - n_0} \cdot \chi_{\mathcal{A}^{small}}(m - n_0 + n, m - n_0 + 2n) \\
&\leq 4 \cdot 6 \cdot \frac{n_0(m - n_0 + n)}{m - n_0} \cdot \frac{\log^2(m - n_0 + 2n)}{\ln(m - n_0 + 2n) - \ln(m - n_0 + n)}.
\end{aligned}$$

Now since $m - n_0 > 2n$ we can infer

$$n_0 + \sum_{t=n_0+1}^{n+n_0} \leq 12 \cdot \frac{3}{2} n_0 \cdot \frac{\log^2(2(m-n_0))}{\ln(\frac{4}{3})} \leq 62 \cdot m \log^2(m).$$

□

Now we proceed with the proof of Lemma 3.8.1.

Proof of Lemma 3.8.1. The algorithm $\mathcal{A}^{tiny}$ proceeds as follows. First, it inserts $\frac{3}{4}m$ items using algorithm $\mathcal{A}^{small}$ at the cost $\chi_{\mathcal{A}^{small}}\left(\frac{3}{4}m, m\right)$. Then we apply Claim 3.8.2 $r = \left\lceil \log\left(\frac{m}{m-n}\right)\right\rceil$ times. After these r rounds

$$\sum_{i=1}^{r} \frac{1}{4} \cdot \frac{m}{2^i} = \frac{m}{4} - \frac{m}{4 \cdot 2^r} \geq \frac{m}{4} - \frac{1}{4}(m-n) = \frac{n}{4}$$

items were inserted. Together with $\frac{3}{4}m$ items inserted in the first step, this is more than n, while the cost per each round is smaller than $62m \log^2(m)$. Since the number of rounds is at most $\left\lceil \log\left(\frac{m}{m-n}\right)\right\rceil$ the lemma follows. □

4. Online Labeling Problem in Large Arrays

4.1 Introduction

In this chapter we present an algorithm for the online labeling problem which achieves asymptotically optimal time complexity $O(n \cdot \frac{\log n}{\log\log m - \log\log n})$ for the label space $m \in [n^{\log n}, 2^n]$ where c is a constant greater than one. This matches the lower bound from Chapter 6, however leaving a small gap for arrays of size $m \in [n^{\omega(1)}, n^{\log n})$ where no nontrivial upper bound is known.

Prior to our joint work with Michal Koucký and Michael Saks [10] (on which this chapter is based), to the best of our knowledge, no algorithm was published for this range of array sizes.

4.2 Algorithm Outline

First notice that we can store $\log(m)$ items for the cost of 1 per item into an array of size m. In such a case there is no need to relabel any item. This idea can be iterated further to insert $\frac{1}{4} \cdot \frac{\log^2(m)}{\log\log(m)}$ items for the cost of 2 per item. We proceed in k rounds. In each round we insert $\frac{1}{2} \cdot \log(m)$ items without any relabels. After the round we redistribute items inserted in this round as evenly as possible. After k rounds, every two items are at distance at least $\frac{2^k \cdot m}{\log^k(m)}$. So we can repeat the process $\frac{1}{2} \cdot \frac{\log(m)}{\log\log(m)}$ times while keeping the minimum distance at least $\sqrt{m}$. Our algorithm generalizes of this idea.

4.3 Main Result

Theorem 4.3.1. *Let $m > 2^{16}$ and k be integers such that $k \leq 1/2\sqrt{\log m / \log\log m}$. Assume $n \leq \log^{k/3}(m)$. Then $\chi_{A^k}(n, m) \leq (2k-1)n$, i.e., there is an algorithm A^k that inserts n items into an array of size m with amortized cost of $2k-1$ per item.*

For n large enough, $m \geq n^{\log n}$ and $k = \frac{3\log n}{\log\log m}$, the assumptions of Theorem 4.3.1 are satisfied, so we get an algorithm which inserts n items into an array of size m with complexity $O(\frac{\log n}{\log\log m})$. Notice that for $m \geq n^{\log n}$, this is asymptotically the same as $O(\frac{\log n}{\log\log m - \log\log n})$. Theorem 4.3.1 is proved in Section 4.5.

4.4 Definitions

To simplify the description we assume (without loss of generality) that cells 1 and m are initially loaded with items $\min_U$ and $\max_U$ which are, respectively, lower and upper bounds on all inserted items. In other words for each t we assume $\mathbf{f}_t(\min_U) = 1$ and $\mathbf{f}_t(\max_U) = m$. Notice that this is different to other chapters where we assume $\mathbf{f}_t(\min_U) = 0$ and $\mathbf{f}_t(\max_U) = m + 1$.

At any step the array has certain occupied cells. A segment of cells whose leftmost and rightmost cells are occupied and all others are unoccupied is called an *open segment*; the items in the leftmost and rightmost cells of the open segment S are denoted $y^L(S)$ and $y^R(S)$ (we include the occupied end cells in the open segment for convenience in some calculations). The initial open segment has size m. The segment is said to be *usable* if $|S| \geq 3$ (which means there is at least one unoccupied cell). For any new item y not stored in the array there is a unique open segment S such that $y^L(S) < y < y^R(S)$; we say that S is *compatible with* y. If item y is assigned to an unoccupied cell in S then the open segment S is split into two open segments which overlap at the cell containing y; the sum of the sizes of these two segments is $|S| + 1$. A *middle cell* of S is a cell such that the two segments obtained from S each have size at least $|S|/2$. It's easy to check that every usable segment has a middle cell. More generally, it can be checked that given $q - 1$ items to be placed in an open interval S that has at least $q - 1$ unoccupied spaces we can place them evenly so that each of the q open segments produced has size at least $|S|/q$. (The worst case is $|S| = aq + 1$ for some integer a, and in this case each of the q resulting subsegments has length $a + 1 \geq |S|/q$.)

4.5 Algorithm Description and Analysis

For each $k \geq 1$ we define an algorithm A^k. It will be obvious from the definitions that the cost per inserted item is at most $2k - 1$. The main technical question will be how many items A^k can handle. Let us define $n_k(m)$ to be the maximum number of items that A^k can handle in an interval of size at least m (the argument m need not to be an integer) at cost $2k - 1$. Our goal is to show that $n_k(m) \geq \lfloor \log^{k/3}(m) \rfloor$.

First we define algorithm A^1. For each successive item y_t ($t \geq 1$), we identify the open segment S compatible with y_t. If it is usable we store y_t in the middle cell of the segment. If there is not such segment, we stop since we tried to insert more items than A^1 can handle in the array of a given size. The pseudocode of A^1 can be found in Figure 4.1.

Algorithm $A^1(m)$

- $\mathbf{f}_0(\min_U) \longleftarrow 1$, $\mathbf{f}_0(\max_U) \longleftarrow m$
- $t \longleftarrow 1$
- Repeat
  - $\mathbf{f}_t \longleftarrow \mathbf{f}_{t-1}$
  - $S \longleftarrow$ segment compatible with y_t

  {***Check whether*** y_t ***can be stored into the array***}
  - if S is usable
    - $\mathbf{f}_t(y_t) \longleftarrow$ middle cell of segment S
  - else
    - goto END
  - $t \longleftarrow t + 1$
- label END
- Output: $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{t-1}$

Figure 4.1: Pseudocode of Algorithm A^1

We analyze the cost of algorithm A^1. We never move any inserted item, so the cost per inserted item is 1 which equals $2k - 1$ (since $k = 1$). Next we want to lower bound the number of items that can be inserted. The size of the initial open segment is m, so after $t-1$ items are inserted every open segment has size at least $m/2^{t-1}$. Recall we can handle arbitrary y_t if this is at least 3. Thus we can insert t items provided that $t - 1 \leq \log(m/3) + 1$. So $n_1(m) \geq \log(m/3) + 1$, which is at least $\log^{1/3}(m)$ for m large enough.

For $k \geq 2$ we define A^k, which makes use of A^{k-1}. We initially insert $\log^{(k-1)/3}(m)$ items using algorithm A^{k-1}, which by induction can be done at amortized cost of $2k-3$ moves per item. Then we rearrange all of the items so that they are spaced as evenly as possible along the array which increases the amortized cost per item to $2k - 2$.

Next the algorithm works in *rounds*. Let old^R denote the set of items inserted prior to round R. Let s^k_{R-1} denote the minimal size of the open segments defined by the allocation of old^R at the beginning of the round R of algorithm $\mathcal{A}^k$.

Each round consists of two phases. During the first phase we refer to the open segments defined by the allocation of inserted items at the beginning of the

phase as *working segments*. We will run A^{k-1} independently on each working segment. When an item is presented we assign it to the working segment it is compatible with, and insert it into the working segment using A^{k-1}. We insert at most $\log^{(k-1)/3}(s^k_{R-1})$ items during the whole round R. Since each working segment has length at least s^k_{R-1}, we are guaranteed that each of the independent copies of A^{k-1} successfully inserts all of their assigned items at amortized cost of $2k-3$.

After the first phase of each round, there may exist non-inserted items such that their compatible segments are not usable. Thus we need to rearrange inserted items to ensure usable segment for each each non-inserted item. Therefore during the second phase we proceed as follows. We say that inserted item y is *old* if $y \in \text{old}^R$ and we say that it is *new* otherwise. Let *excess* of the segment S be the number of old items in it minus the number of new items in it (with respect to a current allocation). First we define $\mathcal{S}$ to be a set of disjoint segments such that each segment in $\mathcal{S}$ has excess exactly 1, each new item is in exactly one segment of $\mathcal{S}$ and the smallest and greatest items of each segment are old. Thus each segment of $\mathcal{S}$ is a concatenation of some working segments. Such set always exists as the whole array has the positive excess. If there are more such sets we choose an arbitrary one. Then we rearrange all segments in $\mathcal{S}$ so that the labels of the smallest and greatest items in each segment are preserved and the remaining items in the segment (both old and new) are redistributed as uniformly as possible.

Since the number of rearranged old items in the segments of $\mathcal{S}$ is at most the number of new items minus one (recall that the smallest and the greatest items of each segment of $\mathcal{S}$ are not rearranged and the excess of each segment is 1), it gives an additional amortized cost of at least 2 per new item. Thus the total amortized cost of A^k is at most $2k-1$ per item.

Therefore to prove Theorem 4.3.1 it only remains to show that $n_k(m) \geq \log^{k/3}(m)$. To do this, we first lower bound the minimal size of the working segment at the start of each round R, s^k_{R-1}. We prove that s^k_{R-1} for $R \geq 1$ is at least $2^{-R+1}m/(q+1)$, where $q = \log^{(k-1)/3}(m)$. For $R = 1$ this is obvious from the definition of A^k as prior to first round we insert $n_{k-1}(m)$ items.

To prove it for $R > 1$ assuming that $s_{R-2} \geq 2^{-R}m/(q+1)$, we focus on the second phase of each round. Let S be an arbitrary segment of $\mathcal{S}$ and let n^{old}_S be the number of old items in it. Obviously its length is at least $(n^{old}_S - 1) \cdot s^k_{R-2}$ as the number of internal old items was $n^{old}_S - 2$. But since the number of new items in the segment is exactly $n^{old}_S - 1$ we can infer that $s_{R-1} \geq \frac{(n^{old}_S-1)\cdot s^k_{R-2}}{2(n^{old}_S-1)}$ which immediately implies that $s^k_{R-1} \geq 2^{-R+1}m/(q+1)$.

Now we can lower bound the number $n_k(m)$ for $k > 1$. (Recall $n_1(m) > \log^{1/3}(m)$.) For $k > 1$ let us assume that $n_{k-1}(n) \geq \log^{(k-1)/3}(m)$. Let r denote the number of rounds. We have $s^k_0 = \frac{m}{q+1} \geq \frac{m}{2\log^{(k-1)/3}(m)}$. Then the number of

items inserted during all rounds is at least

$$\begin{aligned}
\sum_{i=1}^{r} n_i(s_0^k) &\geq \sum_{i=1}^{r} \log^{(k-1)/3}(s_{i-1}^k) \\
&\geq \sum_{i=1}^{r} \log^{(k-1)/3}\left(\frac{s_0^k}{2^{i-1}}\right) \\
&= \sum_{i=1}^{r} \left(\log\left(\frac{m}{2^i \log^{(k-1)/3}(m)}\right)\right)^{(k-1)/3} \\
&= \sum_{i=1}^{r} \left(\log\left(\frac{m}{\log^{(k-1)/3}(m)}\right) - i\right)^{(k-1)/3} \\
&\geq r\left(\log\left(\frac{m}{\log^{(k-1)/3}(m)}\right) - r\right)^{(k-1)/3}.
\end{aligned}$$

Let us chose $r = \sqrt{\log m}$ and we obtain

$$\begin{aligned}
&\sqrt{\log m}\left(\log\left(\frac{m}{\log^{(k-1)/3}(m)}\right) - \sqrt{\log m}\right)^{(k-1)/3} \\
&= \left(\log^{(2k+1)/6}(m)\right)\left(1 - \frac{\sqrt{\log m}}{\log m} - \frac{k-1}{3}\frac{\log\log m}{\log m}\right)^{(k-1)/3}
\end{aligned}$$

Recall that $k < 1/2\sqrt{\log m/\log\log m}$ (statement of Theorem 4.3.1). Thus we obtain the lower bound for this expression

$$\left(\log(m)^{(2k+1)/6}\right)\left(1 - \sqrt{\frac{\log\log m}{\log m}}\right)^{\frac{1}{6}\cdot\sqrt{\frac{\log m}{\log\log m}}} \geq \left(\log(m)^{(2k+1)/6}\right)\left(\frac{1}{2e}\right)^{\frac{1}{6}}$$

which is for $m > 2^{16}$ greater than $\log(m)^{2k/6}$. Therefore during all rounds of A^k, $\log^{k/3}(m)$ items are inserted with an amortized cost $2k-1$ per insertion. This implies Theorem 4.3.1.

Part II

Online Labeling Problem Lower Bounds

5. Introduction

In this part, we present all lower bounds for the online labeling problem known to date. These results prove tight lower bounds for deterministic online labeling algorithms for all array sizes. In addition we provide the first (tight) lower bound for linear size arrays for the case when the universe U is small (recall that for the case $|U| < m$ the online labeling problem is trivial). This result implies that even for small universe size algorithms using arrays of linear size cannot perform asymptotically better than in the case of exponential size universe. Finally we present the first nontrivial lower bound for randomized algorithms, which is tight for arrays of polynomial size. Refer to Table 5.1 for the overview of bounds.

Prior to our results, only little was known about lower bounds for the online labeling problem. There were only two results: the tight lower bound for polynomial size arrays (which of course applies also for smaller arrays, but provides non-tight bound) by Dietz et al. [13] and another result by Dietz et al. [12], that proves a tight lower bound for a *restricted* class of online labeling algorithms (so called *smooth* algorithms) in case of linear size arrays. Both of these results appear also in Ph.D. thesis by Zhang [21], which prior to our work was the most comprehensive source for the problem.

5.1 Overview of Results

Here we briefly describe out results that we present in this part.

In Chapter 6 we present the tight lower bound for deterministic algorithms using arrays of size from n^c ($c > 1$) to 2^n. Our proof extends and simplifies the result of Dietz at al. [13] (also in [21]) which is valid only for arrays of polynomial size.

In Chapter 7 we describe the first lower bound for randomized algorithms. In particular, this result is tight for polynomial size arrays. Since this bound is the same (up to a constant factor) as the one presented in Chapter 6 we can infer that at least for polynomial size arrays randomized algorithms are not asymptotically better in expectation than the deterministic ones. As this section uses basic ideas similar to Chapter 6 we advice you to read first Chapter 6 which uses simpler argument and introduces many of the necessary concepts.

The remaining chapters focus on arrays of almost linear size.

In Chapter 8 we present the first tight lower bound for the superlinear (but "subpolynomial") arrays. This result also proves the tight lower bounds for linear arrays which makes it seemingly superior to the result in the later Chapter 9. However, in this result we do not prove the tight bounds for array of size close to

Array size (m)	Asymptotic bound	Note	Reference
$n < m \leq 2n$	$\Theta\big(n \log^2(n) \log(\frac{n}{m-n})\big)$		Chapter 9
$m = cn$, constant $c > 1$	$\Theta\big(n \log^2(n)\big)$	1)	Chapters 8, 9
$m = n^{1+o(1)}$	$\Theta\Big(\frac{n \log^2(n)}{1+\log m - \log n}\Big)$		Chapter 8
$m = n^C$, constant $C > 1$	$\Theta(n \log n)$	2)	Chapters 6,7
$m \in [n^{1+\omega(1)}, n^{\log(n)})$	$\Omega\Big(\frac{n \log n}{1+\log\log m - \log\log n}\Big)$	3)	Chapter 6
$m = n^{\Omega(\log(n))}$	$\Theta\Big(\frac{n \log n}{1+\log\log m - \log\log n}\Big)$		Chapter 6

1) Valid even for the case when the universe U is small.
2) Valid even for randomized algorithms.
3) We do not know a matching upper bound.

Table 5.1: Overview of results of Part II

n and we do not consider the case of small item universe.

Finally in the last Chapter 9, we prove the tight lower bound for linear arrays even in the case of arrays of size of n (recall that for such arrays we have an upper bound $O(n \log^3(n))$) for arbitrary deterministic algorithms. The previous results by Dietz et al. [12, 21] consider only the restricted class of *smooth* algorithms. This restriction makes the construction of an adversary against the algorithms easy (as opposed to the case of arbitrary algorithms), still the rest of their analysis is non-trivial and contains useful ideas which we build on.

In addition this is the first result which considers the case of limited universe of items we insert. This is important question as we already mention that for the universe U of size $r < m$ the online labeling problem is trivial and it is not clear whether a small universe of size comparable to m cannot help one to develop better algorithms. However, we show that for linear size arrays the lower bound is asymptotically the same as for non-limited universe.

Chapter 9 uses similar ideas as Chapter 8, but Chapter 8 is significantly easier to follow, as it proves the lower bound only for the online labeling problem without further limitation. Thus we recommend you to read Chapter 8 prior to Chapter 9.

5.2 Definitions

In the lower bound chapters it will be convenient to introduce the following assumptions and definitions. First recall that by log we denote the binary logarithm unless otherwise specified and by ln we denote a natural logarithm.

Let $\mathcal{A}$ be an arbitrary online labeling algorithm with parameters (n, m, r), where $n \leq m < r$. By $U = \{1, \ldots, r\}$ we denote the universe from which the items to be inserted are chosen. Recall the definitions from Section 1.2. By Y_t we denote the set of items inserted in first t steps. $\mathbf{f}_{\mathcal{A},t}$ is a labeling of items from Y_t at step t produced by algorithm $\mathcal{A}$. The set of items relabeled at step t is denoted $\mathbf{Rel}_t$. For the sake of analysis we define $\min_U = 0$ and $\max_U = 2^{r+1}$, i.e., items which are smaller and larger than any item in the universe. Without loss of generality we assume that any online labeling algorithm $\mathcal{A}$ assigns $\mathbf{f}_{\mathcal{A},t}(\min_U) = 0$ and $\mathbf{f}_{\mathcal{A},t}(\max_U) = m + 1$ at each step t. Clearly this assumption does not affect the cost of the algorithm.

Let Y be a subset of items of the universe U. We say that I is an *interval* of Y if

$$I = Y \cap \{\min(I), \ldots, \max(I)\}.$$

We may omit Y if it is clear from the context.

Consider an arbitrary interval I of $Y_t \cup \{\min_U, \max_U\}$. We define the span of an interval I as follows

$$\mathbf{span}_t(I) = \mathbf{f}_{\mathcal{A},t}(\max(I)) - \mathbf{f}_{\mathcal{A},t}(\min(I)).$$

5.3 Proof Techniques

The online labeling problem can be seen as a game of two players - maintenance algorithm $\mathcal{A}$ and the Adversary. The goal in constructing the adversary is to force the online algorithm $\mathcal{A}$ to perform many relabellings during insertion of n keys. The game is played in rounds. In each round t, Adversary may look at the labeling $\mathbf{f}_{\mathcal{A},t-1}$ and then it determines next item y_t to be inserted. In the case of randomized algorithms, the adversary doesn't get to see a particular labeling but only the expected state of $\mathbf{f}_{\mathcal{A},t-1}$.

The main idea of all adversaries is to insert to the segments of array which already contains many items. Repeated insertions eventually force the algorithm to move the existing items. Deriving a lower bound based on this idea has various complications. The natural notion of crowding of a segment is the ratio of stored items to the size of the segment. Whether a particular portion of the array is considered to be crowded may depend on the scale of segments being considered; there may be a relatively small segment that is very crowded, but larger segments

containing it are uncrowded. To force the algorithm to work hard, we want to identify a segment that is crowded at many different scales. This suggests identifying a long nested sequence of segments covering a wide range of scales, such that each is crowded. The hope is that loading many items having value in the middle of the range of items stored in the smallest nested segment will eventually force the algorithm to do costly rearrangements at many different scales.

A straightforward way to accomplish this is to start with the entire array, and successively select a nested subsegment. The way we select such subsegment differs among the adversaries, however the basic idea is always that same: we try to track the density in the array.

A dual view is to recursively pick subintervals of inserted items that span the shortest possible segment(i.e., the highest density segment). Except in Chapter 9 we use this dual view. The advantage of the dual view will be apparent in the case of randomized labeling where we know what are the inserted items but we do not know their particular location in the array.

5.4 Lazy Algorithms

When proving our lower bounds we will restrict our focus to a subclass of online labeling algorithms that are *lazy*. Next we provide definition of this property and show that any labeling algorithm can be made lazy without affecting its cost. This property was considered in [21] under the name *normalized*

Definition 5.4.1. *Let $\mathcal{A}$ be an arbitrary algorithm with parameters (n, m, r). We say that the algorithm $\mathcal{A}$ is* lazy *if for each $t \in \{1, \ldots, n\}$ it holds that $\mathbf{Rel}_{\mathcal{A},t}$ is an interval of Y_t.*

In other words the algorithm is lazy if in each step all rearranged items form a subinterval of already inserted items.

Now we can show the main property of the lazy algorithms.

Lemma 5.4.2. *Let $\mathcal{A}$ be an arbitrary algorithm with parameters (n, m, r), where $n \leq m < r$. Then there exists a lazy algorithm $\mathcal{A}'$ such that for each n,*

$$\chi_{\mathcal{A}}(n, m, r) \geq \chi_{\mathcal{A}'}(n, m, r).$$

To prove this lemma we construct algorithm $\mathcal{A}'$ inductively on t based on the original algorithm $\mathcal{A}$. The basic idea is that we split $\mathbf{Rel}_t$ into minimum possible number of intervals of Y_t and we relabel only the items in the interval of Y_t which contains y_t while we postpone relabeling of all the other intervals.

We proceed with the pseudocode of $\mathcal{A}'$.

Algorithm$(\mathcal{A}', n, m)$

- $\mathbf{Postponed}'_0 \longleftarrow \emptyset$.
- for $t \in \{1, \ldots, n\}$:
  - $\mathbf{Rel}'_t \longleftarrow$ maximal subset of $\mathbf{Postponed}'_{t-1} \cup \mathbf{Rel}_t$ which contains y_t and is an interval of Y_t
  - for each $y \in Y_t \setminus \mathbf{Rel}'_t$:
    - $\mathbf{f}_{\mathcal{A}',t}(y) \longleftarrow \mathbf{f}_{\mathcal{A}',t-1}(y)$
  - for each $y \in \mathbf{Rel}'_t$:
    - $\mathbf{f}_{\mathcal{A}',t}(y) \longleftarrow \mathbf{f}_{\mathcal{A},t}(y)$
  - $\mathbf{Postponed}'_t \longleftarrow (\mathbf{Postponed}'_{t-1} \cup \mathbf{Rel}_t) \setminus \mathbf{Rel}'_t$

Now we prove a few claims about the properties of $\mathcal{A}'$. First we state a claim which is an immediate consequence of the construction of $\mathcal{A}'$.

Claim 5.4.3. *For each $t \in \{0, \ldots, n\}$ and for each $y \in Y_t$ it holds that $y \in \mathbf{Postponed}'_t$ or $\mathbf{f}_{\mathcal{A}',t}(y) = \mathbf{f}_{\mathcal{A},t}(y)$.*

Now we can prove that $\mathbf{f}_{\mathcal{A}',t}$ preserves the ordering of items in Y_t.

Claim 5.4.4. *For each $t \in \{0, \ldots, n\}$ and for each $y, y' \in Y_t$ such that $y < y'$ it holds $\mathbf{f}_{\mathcal{A}',t}(y) < \mathbf{f}_{\mathcal{A}',t}(y')$.*

Proof. From the construction of $\mathcal{A}'$ and the previous claim we can immediately infer for each t that for maximal $y \in Y_t \cup \{\min_U, \max_U\}$ such that $y < \min(\mathbf{Rel}'_t)$ it holds that $\mathbf{f}_{\mathcal{A}',t}(y) = \mathbf{f}_{\mathcal{A},t}(y)$ and similarly for minimal $y \in Y_t$ such that $y > \max(\mathbf{Rel}'_t)$ it holds that $\mathbf{f}_{\mathcal{A}',t}(y) = \mathbf{f}_{\mathcal{A},t}(y)$. Thus by relabeling items in $\mathbf{Rel}'_t$ we cannot break the ordering. Now it is easy to proceed by induction on t to finish the proof. □

Now we proceed with the proof of Lemma 5.4.2.

Proof of Lemma 5.4.2. Since we already know that $\mathcal{A}'$ produces correct labeling (Claim 5.4.4) it only remains to show that its cost is small enough. We actually prove that for each $t \in \{0, \ldots, n\}$ it holds that $\chi_{\mathcal{A}}(t) \geq \chi_{\mathcal{A}'}(t) + |\mathbf{Postponed}'_t|$. This is obvious for $t = 0$ so let us assume that claim is true for $t - 1$. Recall that

$$\mathbf{Postponed}'_t = (\mathbf{Postponed}'_{t-1} \cup \mathbf{Rel}_t) \setminus \mathbf{Rel}'_t$$

$$\mathbf{Rel}'_t \subseteq \mathbf{Postponed}'_{t-1} \cup \mathbf{Rel}_t.$$

Thus

$$|\mathbf{Postponed}'_t| \leq |\mathbf{Postponed}'_{t-1}| + |\mathbf{Rel}_t| - |\mathbf{Rel}'_t|.$$

This implies the following chain of inequalities

$$\begin{aligned}\chi_{\mathcal{A}}(t) - \chi_{\mathcal{A}'}(t) &= \chi_{\mathcal{A}}(t-1) - \chi_{\mathcal{A}'}(t-1) + |\mathbf{Rel}_t| - |\mathbf{Rel}'_t| \\ &\geq |\mathbf{Postponed}'_{t-1}| + |\mathbf{Rel}_t| - |\mathbf{Rel}'_t| \\ &\geq |\mathbf{Postponed}'_t|.\end{aligned}$$

where the first inequality follows from the induction hypothesis. The proof follows. □

The consequence of Lemma 5.4.2 is immediate. When proving lower bounds for the online labeling problem, we can focus only on the lazy algorithms. We use this lemma in all lower bound proofs.

6. Online Labeling with Large Label Space

6.1 Introduction

In this chapter we prove an $\Omega\left(n \cdot \frac{\log(n)}{\log\log(m)-\log\log(n)}\right)$ lower bound on the number of moves for inserting n items. This lower bound holds for m between n and 2^n. Note that for polynomially many labels this bound simplifies to $\Omega(n \log n)$. This is tight except for $m \in [n^{\omega(1)}, n^{\log n})$, where we do not know a matching upper bound.

For the case of polynomially many labels, Dietz at al. [13] (also in [21]) proved a matching lower bound for the $O(n \log n)$ upper bound. Their result consists of two parts; a lower bound for a problem they call *prefix bucketing* and a reduction from prefix bucketing to online labeling. We provide a simpler, tighter and more general lower bound for prefix bucketing, which allows us to extend the lower bounds for online labeling to the case when the label space size is as large as 2^n.

Our initial study of the proof in [13] of the reduction from prefix bucketing to online labeling led us to think that there is a significant gap in their proof. We modified their proof and obtained a correct version of the reduction. In [2] (on which this chapter is based) we claimed that we were correcting an apparently significant gap in the previous proof; we made this claim after checking with one of the authors who agreed with it. Having done a more careful comparison of our final proof to theirs, we see that while the proof in [13] has some misleading statements and missing details, it is essentially correct. We present the details of our modification in order to clarify the ambiguities that were present in [13].

Recall that we use definitions from Sections 1.2.

6.2 The Main Theorem

In this section, we state the main result of this chapter.

Theorem 6.2.1. *There are positive constants C_0, and C_1 so that the following holds. Let $\mathcal{A}$ be a deterministic algorithm with parameters (n, m, r), such that $C_0 \leq n \leq m \leq 2^n$ and $r \geq 2^n - 1$. Then $\chi_{\mathcal{A}}(n, m, r) \geq C_1 \cdot \frac{n \log n}{3+\log\log m-\log\log n}$.*

Notice that this theorem implies a nontrivial lower bound even for the linearly sized array ($\Omega(n \log n)$), however, it is not tight.

To prove the theorem we fix a labeling algorithm $\mathcal{A}$ and describe an adversary who builds a sequence $y_1, y_2, \ldots, y_t$ of items based on the behavior of $\mathcal{A}$ that will cause the algorithm to incur the desired cost. In Section 6.5 we will describe the

adversary, and state Lemma 6.5.1, which proves a lower bound on the cost incurred by algorithm $\mathcal{A}$ on the sequence produced by the adversary. Theorem 6.2.1 follows immediately from Lemma 6.5.1.

6.3 Reducing Prefix Bucketing to Online Labeling

Dietz et al. [13] sketched a reduction from prefix bucketing to online labeling. In their reduction they describe an adversary for the labeling problem. They show that given any algorithm for online labeling, the behavior of the algorithm against the adversary can be used to construct a strategy for prefix bucketing. As mentioned above, while the idea of their adversary and the reduction from prefix bucketing to online labeling are essentially correct, the description is incomplete and ambiguous. In Section 6.5 we will present a modification of their adversary, and in Section 6.7 we give a full proof of the connection to prefix bucketing. We now sketch the adversary construction and the reduction.

The goal in constructing an adversary is to force any online algorithm to perform many relabelings during insertion of n items. The adversary starts by inserting the artificial items $\min_U$ and $\max_U$ to positions 0 and $m+1$. Then in each further step it determines one of the inserted items y and chooses next item so that it will be adjacent (i.e. there is no inserted item between them) to y. The central issue in defining the adversary is to determine at each step which inserted item to choose.

As a guide to picking each successive item, the adversary maintains a sequence (*chain*) of nested subintervals of already inserted items. The chain serves a dual purpose: the chain after step t is used by the adversary to select the item inserted at step $t+1$, and the sequence of chains over time provides a way to lower bound the total cost incurred by the algorithm. Each successive interval in the chain has span at most half the previous interval, and its density is within a constant factor of the density of the previous interval. The chain ends with an interval containing between 2 and 7 items. The next item to be inserted is then chosen so that it is adjacent to the smallest item in the lowest interval of the chain.

Initially, and during the first 7 inserts, the chain consists of just the single interval containing at most 7 items. After each subsequent insert, the algorithm $\mathcal{A}$ specifies the label of the next item and (possibly) relabels some items. The adversary then updates its chain. For the chain immediately prior to the insert, the adversary specifies the *critical interval* to be the smallest interval of the chain such that its smallest and greatest items were not relabeled. Its index in the chain is then called critical level and is denoted by q_t. The new chain is then obtained as follows. We say that intervals with index at most q_t are *preserved* for step t which means that they are carried over from the previous chain, with the addition of y_t.

Otherwise the intervals are *rebuilt*. Beginning from the critical interval the chain is extended as follows. Having chosen the interval I from the chain, define its *left buffer* to be the smallest $1/8$ items of I, and its *right buffer* to be the greatest $1/8$ items of I. Let I' be the interval obtained from I by deleting the left and right buffers. The successor interval of I in the chain is a subinterval of I' with the minimum span that contains exactly half (rounded down) of the items of I'. The chain ends when we reach an interval with at most seven items.

In [13], the authors show that there is always a *dense point*, which is a point with the property that every subsegment of the array containing it has density at least half the overall density of the label space. They use this as the basis for building the chain. We build a chain with similar properties using a simpler argument.

It remains to prove that the algorithm will make lot of relabels on the sequence of items produced by the adversary. Following Dietz et al. [13], we do this by relating online labeling to the prefix bucketing game. (Our definition of the game differs slightly from that in [13].)

A *prefix bucketing of n items into k buckets* (numbered 1 to k) is a one player game consisting of n steps. At the beginning of the game all the buckets are empty. In each step a new item arrives and the player selects an index $p \in \{1, \ldots, k\}$. The new item as well as all items in buckets $1, \ldots, p-1$ are moved into bucket p at a cost equal to the total number of items in bucket p after the move. The run of the game is therefore completely specified by the sequence $p_1, \ldots, p_n$, where p_t is the bin into which the player placed the new item at step t. The goal is to minimize the total cost of n steps of the game.

The lower bound on online labeling is obtained by the following correspondence. Consider the run of an arbitrary online labeling algorithm $\mathcal{A}$ against the given adversary. At each step t, the adversary determines a particular level p_t of the chain to be the critical level (which is always at most $k = \lceil \log(m+1) \rceil$). Consider the sequence $p_1, \ldots, p_n$ as a sequence of placements defining a prefix bucketing of n items into $k = \lceil \log(m+1) \rceil$ buckets. It turns out that the total cost of the prefix bucketing will be within a constant factor of the total number of relabelings performed by the online labeling algorithm. Hence, a lower bound on the cost of a prefix bucketing of n items into k buckets will imply a lower bound on the cost of the algorithm against our adversary.

The connection between the cost of $\mathcal{A}$ against the adversary, and the cost of the associated prefix bucketing is obtained as follows. Recall that we may assume that the algorithm is *lazy*. The cost of the bucketing merge step p_t at step t is at most the number of items in the critical interval, so to relate this to the cost incurred by the online labeling algorithm, it is enough to argue that at step t a constant fraction of the items in the critical interval were relabeled. This is done

by arguing that for each successor (sub)interval of the critical interval, either all labels in its left buffer or all labels in its right buffer were reassigned, and the total number of such items is a constant fraction of the items in the critical interval.

6.4 An Improved Analysis of Bucketing

It remains to give a lower bound for the cost of prefix bucketing. This was previously given by Dietz et al. [13] for $k \in \Theta(\log n)$. We give a different and simpler proof that gives asymptotically optimal bound for k between $\log n$ and $O(n^\epsilon)$. We define a family of trees called k-admissible trees and show that the cost of bucketing for n and k, is between $dn/2$ and dn where d is the minimum depth of a k-admissible tree on n vertices. We further show that the minimum depth of a k-admissible tree on n vertices is equal $g_k(n)$ which is defined to be the smallest d such that $\binom{k+d-1}{k} \geq n$. This gives a characterization of the optimal cost of prefix bucketing (within a factor of 2). When we apply this characterization we need to use estimates of $g_k(n)$ in terms of more familiar functions (Lemma 6.8.12), and there is some loss in these estimates.

6.5 Adversary Construction

We now specify an adversary **Adversary**$(\mathcal{A}, n, m)$ which given an online labeling algorithm $\mathcal{A}$, an integer n, and label space size m, constructs an item sequence $y_1, y_2, \ldots, y_n$ from the universe $U = \{1, \ldots, 2^n - 1\}$.

To pick the sequence of items $y_1, \ldots, y_n$, the adversary will maintain a sequence of nested intervals of inserted items $Y_t = \{y_1, y_2, \ldots, y_t\}$, [1]

$$I_t(\mathbf{depth}_t) \subseteq \cdots \subseteq I_t(2) \subseteq I_t(1) = Y_t \cup \{\min_U, \max_U\},$$

updating them after each time step t. In what follows, $\mathbf{f}_t$ is the allocation of Y_t by the algorithm $\mathcal{A}$. Consider an arbitrary subinterval I of $I_t(1)$. For a positive integer b, $b \leq |I|/2$, let $\mathbf{densify}_t(I, b)$ be the subinterval T of I such that

- T does not contain any of the b largest or smallest elements of I.
- $|T| = \left\lfloor \frac{|I|-2b}{2} \right\rfloor$.
- $\mathbf{span}_t(T)$ is minimal among all possible T's.

[1] Recall, we use subscript to denote step and we use $(\cdot)$ notation to denote a particular coordinate of such a vector or sequence at that step.

If there are more such subintervals T we choose the one with the minimal smallest item. Hence, $\mathbf{densify}_t(I, b)$ is the subinterval T of I with the minimum span that contains the appropriate number of items but which excludes at least b smallest and largest items from I. These b items on either side of T form the left and right buffers described in Section 6.3.

We are now ready to present our adversary. In the adversary, $b_t(i)$ denotes the number of items in the left and right buffers of $I_t(i)$. This number changes only when $I_t(i)$ is rebuild. Also, q_t denotes the critical level of the chain after inserting item y_t, i.e., the level from which the chain will be rebuilt.

The adversary pseudocode is given in Figure 6.1.

A crucial assumption for this adversary is that $y_1, \ldots, y_t$ are distinct. It is easy to see by induction on t that the items belonging to $Y_t \cup \{\min_U, \max_U\}$ are multiples of 2^{n-t}, and it follows that y_t is strictly between the smallest and second smallest elements of $I_{t-1}(\mathbf{depth}_{t-1})$.

We make the following claim about the adversary which together with Lemma 5.4.2 implies Theorem 6.2.1.

Lemma 6.5.1. *Let m, n be positive integers such that $n \leq m$. Let $\mathcal{A}$ be a lazy online labeling algorithm with the range m. Let $y_1, y_2, \ldots, y_n$ be the output of* $\mathbf{Adversary}(\mathcal{A}, n, m)$ *(Figure 6.1). Then the cost*

$$\chi_{\mathcal{A}}(y_1, y_2, \ldots, y_n) \geq \frac{1}{512} \cdot \frac{n \log n}{3 + \log\lceil \log(m+1) \rceil - \log\log n} - \frac{n}{6}.$$

The constants are chosen for ease of exposition and can certainly be improved.

To prove the lemma we will design a so called *prefix bucketing game* from the interaction between the adversary and the algorithm, we will relate the cost of the prefix bucketing to the cost $\chi_{\mathcal{A}}((y_1, y_2, \ldots, y_n), m)$ (Lemma 6.7.2) and we will lower bound the cost of the prefix bucketing (Lemma 6.8.13). The proof of Lemma 6.5.1 is on page 50.

In preparation for this, we prove several useful properties of the adversary and introduce some additional definitions.

By $\mathbf{birth}_t(i)$ we denote the greatest t' such $t' \leq t$ and $q_{t'} < i$, i.e., when $I_i(t)$ was rebuilt last time.

We start by setting limits on the size of the smallest interval in the chain.

Claim 6.5.2. *For each $t \in \{0, \ldots, n\}$ it holds that $|I_t(\mathbf{depth}_t)| \geq 2$.*

Proof. This is obviously true for $t = 0$. Now assume that the claim is true for $t - 1$. From $I_{t-1}(\mathbf{depth}_{t-1}) \subseteq I_{t-1}(q_t)$ it follows that $|I_{t-1}(q_t)| \geq 2$. Using this, the termination condition of Rebuilding Rule implies the claim immediately. □

Adversary$(\mathcal{A}, n, m)$

{***Initialization***}

- $y_1 \longleftarrow 0$
- $I_0(1) \longleftarrow \{\min_U, \max_U\}$, $\mathbf{depth}_0 \longleftarrow 1$, $b_0(1) \longleftarrow 1$
- For $t = 1, \ldots, n$ do
  - $y_t \longleftarrow \min(I_{t-1}(\mathbf{depth}_{t-1})) + 2^{n-t}$
  - Run $\mathcal{A}$ on $(y_1, y_2, \ldots, y_t)$ to set $\mathbf{f}_t$ and $\mathbf{Rel}_t$.
  - {***Choose Critical Level***}
  - $q_t \longleftarrow i$ maximal such that $\min(I_{t-1}(i)), \max(I_{t-1}(i)) \notin \mathbf{Rel}_t$
  - $i \longleftarrow 1$
  - {***Preservation Rule***}
  - While $(i \leq q_t)$
    - $I_t(i) \longleftarrow I_{t-1}(i) \cup \{y_t\}$
    - $b_t(i) \longleftarrow b_{t-1}(i)$
    - $i \longleftarrow i + 1$
  - {***Rebuilding Rule***}
  - While $|I_t(i-1)| \geq 8$
    - $I_t(i) \longleftarrow \mathbf{densify}_t(I_t(i-1), b_t(i-1))$
    - $b_t(i) \longleftarrow \lceil |I_t(i)|/8 \rceil$
    - $i \longleftarrow i + 1$
  - $\mathbf{depth}_t \longleftarrow i - 1$
  - $b_t(\mathbf{depth}_t) \longleftarrow 1$

Output: $y_1, y_2, \ldots, y_n$

Figure 6.1: Pseudocode for the adversary

Next claim gives us a better understanding of how the intervals are changed during each step. We show that unless Rebuilding Rule is applied to an interval, the interval changes only by adding y_t at step t.

Claim 6.5.3. *For any $t \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, q_t\}$ it holds that*

$$\mathbf{Rel}_t \subseteq I_t(i) \setminus \{\min(I_t(i)), \max(I_t(i))\}.$$

Proof. If we prove that for $i = q_t$ we are done since for $i < q_t$ $I_t(q_t) \subseteq I_t(i)$. From the construction we know immediately that $y_t > \min(I_{t-1}(\mathbf{depth}_{t-1}))$. To prove $y_t < \max(I_{t-1}(\mathbf{depth}_{t-1}))$ it is enough to use Claim 6.5.2. Since $I_{t-1}(\mathbf{depth}_{t-1}) \subseteq I_{t-1}(q_t)$ we also obtain $y_t > \min(I_{t-1}(q_t))$ and $y_t < \max(I_{t-1}(q_t))$.

To finish the proof we only have to combine the fact from the previous paragraph, the fact that $\mathcal{A}$ is lazy and finally the fact that $\min(I_{t-1}(q_t)) \notin \mathbf{Rel}_t$ and $\max(I_{t-1}(q_t)) \notin \mathbf{Rel}_t$ and $I_t(q_t) = I_{t-1}(q_t) \cup y_t$. □

Now we show that unless the Rebuilding Rule is applied to some interval of the chain its span is preserved.

Claim 6.5.4. *For any $t \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, \mathbf{depth}_t\}$, let $t_b = \mathbf{birth}_t(i)$. Then $\mathbf{span}_{t_b}(I_{t_b}(i)) = \mathbf{span}_t(I_t(i))$.*

Proof. We proceed by induction on t. For $t = t_b$ the equality is trivial. Now assume that equality is true for $t - 1 \geq t_b$. From Claim 6.5.3 we obtain immediately that $\mathbf{span}_{t-1}(I_{t-1}(i)) = \mathbf{span}_t(I_t(i))$. Now we can use the induction hypothesis and the claim follows. □

By application of previous claim, we can show the relation between the span of two consecutive intervals.

Claim 6.5.5. *For any $t \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, \mathbf{depth}_t - 1\}$, it holds that $\mathbf{span}_t(I_t(i)) \geq 2 \cdot \mathbf{span}_t(I_t(i+1))$.*

Proof. We prove this again by induction on t. For $t < 8$ there is nothing to prove as the $\mathbf{depth}_t = 1$. So let us assume the claim is true for $t - 1$. We distinguish two cases.

For $i < q_t$ we only have to combine induction hypothesis and Claim 6.5.4.

For $i \geq q_t$ the situation is more complicated. Consider intervals $I_t(i)$ and $I_t(i+1)$. Recall that $I_t(i+1) = \mathbf{densify}_t(I_t(i), b_t(i))$. Now consider interval L which consists of $\left\lfloor \frac{|I_t(i)|}{2} \right\rfloor$ smallest items of $I_t(i)$ and R which consists $\left\lfloor \frac{|I_t(i)|}{2} \right\rfloor$ greatest items of $I_t(i)$. It holds that $\mathbf{span}_t(L) \leq \frac{1}{2} \cdot \mathbf{span}_t(I_t(i))$ or $\mathbf{span}_t(R) \leq \frac{1}{2} \cdot \mathbf{span}_t(I_t(i))$ as $L \cap R = \emptyset$. However it is obvious that $\mathbf{span}_t(I_t(i+1))$ is smaller than both of L and R. This finishes the proof. □

Next lemma is an immediate corollary of the previous claim and the termination condition of Rebuilding Rule.

Lemma 6.5.6. *For each $t \in \{1, \ldots, n\}$ it holds that* $\mathbf{depth}_t \leq \log(m+1)$.

We proceed with the claim which connects the size of the difference of two consecutive intervals of the chain and $b_t(i)$. For any $t \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, \mathbf{depth}_t - 1\}$, the difference $I_t(i) \setminus I_t(i+1)$ is a pair of intervals denoted $L_t(i)$ (to the *left* of $I_t(i+1)$) and $R_t(i)$.

Claim 6.5.7. *For any $t \in \{1, \ldots, n\}$ and any $i \in \{1, \ldots, \mathbf{depth}_t - 1\}$,*

$$|L_t(i)| \geq b_t(i) \text{ and } |R_t(i)| \geq b_t(i).$$

Proof. We prove this by induction on t. Let $t_b = \mathbf{birth}_t(i)$. For $t = t_b$ the claim is trivial. So let us assume the claim is true for $t-1$. We distinguish two cases.

If $i < q_t$ then the construction of adversary implies that $b_t(i) = b_t(i-1)$, $I_t(i) = I_{t-1}(i) \cup y_t$ and $I_t(i+1) = I_{t-1}(i+1) \cup y_t$. Thus if the claim was true for $I_{t-1}(i)$ and $I_{t-1}(i+1)$ it has to be true also for $I_t(i)$ and $I_t(i+1)$

If $i \geq q_t$ the adversary construction implies that $I_t(i+1) = \mathbf{densify}_t(I_t(i), b_t(i))$. Thus there are at least $b_t(i)$ items smaller than $\min(I_t(i+1))$ in $I_t(i)$ and also at least $b_t(i)$ items greater than $\max(I_t(i+1))$ in $I_t(i)$. This finishes the proof. □

This lemma reflects a subtle point in the adversary. In the definition of $b_t(i)$ we set it to $\lceil |I_t(i)|/8 \rceil$ only in the case that $i > q_{t-1}$, while it might seem more natural to always define $b_t(i)$ in this way. However, our actual definition which sets $b_t(i) = b_{t-1}(i)$ for $i \leq q_{t-1}$ is needed for the induction step in the above proof.

We now use this to relate the cost of relabeling at step t to the quantities $b_{t-1}(i)$:

Lemma 6.5.8. *If $\mathcal{A}$ is lazy then for any $t \in \{1, \ldots, n\}$,* $|\mathbf{Rel}_t| \geq \sum_{i=q_t+1}^{\mathbf{depth}_{t-1}} b_{t-1}(i)$.

Proof. If $q_t = \mathbf{depth}_{t-1}$ then the sum on the right hand side of the inequality evaluates to zero while $|\mathbf{Rel}_t| \geq 1$ since $y_t \in \mathbf{Rel}_t$.

Thus we may assume $q_t < \mathbf{depth}_{t-1}$. From the choice of the critical level we know that

$$\min(I_{t-1}(q_t+1)) \in \mathbf{Rel}_t \text{ or } \max(I_{t-1}(q_t+1)) \in \mathbf{Rel}_t$$

(or both). Assume $\min(I_{t-1}(q_t+1)) \in \mathbf{Rel}_t$ and set $y' = \min(I_{t-1}(q_t+1))$. (For $\max(I_{t-1}(q_t+1)) \in \mathbf{Rel}_t$ the proof is symmetric.) Consider interval I such that

$$I = \{y : y \in Y_{t-1} \text{ and } y' < y \text{ and } y < y_t\}.$$

Since $\mathcal{A}$ is lazy we can infer $I \subset \mathbf{Rel}_t$. Additionally, as $y_t > \min(I_{t-1}(\mathbf{depth}_{t-1}))$ it follows that $\bigcup_{i=q_t+1}^{\mathbf{depth}_{t-1}-1} L_{t-1}(i) \subseteq I$. Thus Claim 6.5.7 implies that

$$|\mathbf{Rel}_t| \geq |I| + 1 \geq 1 + \sum_{i=q_t+1}^{\mathbf{depth}_{t-1}-1} b_{t-1}(i).$$

The proof finishes the fact that from construction we know that $b_{t-1}(\mathbf{depth}_{t-1}) = 1$ for each t.

□

The next step is deriving a lower bound on $b_t(i)$.

Lemma 6.5.9. *For any $t \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, \mathbf{depth}_t - 1\}$,*

$$64 \cdot b_t(i+1) \geq |I_t(i)| - |I_t(i+1)|.$$

Proof. For $t < 8$, there is no $i \in \{1, \ldots, \mathbf{depth}_t - 1\}$ so the lemma is true trivially. Thus, consider $t \geq 8$ and assume that claim is true for $t-1$. We distinguish two cases:

If $i < q_t$ then the construction of adversary implies that $b_t(i) = b_{t-1}(i)$, $I_t(i) = I_{t-1}(i) \cup y_t$ and $I_t(i+1) = I_{t-1}(i+1) \cup y_t$. Thus if the claim was true for $I_{t-1}(i)$ and $I_{t-1}(i+1)$ it has to be true also for $I_t(i)$ and $I_t(i+1)$.

If $i \geq q_t$ (and $i \leq \mathbf{depth}_t - 1$) we first notice that $|I_t(i)| \geq 8$ which follows from the construction. Thus

$$b_t(i) = \lceil |I_t(i)|/8 \rceil \leq |I_t(i)|/4,$$

which immediately implies

$$|I_t(i+1)| = \lfloor (|I_t(i)| - 2 \cdot b_t(i))/2 \rfloor \geq \lfloor |I_t(i)|/4 \rfloor \geq |I_t(i)|/8.$$

The adversary definition also implies

$$b_t(i+1) = \lceil |I_t(i+1)|/8 \rceil \geq |I_t(i+1)|/8.$$

Connecting these facts together we obtain

$$b_t(i+1) \geq |I_t(i)|/64 \geq (|I_t(i)| - |I_t(i+1)|)/64.$$

The lemma follows immediately. □

Corollary 6.5.10. *For any $t \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, \mathbf{depth}_t - 1\}$,*

$$\sum_{i=j+1}^{\mathbf{depth}_t} b_t(i) \geq \frac{1}{64} \cdot |I_t(j)| - \frac{1}{8}.$$

Proof. For fixed t, sum the inequality in Lemma 6.5.9 for i from j to $\mathbf{depth}_t - 1$, and note that $I_t(\mathbf{depth}_t) < 8$. Then divide through by 64. □

We now come to the main lower bound of this section.

Corollary 6.5.11. *Let $\mathcal{A}$ be a lazy algorithm with parameters (n, m). Let sequence $(y_1, y_2, \ldots, y_n)$ be obtained by* **Adversary**$(\mathcal{A}, n, m)$*. Then*

$$\chi_{\mathcal{A}}\left((y_1, y_2, \ldots, y_n), m\right) \geq \frac{1}{64} \sum_{t=1}^{n} |I_{t-1}(q_t)| - \frac{n}{8}.$$

Proof. $\chi_{\mathcal{A}}\left((y_1, y_2, \ldots, y_n), m\right)$ is at least $\sum_{t=1}^{n} |\mathbf{Rel}_t|$. Now combine Lemmas 6.5.8 and Corollary 6.5.10. (For t such that $q_t = \mathbf{depth}_t$ we use the fact that $|\mathbf{Rel}_t| > 0 > \frac{1}{64} \cdot |I_{t-1}(q_t)| - \frac{1}{8}$.) □

6.6 Prefix Bucketing

We start by a definition of the prefix bucketing. A *bucket configuration* for k buckets and t items is a sequence $\mathbf{a}(1), \ldots, \mathbf{a}(k)$ of nonnegative integers summing to t. One should think of $\mathbf{a}(i)$ as the number of items in bucket i.

Given a bucket configuration $\mathbf{a}(1), \ldots, \mathbf{a}(k)$, *placing a new item in bucket p* transforms the configuration as follows: A new item is added to bucket p and all items in buckets higher than p are moved to bucket p. Buckets $i > p$ are unchanged. Formally the configuration $\mathbf{b}$ produced from $\mathbf{a}$ by the placement p satisfies:

- $\mathbf{b}(i) = \mathbf{a}(i)$ for $i > p$.
- $\mathbf{b}(p) = 1 + \sum_{i \leq p} \mathbf{a}(p)$, and
- $\mathbf{b}(i) = 0$ for $i < p$,

A *prefix bucketing* of n items into k buckets is a sequence $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ such that each $\mathbf{a}_t$ is a bucket configuration of t items into k buckets, and there is a sequence $p_1, \ldots p_n$ of integers in $\{1, \ldots, k\}$ such that for each $t \in \{1, \ldots, n\}$, $\mathbf{a}_t$ is obtained from $\mathbf{a}_{t-1}$ by placing a new item in bucket p_t according to the transformation defined above. The sequence $p_1, \ldots, p_n$ is called the *placement sequence* of the bucketing.

The *cost of the bucketing* $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ is $c(\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n) = \sum_{t=1}^{n} \mathbf{a}_t(p_t)$.

6.7 Connecting Bucketing to Online Labeling

In this section we show that given any lazy online labeling algorithm $\mathcal{A}$, we can use the adversary defined in the previous section to construct a prefix bucketing whose cost provides a lower bound on the cost of $\mathcal{A}$ in labeling the sequence $Y = \{y_1, \ldots, y_n\}$ produced by the adversary.

Fix a lazy online labeling algorithm $\mathcal{A}$ and for $1 \leq t \leq n$, let $\mathbf{f}_t, I_t(i), \mathbf{Rel}_t, q_t, y_t$ and $I_0(1)$ be as defined by the **Adversary**$(\mathcal{A}, n, m)$ (Figure 6.1). Recall that Y_n denotes the set $\{y_1, y_2, \ldots, y_n\}$.

Set $k = \lceil \log(m+1) \rceil$. For integer $i \in \{1, \ldots, k\}$ we define $\bar{i}$ to be $\bar{i} = (k+1) - i$. For $t = 0, 1, \ldots, n$ we define a sequence $(A_t(i) : 1 \leq i \leq k)$ of subsets of Y_n as follows: For all $i = 1, \ldots, k$, $A_t(i) = \emptyset$. For $t > 0$:

- $A_t(i) = A_{t-1}(i)$, for all $i \in \{\bar{q}_t + 1, \ldots, k\}$,
- $A_t(\bar{q}_t) = \{y_t\} \cup \bigcup_{i \leq \bar{q}_t} A_{t-1}(i)$, and
- $A_t(i) = \emptyset$, for all $i \in \{1, \ldots, \bar{q}_t - 1\}$.

For each $t \in \{1, \ldots, n\}$, let $\mathbf{a}_t = \mathbf{a}_t(1), \ldots, \mathbf{a}_t(k)$ be the sequence defined by $\mathbf{a}_t(i) = |A_t(i)|$. It is easy to see (by induction on t) that $\mathbf{a}_0, \ldots, \mathbf{a}_n$ is a prefix bucketing of n items into k buckets with placement sequence $\bar{q}_1, \ldots, \bar{q}_n$. The cost of this bucketing is $c(\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n) = \sum_{i=1}^{n} |A_t(\bar{q}_t)|$. Our next step is to relate this cost to the cost of online labeling.

We start by observing:

Lemma 6.7.1. *Let $k = \max\{\mathbf{depth}_{t'}, t' \in \{1, \ldots, n\}\}$. Then for any $t \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, \mathbf{depth}_t\}$, $A_t(\bar{i}) \subseteq I_t(i)$ and for $j \in \{\mathbf{depth}_t + 1, \ldots, k\}$ it holds that $A_t(\bar{j}) = \emptyset$.*

Proof. We prove the claim by induction on t. For $t = 1$, the only non-empty set is $A_1(k) = \{y_1\}$ so the claim is true. Let assume the claim is true for $t - 1 \geq 1$. Thus we know that $A_{t-1}(\bar{i}) \subseteq I_{t-1}(i)$ for $i \in \{1, \ldots, \mathbf{depth}_{t-1}\}$. Now we distinguish three cases:

- For $i \in \{q_t + 1, \ldots, k\}$: $A_t(\bar{i}) = \emptyset$ and thus it is surely a subset of $I_t(i)$.
- For $i = q_t$: $A_t(\bar{q}_t) = \{y_t\} \cup \bigcup_{j \leq \bar{q}_t} A_{t-1}(j)$. Since $A_{t-1}(j) \subseteq I_{t-1}(\bar{j}) \subseteq I_{t-1}(q_t)$ for each $j \leq \bar{q}_t$ and $I_t(q_t) = \{y_t\} \cup I_{t-1}(q_t)$ we are done.
- For $i \in \{1, \ldots, q_t - 1\}$: $A_t(\bar{i}) = A_{t-1}(\bar{i})$ while $I_t(i) = \{y_t\} \cup I_{t-1}(i)$ and therefore the claim follows immediately.

□

This lemma actually proves more than we need, but it reflects nicely the relation between the labeling and the bucketing.

However we only need to show, that for each $t \in \{1, \ldots, n\}$, $|I_{t-1}(q_t)| \geq |A_t(\bar{q}_t)| - 1$. This follows immediately from previous lemma and the fact that $I_t(q_t) = \{y_t\} \cup I_{t-1}(q_t)$. Combining this with Corollary 6.5.11 we obtain the following connection between the cost of online labeling and prefix bucketing. Recall that for constructing the sequence $(y_1, y_2, \ldots, y_n)$ we used $\mathbf{Adversary}(\mathcal{A}, n, m)$.

Lemma 6.7.2. *Let the prefix bucketing $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ be defined by $\mathbf{a}_t(i) = |A_t(i)|$, for all $t = 0, \ldots, n$ and $i = 1, \ldots, k$. The cost $c(\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n)$ of the bucketing satisfies:*

$$\chi_{\mathcal{A}}\left((y_1, y_2, \ldots, y_n), m\right) \geq \frac{1}{64} \cdot c(\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n) - \frac{9}{64} n.$$

6.8 Lower Bound for Bucketing

In this section we derive a lower bound (Lemma 6.8.13) on the cost of any prefix bucketing. To do so we map any prefix bucketing to a k-tuple of ordered rooted trees. We prove a lower bound on the sum of the depths of the nodes of the trees, and this will imply a lower bound for the cost of the bucketing.

Ordered trees An *ordered rooted tree* is a rooted tree where the children of each node are ordered from left to right. Since these are the only trees we consider, we refer to them simply as trees. The *i-th subtree of T* is the tree rooted in the i-th child of the root from the left. If the root has less than i children, we consider the i-th subtree to be empty. The number of nodes of T is called its *size* and is denoted $|T|$. The *depth* of a node is one more than its distance to the root, e.g., the root has depth 1. The depth of a tree is the maximum depth of its nodes. The *cost* of T, denoted $\kappa(T)$, is the sum of the depths of its nodes. The cost and size of an empty tree is defined to be zero.

To each prefix bucketing $\mathbf{a} = \mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ into k buckets, we associate a k-tuple of trees $T_{\mathbf{a}}(1), T_{\mathbf{a}}(2), \ldots, T_{\mathbf{a}}(k)$ inductively as follows: The trivial bucketing $\mathbf{a} = \mathbf{a}_0$ is mapped to the k-tuple of empty trees. For bucketing $\mathbf{a} = \mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ with placement sequence $q_1, \ldots, q_n$ let $\mathbf{a}'$ be the bucketing $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{n-1}$, and assume $T_{\mathbf{a}'}$ has been defined. We define $T_{\mathbf{a}'}$ by:

- For $1 \leq i < q_n$, $T_{\mathbf{a}}(i) = T_{\mathbf{a}'}(i)$.
- $T_{\mathbf{a}}(q_n)$ consists of a root node whose children are the non-empty trees among $T_{\mathbf{a}'}(q_n), T_{\mathbf{a}'}(q_n + 1), \ldots, T_{\mathbf{a}'}(k)$ ordered left to right by increasing index.
- $T_{\mathbf{a}}(i)$ is an empty tree for $q_n < i \leq k$.

We make several simple observations about the trees assigned to a bucketing. Recall, that for integer $i \in \{1, \ldots, k\}$ we define $\bar{i}$ to be $\bar{i} = (k+1) - i$

Proposition 6.8.1. *For any positive integer k, if $\mathbf{a} = \mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ is a prefix bucketing into k buckets then for each $i \in \{1, \ldots, k\}$, $|T_{\mathbf{a}}(i)| = \mathbf{a}_n(\bar{i})$.*

Proof. The proof is straightforward by induction on n. □

The next lemma relates the cost of bucketing to the cost of its associated trees.

Lemma 6.8.2. *For any positive integer k, if $\mathbf{a} = \mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ is a prefix bucketing into k buckets then $c(\mathbf{a}) = \sum_{i=1}^{k} \kappa(T_{\mathbf{a}}(i))$.*

Proof. By induction on n. For $n = 0$, both sides of the equality are 0. Suppose $n \geq 1$ and assume that the claim is true for $n-1$. Let $\mathbf{a}' = \mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{n-1}$ and q_n be as in the definition of prefix bucketing.

$$\begin{aligned} c(\mathbf{a}) = c(\mathbf{a}') + 1 + \sum_{i=1}^{\bar{q}_n} \mathbf{a}_{n-1}(i) &= \sum_{i=1}^{k} \kappa(T_{\mathbf{a}'}(i)) + 1 + \sum_{i=q_n}^{k} |T_{\mathbf{a}'}(i)| \\ &= \sum_{i=1}^{q_n - 1} \kappa(T_{\mathbf{a}}(i)) + 1 + \sum_{i=q_n}^{k} (\kappa(T_{\mathbf{a}'}(i)) + |T_{\mathbf{a}'}(i)|) \end{aligned}$$

where the second equality uses the induction hypothesis with Proposition 6.8.1, and the last equality follows from the definition of $T_{\mathbf{a}}(i)$ for $i = 1, \ldots, q_n - 1$. For $i \geq q_n$ the depth of each node in $T_{\mathbf{a}'}(i)$ increases by one when it becomes a child of $T_{\mathbf{a}}(q_n)$, hence

$$\kappa(T_{\mathbf{a}}(q_n)) = 1 + \sum_{i=q_n}^{k} (\kappa(T_{\mathbf{a}'}(i)) + |T_{\mathbf{a}'}(i)|)$$

For $i > q_n$, $\kappa(T_{\mathbf{a}}(i)) = 0$ so the lemma follows. □

Thus to get a lower bound on the cost of a prefix bucketing it suffices to prove a lower bound on the sum of the costs of the trees that occur in the associated k-tuple. The following definition will help describe the structure of trees that occur in such a k-tuple.

Definition 6.8.3 (k-admissible)**.** *Let k be a positive integer. The empty tree is k-admissible. A non-empty tree T is k-admissible if its root has at most k children and the i-th non-empty subtree of T is $(k+1-i)$-admissible.*

For example, T is 1-admissible if and only if T is empty or a rooted path. We collect some basic properties of k-admissibility.

Lemma 6.8.4. *Let T be a (rooted ordered) tree and $k \geq 1$, and suppose T is k-admissible. Let v be a leaf of T.*

1. *If $k' > k$ then T is k'-admissible.*
2. *If v is deleted from T then the resulting tree is k-admissible.*
3. *If a new node is added as a child of v then the resulting tree is k-admissible.*
4. *If T has at least two nodes and $k \geq 2$, then the tree obtained from T by removing its first subtree is $(k-1)$-admissible.*

Proof. The first and last parts are essentially immediate from the definition of k-admissibility. We prove the other two parts by induction on $|T|$.

Let T' be the tree resulting from deleting v and T'' be the tree resulting from adding a child to v. If $|T| = 1$ then T, T' and T'' are k-admissible for all $k \geq 1$. Suppose $|T| > 1$. Let v belong to the i-th subtree $T(i)$ of T. By definition of k-admissible, T_i is $(k-i+1)$-admissible, and by induction the corresponding subtree $T''(i)$ is also $(k-i+1)$-admissible. It follows immediately that T'' is k-admissible. If $T(i)$ consists of a single vertex, namely v, then by removing it, the j-th subtree of T (which must be $(k-j+1)$-admissible) becomes the $(j-1)$-st subtree of T' (and is $(k-(j-1)+1)$-admissible by the first part) so T' is k-admissible. Otherwise, the subtree $T'(i)$ that corresponds to $T(i)$ is non-empty and by induction it is $(k-i+1)$-admissible. Since all other subtrees of T' are the same as in T, T' is k-admissible. □

The connection of admissibility to prefix bucketing is given by the following proposition, which is obtained by an easy induction on n (using the first observation in Lemma 6.8.4):

Proposition 6.8.5. *For any positive integer k, if $\mathbf{a} = \mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ is a prefix bucketing into k buckets then for each $i \in \{1, \ldots, k\}$, $T_{\mathbf{a}}(i)$ is $(k+1-i)$-admissible.*

Let us define $\mu(n, k)$ to be the minimum cost of a k-admissible tree of n vertices.

Proposition 6.8.6. *For any bucketing $\mathbf{a}$ of n items into k buckets, we have $c(\mathbf{a}) \geq \mu(n, k) - n + 1$.*

Proof. Modify $\mathbf{a}$ to the bucketing $\mathbf{b} = \mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n$ where $\mathbf{b}_i = \mathbf{a}_i$ for $i < n$ and $\mathbf{b}_n = (0, 0, \ldots, 0, n)$. This corresponds to placing the final item in bucket 1. This can increase the cost by at most $n-1$ so $c(\mathbf{a}) \geq c(\mathbf{b}) - n + 1$. The first tree U in the k-tuple $T_{\mathbf{b}}(1)$ has size n and is k-admissible by Proposition 6.8.5. By Lemma 6.8.2, $c(\mathbf{b}) \geq \kappa(U)$ which is at least $\mu(n, k)$. □

It remains to give a lower bound on $\mu(n, k)$.

Lemma 6.8.7. *Let d be a positive integer and T be an arbitrary rooted tree with all leaves of depth at least d. Then $\kappa(T) \geq \frac{(d+1)}{2} \cdot |T|$.*

Proof. Let n_i denote the number of nodes of T at the depth $i = 1, \ldots, d-1$, and let n_d denote the number of all nodes at depth at least d. Then it holds $|T| = n_1 + n_2 + \cdots + n_d$. Since each node of T at depth $i < d$ has at least one child at depth $i+1$, we have $0 \leq n_1 \leq n_2 \leq \cdots \leq n_d$. Clearly, $c(T) \geq \sum_{i=1}^{d} i n_i$. Given the constraints, this sum is minimized (over reals) when $n_1 = n_2 = \cdots = n_d = |T|/d$. Thus $c(T) \geq \frac{d(d+1)}{2} \cdot \frac{|T|}{d}$. □

A k-admissible tree may have some leaves at low depth, so the above bound is not immediately useful, so we need the following:

Definition 6.8.8 (Balanced tree)**.** *A tree of depth d is balanced if all its leaves are of depth d or $d-1$.*

Lemma 6.8.9. *Let T be a k-admissible tree of size n having the minimum cost $\mu(n,k)$. Then T is balanced.*

Proof. Let d be the depth of T. Suppose T is unbalanced. Let u be a leaf of depth at most $d-2$ and let v be a leaf of depth d. Let T' be the tree obtained by removing v and reattaching v as a child of u. Then $\kappa(T') < \kappa(T)$, and by the second and third parts of Lemma 6.8.4, T' is k-admissible which contradicts the minimality of $\kappa(T)$. Thus T is balanced. □

Lemma 6.8.10. *Let $k, d \geq 1$. If T is a k-admissible tree of depth d, then $|T| \leq \binom{k+d-1}{k}$.*

Proof. If $k = 1$ then T must be a rooted path of depth d and clearly $|T| = d \leq \binom{d}{1}$. For $k \geq 2$, we prove the result by induction on $|T|$. If $|T| = 1$ the result is trivial so assume $|T| \geq 2$. Let L be the first subtree of T and let R be the tree created by removing L from T. By the definition of k-admissibility L is a k-admissible tree of depth at most $d-1$, and by the last part of Lemma 6.8.4, R is a $(k-1)$-admissible tree of depth at most d. By the induction hypothesis, $|T| = |L| + |R| \leq \binom{k+d-2}{k} + \binom{k+d-2}{k-1} = \binom{k+d-1}{k}$. □

Corollary 6.8.11. *For any $n \geq k \geq 1$, $\mu(n,k)$ is at least $n(w+1)/2$ where w is the smallest integer such that $\binom{k+w}{k} \geq n$.*

Proof. Let T be a k-admissible tree of size n having minimum cost. Let d be the depth of T. By Lemma 6.8.10 we have $\binom{k+d-1}{k} \geq n$ and so $d-1 \geq w$. By Lemma 6.8.9, T is balanced so all leaves are at depth at least $d-1 \geq w$ and so by Lemma 6.8.7, $\kappa(T) \geq (w+1)n/2$. □

Lemma 6.8.12. *Let n, k, w be integers such that $n \geq 2$ and $k \geq \log n$. If $\binom{k+w}{k} \geq n$, then $w \geq \frac{\log n}{4(\log 8k - \log\log n)}$.*

Proof. If $w \geq k$ then the conclusion holds, so assume $w \leq k$. Recall that $\log \binom{r}{s} \leq H(s/r)r$ where H stands for the binary entropy function defined on $[0,1]$ by $H(x) = x \log \frac{1}{x} + (1-x) \log \frac{1}{1-x}$, where the base of the logarithm is 2. For $x \in (0, 1/2]$, $H(x) \leq 2x \log \frac{1}{x}$. Therefore:

$$\begin{aligned}\log(n) \leq \log \binom{k+w}{k} = \log \binom{k+w}{w} &\leq (k+w) H\left(\frac{w}{k+w}\right) \\ &\leq 2w \log\left(\frac{k+w}{w}\right) \\ &\leq 2w \log\left(\frac{2k}{w}\right).\end{aligned}$$

Defining $a = \log(n)/w$, we get

$$a \leq 2 \log\left(\frac{2ak}{\log(n)}\right) = 2(\log(8k) + \log(a/4) - \log\log(n)).$$

Using $\log(x) < x$ we get

$$\begin{aligned}a &\leq 2(\log(8k) + a/4 - \log\log(n)) \\ \frac{a}{2} &\leq 2(\log(8k) - \log\log(n)),\end{aligned}$$

which implies the claimed bound on w. □

We now deduce the following lower bound on the cost of prefix bucketing:

Lemma 6.8.13. *Let k, n be positive integers such that $k \geq \log n$. The cost of any prefix bucketing of n items into k buckets is at least $\frac{n \log n}{8(\log 8k - \log\log n)} - n$.*

Proof. By Proposition 6.8.6, the cost of any such bucketing is at least $\mu(n,k) - n$, and by Corollary 6.8.11, this is at least $\frac{nw}{2} - n$ where w is the smallest integer such that $\binom{k+w}{k} \geq n$, which is at least $\frac{\log n}{4(\log 8k - \log\log n)}$ by Lemma 6.8.12. □

Proof of Lemma 6.5.1. By Lemma 6.7.2, the cost incurred by a lazy algorithm on the sequence $y_1, y_2, \ldots, y_n$ (constructed by $\mathbf{Adversary}(\mathcal{A}, n, m)$) is lower-bounded by

$$\chi_{\mathcal{A}}\left((y_1, y_2, \ldots, y_n), m\right) \geq \frac{1}{64} \cdot c(\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n) - \frac{9}{64} n$$

where $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ is a certain prefix bucketing of n items into $k = \lceil \log(m+1) \rceil$ buckets (Claim 6.5.6). The previous lemma provides a lower bound on the cost of any such bucketing. Using that lower bound we get:

$$\chi_{\mathcal{A}}\left((y_1, y_2, \ldots, y_n), m\right) \geq \frac{1}{512} \cdot \frac{n \log n}{3 + \log\lceil \log(m+1) \rceil - \log\log n} - \frac{n}{6}.$$

□

7. Randomized Online Labeling with Polynomially Many Labels

7.1 Introduction

In this chapter we prove an $\Omega(n \cdot \log(n))$ lower bound on the expected number of moves for inserting n items for *randomized* online labeling algorithms. This lower bound is valid for m between cn and n^c ($c > 1$) for any *randomized* online labeling algorithm. For $m = n^c$, this matches the known deterministic upper bounds up to constant factors, and thus randomization provides no more than a constant factor advantage over determinism.

Prior to our joint work with Michal Koucký and Michael Saks [11], on which this chapter is based, all lower bound proofs considered only deterministic algorithms. There are however many online problems where randomized algorithms perform provably better than deterministic ones. For example, the best deterministic algorithm for the paging problem with k pages has competitive ratio k but there are randomized algorithms having competitive ratio $\Theta(\log(k))$ [8]. Thus it is a natural question whether there exists randomized algorithms which are (in expectation) better than deterministic.

Thorough this chapter we use a model in which the cost of a randomized labeling algorithm is the worst case over all input sequences of a given length n of the expected number of moves made by the algorithm. This corresponds to running the algorithm against an *oblivious adversary* (see [8]) who selects the input sequence having full knowledge of the algorithm, but not of the random bits flipped in the execution of the algorithm.

Unlike many other lower bounds for non-uniform computation models, our proof does not use Yao's principle. Yao's principle says (roughly) that to prove a lower bound on the expected cost of an arbitrary randomized algorithm it suffices to fix a distribution over inputs, and prove a lower bound on the expected cost of a deterministic algorithm against the chosen distribution. Rather than use Yao's principle, our proof takes an arbitrary randomized algorithm and selects a (deterministic) sequence that is hard for that algorithm.

The construction and analysis of the hard sequence follow the same overall strategy of the lower bound in Chapter 6 for deterministic algorithms in the case of polynomially many labels. This involves relating online labeling to a *bucketing games*. We define a map (an *adversary*) which associates to a labeling algorithm $\mathcal{A}$ a hard sequence of items. We then show that the behavior of the algorithm on this hard sequence can be associated to a strategy for playing a particular bucketing

game, such that the cost incurred by the algorithm on the hard sequence is bounded below by the cost of the associated bucketing game strategy. Finally we prove a lower bound on the cost of any strategy for the bucketing game, which therefore gives a lower bound on the cost of the algorithm on the hard input sequence.

In extending this argument from the case of deterministic algorithms to the randomized case, each part of the proof requires significant changes. The adversary which associates an algorithm to a hard sequence requires various careful modifications. The argument that relates the cost of $\mathcal{A}$ on the hard sequence to the cost of an associated bucketing strategy does not work for the original version of the bucketing game, and we can only establish the connection to a new variant of the bucketing game called tail-bucketing. Finally the lower bound proof on the cost of any strategy for tail-bucketing is quite different from the previous lower bound for the original version of bucketing.

7.2 The Main Theorem

In this section we state the main theorem of this chapter. Recall that randomized online labeling algorithm $\mathcal{A}$ is a probability distribution on deterministic online labeling algorithms. The cost $\chi_{\mathcal{A}}(n, m, r)$ is the expected cost of the algorithm sampled from this distribution.

Theorem 7.2.1. *For any constant C_0, there are positive constants C_1 and C_2 so that the following holds. Let $\mathcal{A}$ be a randomized algorithm with parameters (n, m, r), where $n \geq C_1$, $r \geq 2^n - 1$ and $m \leq n^{C_0}$. Then $\chi_{\mathcal{A}}(n, m, r) \geq C_2 n \log(n)$.*

This theorem will be an immediate consequence of Lemma 7.5.1 in Section 7.5.

7.3 Mapping a Randomized Algorithm to a Hard Input Sequence

We now give an overview of the adversary which maps an algorithm to a hard input sequence $y_1, \ldots, y_n$. The adversary is deterministic. Its behavior will be determined by the expected behavior of the randomized algorithm. Even though we are choosing the sequence obliviously, without seeing the actual responses of the algorithm, we view the selection of the sequence in an online manner. We design the sequence item by item. Having selected the first $t - 1$ items, we use the known randomized algorithm to determine a probability distribution over the sequence of labellings determined by the algorithm after each step. We then use this probability distribution to determine the next item, which we select so as to ensure that the expected cost incurred by the algorithm is large.

The adversary will maintain an *interval chain* consisting of a nested sequence of intervals of items inserted so far. The chain serves a dual purpose: the chain after step t is used by the adversary to select the item inserted at step $t+1$, and the sequence of chains over time provides a way to lower bound the total (expected) cost incurred by the algorithm.

This chain is denoted[1]

$$I_t(1) \supset T_t(2) \supset I_t(2) \supset T_t(3) \supset \cdots \supset T_t(d) \supset I_t(d).$$

The interval $I_t(1)$ equals $Y_t \cup \{\min_U, \max_U\}$, the final interval $I_t(d)$ has between 2 and 6 elements. The next item to be inserted is selected to be an item that is between two items in the final interval $I_t(d)$.

The chain at step t is constructed as follows. The chain for $t = 0$ has $\mathbf{depth}_0 = 1$ and $I_0(1) = \{\min_U, \max_U\}$. The chain at step $t \geq 1$ is constructed based on the chain at the previous step $t-1$ and the expected behavior of the algorithm on $y_1, \ldots, y_t$.

We build the intervals for the chain at step t in order of increasing level (i.e., decreasing size). Intervals are either *preserved* (carried over from the previous chain, with the addition of y_t) or *rebuilt*. To specify which intervals are preserved, we specify a *critical level for step* t, q_t which is at most the depth $\mathbf{depth}_{t-1}$ of the previous chain. We'll explain the choice of q_t below. At step t, the intervals $T_t(i)$ and $I_t(i)$ for $i \leq q_t$ are *preserved*, which means that it is obtained from the corresponding interval at step $t-1$ by simply adding y_t. The intervals $T_t(i)$ and $I_t(i)$ for $i \geq q_t$ are *rebuilt*. The rule for rebuilding the chain for $i > q_t$ is defined by induction on i as follows: Given $I_t(i-1)$, $T_t(i)$ is defined to be either the first or second half of $I_t(i-1)$, depending on which of these intervals is more likely to have a smaller range of labels (based on the distribution over labels determined by the given algorithm). More precisely, we look at the median item of $I_t(i-1)$ and check whether (based on the randomized labeling) it is more likely that its label is closer to the label of the minimum or to the maximum element of $I_t(i-1)$. If the median is more likely to have label close to the minimum we pick the first half as $T_t(i)$ otherwise the second half. Having chosen $T_t(i)$, we take $I_t(i)$ to be the middle third of items in $T_t(i)$. This process terminates when $|I_t(i)| < 7$ and the depth $\mathbf{depth}_t$ of the chain is set to this final i. The adversary selects the next requested item y_{t+1} to be between two items in $I_t(\mathbf{depth}_t)$.

This construction of the chain is similar (except we do not have two types of segments) to that used in Chapter 6 in the deterministic case. An important difference comes in the definition of the critical level q_t. In the deterministic case the critical level is the smallest index i such that neither endpoint of $I_t(i)$ was

[1]Recall, we use subscript to denote step and we use $(\cdot)$ notation to denote a particular coordinate of such a vector or sequence at that step.

moved by the algorithm when inserting y_t. In the randomized case we need a probabilistic version of this: the critical level is the smallest index i such that the probability that either endpoint of $T_{t-1}(i)$ was moved since the last step when it was rebuilt is less than $1/4$.

One of the crucial requirements in designing the adversary is that the chain never grows too deep. Note that when we rebuild $T_t(i)$ it's size is at most $|I_t(i-1)|/2$ and when we rebuild $I_t(i)$ its size is at most $|T_t(i)|/3$. This suggests that as we proceed through the chain each interval is at most $1/2$ the size of the previous and so the depth is at most $\log(n)$. This reasoning is invalid because during a sequence of steps in which an interval in the chain is not rebuilt its size grows by 1 at each step and so the condition that the interval is at most half the size of its predecessor may not be preserved. Nevertheless we can show that the depth never grows to more than $4\log(m+1)$ levels.

In the deterministic case, Lemma 5.4.2 and the definition of the critical level q_t can be used to show that when the algorithm responds to the item y_t it moved at least a constant fraction of the items belonging to $I_{t-1}(q_t)$ and so the total cost of the algorithm is at least $DLB = \Omega(\sum_t |I_{t-1}(q_t)|)$ (Corollary 6.5.11). In the randomized case we get a related bound that the expected total number of moves is $RLB = \Omega(\sum_t |I_{t-1}(q_t) \setminus I_{t-1}(q_t+1)|)$ (Lemma 7.5.3). So the cost incurred by the algorithm is related to the extent of changes in the chain.

7.4 Bucketing Game

The next step in the analysis is to define bucketing games, and to show that the lower bound on the cost of the algorithm given in the previous paragraph is an upper bound on the cost of an appropriate bucketing game.

Recall, the prefix bucketing game with n items and k buckets is a one player game. The game starts with k empty buckets indexed $1, \ldots, k$. At each step the player places an item in some bucket p. All the items from buckets $1, \ldots, p-1$ are then moved into bucket p as well, and the cost is the number of items in buckets $1, \ldots, p$ before the merge, which is the number of items in bucket p after the merge. The total cost is the sum of the costs of each step. The goal is to select the sequence of indexes p so that we would minimize the total cost. In Chapter 6 (see also [2]) it is shown that any deterministic labeling algorithm could be associated to a bucketing strategy such that the cost of the labeling algorithm against our adversary is at least a constant times the cost of the bucketing strategy. This result is deduced using the lower bound of $\Omega(\sum_t |I_{t-1}(q_t)|)$ for the cost of the algorithm mentioned earlier. It was also shown in Chapter 6 (see also [2]) that the minimal cost of any bucketing strategy (for more than $2\log(n)$ buckets) is $\Omega(n\log(n)/(\log(k) - \log\log(n))$. These results together gave the lower bound on

deterministic labeling.

We use the same basic idea for the randomized case, but require several significant changes to the game. The first difficulty is that the lower bound on the cost of the randomized algorithm stated earlier, RLB, is not the same as the lower bound DLB that was known for deterministic algorithms. While DLB was shown to be at least the minimal cost of the prefix bucketing, this is not true for RLB. To relate RLB to bucketing, we must replace the cost function in bucketing by a smaller cost function, which is the number of items in the bucket p *before* the merge, not after. In general, this cost function is less expensive (often much less expensive) than the original cost function and we call it the *cheap* cost function. The argument relating the cost of a randomized algorithm to a bucketing strategy requires that the number of buckets be at least $4\log(m)$ buckets. If we could prove a lower bound on the cost of bucketing under the cheap function similar to the bound mentioned above for the original function this would be enough to deduce the desired lower bound on randomized labeling. However with this cheap cost function this lower bound fails: if the number of buckets is at least $1+\log(n)$, there is a bucketing strategy that costs 0 with the cheap cost function! (For example a strategy which always picks p to be the smallest index of an empty bucket has cost zero; it emulates incrementing a binary counter.) So this will not give any lower bound on the cost of a randomized labeling algorithm

We overcome this problem by observing that we may make a further modification of the rules for bucketing and still preserve the connection between the cost of a randomized algorithm against our adversary and the cheap cost of a bucketing. This modification is called *tail bucketing*. In a tail bucketing, after merging all the items into the bucket p, we redistribute these items back among buckets $1, \ldots, p$, so that bucket p keeps $1-\beta$ fraction of the items and passes the rest to the bucket $p-1$, bucket $p-1$ does the same, and the process continues down until bucket 1 which keeps the remaining items. It turns our that our adversary can be related to tail bucketing for $\beta = 1/6$. We can prove that the minimal *cheap* cost of tail bucketing is $\Omega(n\log(n))$ when $k = O(\log(n))$. This lower bound is asymptotically optimal and yields a similar bound for randomized online labeling.

The lower bound proof for the cheap cost of tail bucketing has some interesting twists. The proof consists of several reductions between different versions of bucketing. The reductions show that we can lower bound the cheap cost of tail bucketing with $C\log(n)$ buckets (for any C) by the cheap cost of ordinary prefix bucketing with $k = \frac{1}{4}\log(n)$ buckets. Even though the cheap cost of ordinary bucketing dropped to 0 once $k = \log(n) + 1$, we are able to show that for $k = \frac{1}{4}\log(n)$ there is a $\theta(n\log(n))$ bound for ordinary bucketing with the cheap cost.

7.5 Adversary Construction

We now specify an adversary **Adversary**$(\mathcal{A}, n, m)$ which given an online labeling algorithm $\mathcal{A}$, a length n, and label space size m, constructs a item sequence $y_1, y_2, \ldots, y_n$ from the universe $U = \{1, \ldots, 2^n - 1\}$. We use similar notation as in Chapter 6.

We think of the adversary as selecting $y_1, \ldots, y_n$ online, but after each step the adversary only knows a probability distribution over the configurations of the algorithm. It is important to keep in mind that the adversary knows the randomized algorithm $\mathcal{A}$ but does not know the random coins of the algorithm.

During the construction of the adversary sequence $y_1, \ldots, y_n$, the adversary will maintain a nested sequence of intervals of $\{y_1, \ldots, y_t\} \cup \{\min_U, \max_U\}$:

$$I_t(1) \supset T_t(2) \supset I_t(2) \supset T_t(3) \supset \cdots \supset T_t(\mathbf{depth}_t) \supset I_t(\mathbf{depth}_t).$$

called the *interval chain at step* t. Each of the intervals will be of size at least 2 and it will form and interval of $Y_t \cup \{\min_U, \max_U\}$. The depth $\mathbf{depth}_t$ of the chain may vary with t. The intervals $I_t(i)$ and $T_t(i)$ are said to be at *level* i in the chain.

To avoid having to deal with special cases in the description of the adversary, recall that we assume the existence of items $\min_U$ and $\max_U$ which are inserted to positions 0 and $m + 1$ for no cost and which cannot be moved by the algorithm. At the beginning of step t, having chosen items Y_{t-1} and having constructed the chain $I_{t-1}(1) \supset \cdots \supset I_{t-1}(\mathbf{depth}_{t-1})$, the adversary will select y_t to be $\min(I_{t-1}(\mathbf{depth}_{t-1}) + 2^{n-t})$. It is easy to see by induction on t that the items belonging to $Y_t \cup \{\min_U, \max_U\}$ are multiples of 2^{n-t}, and it follows that y_t is strictly between the smallest and second smallest elements of $I_{t-1}(\mathbf{depth}_{t-1})$. Therefore all of the chosen items are distinct.

We need to specify how we determine the chain at step t. The pseudo-code for the adversary is given in Figure 7.1.

The chain for $t = 0$ has $\mathbf{depth}_0 = 1$ and $I_0(1) = \{0, 2^n\}$. The chain at step $t \geq 1$ is constructed based on the chain at the previous step $t - 1$ and the expected behavior of the algorithm on $y_1, \ldots, y_t$ as reflected by the joint probability distribution over the sequence of functions $\mathbf{f}_{\mathcal{A},1}, \ldots, \mathbf{f}_{\mathcal{A},t}$.

We build the intervals for the chain at step t in order of increasing level (i.e., decreasing size). Intervals are either *preserved* (carried over from the previous chain, with the addition of y_t) or *rebuilt*. To specify which intervals are preserved, we specify a *critical level* q_t *for step* t, which is at least 1 and at most the depth $\mathbf{depth}_{t-1}$ of the previous chain. We'll explain the choice of q_t below. At step t, the intervals $I_t(i)$ for $i \leq q_t$ are *preserved*, which means that $I_t(i)$ is obtained simply by adding y_t to $I_{t-1}(i)$, and the rest are *rebuilt*. In particular, for $t \geq 7$, $I_t(1)$ is always preserved, and is equal to $Y_t \cup \{\min_U, \max_U\}$. The rule for rebuilding the chain

for $i > q_t$ is defined by induction on i as follows: If $|I_t(i-1)| < 7$ then the chain is terminated with $\mathbf{depth}_t = i-1$. Otherwise, consider the labeling of $I_t(i-1)$ by $\mathbf{f}_t$ (which is randomly distributed depending on $\mathcal{A}$). If the probability that $I_t(i-1)$ is left-leaning with respect to $\mathbf{f}_t$ is at least $1/2$, then set $T_t(I) = \textbf{left-half}(I_t(i-1))$ otherwise $T_t(i) = \textbf{right-half}(I_t(i-1))$. Set $I_t(i) = \textbf{middle-third}(T_t(i-1)))$. Observe that since $|I_t(i-1)| \geq 7$, we have $|T_t(i)| \geq 4$ and $|I_t(i)| \geq 2$.

Here we use the following notation. Let $I \subseteq Y \subseteq U$. We write $\text{med}(I)$ for the *median* of I which we take to be the $\lceil |I|/2 \rceil$-th largest element of I. We define $\textbf{left-half}(I) = \{y \in I | y \leq \text{med}(I)\}$ and $\textbf{right-half}(I) = \{y \in I | y \geq \text{med}(I)\}$ (note that $\text{med}(I)$ is contained in both). Also define $\textbf{left-third}(I)$ to be the smallest $\lfloor |I|/3 \rfloor$ elements, $\textbf{right-third}(I)$ to be the largest $\lfloor |I|/3 \rfloor$ elements and $\textbf{middle-third}(I) = I - \textbf{left-third}(I) - \textbf{right-third}(I)$. Given a labeling $\mathbf{f}$ of Y and an item interval I of Y, we say that I is *left-leaning* with respect to $\mathbf{f}$ if $\text{med}(I)$ has a label that is closer to the label of $\min(I)$ than it is to the label of $\max(I)$, i.e. $\mathbf{f}(\text{med}(I)) - \mathbf{f}(\min(I)) \leq \mathbf{f}(\max(I)) - \mathbf{f}(\text{med}(I))$. It is *right-leaning* otherwise.

It remains to explain how the critical level q_t is selected. When constructing each interval $I_t(i)$ of the chain for $i \geq 2$, the adversary defines a parameter $\mathbf{birth}_t(i)$ which is set to t if $I_t(i)$ is rebuilt, and is otherwise set to $\mathbf{birth}_{t-1}(i)$. It is easy to see (by induction on t), that $\mathbf{birth}_t(i)$ is equal to the largest step $u \leq t$ such that $I_u(i)$ was rebuilt. It follows that for each $u \in \{\mathbf{birth}_t(i), \ldots, t\}$, $\min(T_u(i)) = \min(T_t(i))$ and $\max(T_u(i)) = \max(T_t(i))$.

Say that item y has *stable label* during interval $\{a, \ldots, b\}$ if the label $\mathbf{f}_u(y)$ is the same for all u in $\{a, \ldots, b\}$, and has *unstable label* on $\{a, \ldots, b\}$ otherwise. We define the event $\mathbf{stable}_t(i)$ to be the event (depending on $\mathcal{A}$) that both $\min(T_t(i))$ and $\max(T_t(i))$ have stable labels during interval $\{\mathbf{birth}_{t-1}(i), \ldots, t\}$.

We are finally ready to define q_t. If there is at least one level $i \geq 2$ for which $\mathbf{Pr}[\mathbf{stable}_t(i)] \leq 3/4$, let $i_{\min}$ be the least such level, and choose $q_t = i_{\min} - 1$. Otherwise set $q_t = \mathbf{depth}_{t-1}$.

Before we describe the $\textbf{Adversary}(\mathcal{A}, n, m)$ (Figure 7.1) we first introduce some notation. Let $I \subseteq Y \subseteq U$. We write $\text{med}(I)$ for the *median* of I which we take to be the $\lceil |I|/2 \rceil$-th largest element of I. We define $\textbf{left-half}(I) = \{y \in I | y \leq \text{med}(I)\}$ and $\textbf{right-half}(I) = \{y \in I | y \geq \text{med}(I)\}$ (note that $\text{med}(I)$ is contained in both). Also define $\textbf{left-third}(I)$ to be the smallest $\lfloor |I|/3 \rfloor$. Given a labeling $\mathbf{f}$ of Y and an item interval I of Y, we say that I is *left-leaning* with respect to $\mathbf{f}$ if $\text{med}(I)$ has a label that is closer to the label of $\min(I)$ than it is to the label of $\max(I)$, i.e. $\mathbf{f}(\text{med}(I)) - \mathbf{f}(\min(I)) \leq \mathbf{f}(\max(I)) - \mathbf{f}(\text{med}(I))$. It is *right-leaning* otherwise.

We will prove the following lemma about the adversary, which together with Lemma 5.4.2 immediately implies Theorem 7.2.1.

Adversary$(\mathcal{A}, n, m)$

- $I_0(1) \longleftarrow \{0, 2^n\}$, $\mathbf{depth}_0 \longleftarrow 1$
- For $t = 1, \ldots, n$ do
  - $y_t \longleftarrow \min(I_{t-1}(\mathbf{depth}_{t-1})) + 2^{n-t}$

  {***Choose Critical Level***}

  - Consider the sequence of (dependent) random functions $\mathbf{f}_1, \ldots, \mathbf{f}_t$ produced by $\mathcal{A}$ in response to $y_1, \ldots, y_t$. If there is an index $i \geq 2$ for which $\mathbf{Pr}[\mathbf{stable}_t(i)] \leq 3/4$, let $i_{\min}$ be the least such index and let $q_t = i_{\min} - 1$. Otherwise set $q_t = \mathbf{depth}_{t-1}$.
  - $I_t(1) \longleftarrow I_{t-1}(1) \cup \{y_t\}$
  - $i \longleftarrow 2$

  {***Preservation Rule***}

  - While $i \leq q_t$ do:
    - $T_t(i) \longleftarrow T_{t-1}(i) \cup \{y_t\}$
    - $I_t(i) \longleftarrow I_{t-1}(i) \cup \{y_t\}$
    - $\mathbf{birth}_t(i) \longleftarrow \mathbf{birth}_{t-1}(i)$
    - $i \longleftarrow i + 1$

  {***Rebuild Rule***}

  - While $|I_t(i)| \geq 7$ do:
    - If $I_t(i-1)$ is left-leaning with respect to $\mathbf{f}_t$ with probability at least $1/2$
      - $T_t(i) \longleftarrow \mathbf{left\text{-}half}(I_t(i-1))$
    - otherwise
      - $T_t(i) \longleftarrow \mathbf{right\text{-}half}(I_t(i-1))$
    - $I_t(i) \longleftarrow \mathbf{middle\text{-}third}(T_t(i))$

    {***Record that*** $I_t(i)$ ***and*** $T_t(i)$ ***were rebuilt***}

    - $\mathbf{birth}_t(i) \longleftarrow t$
    - $i \longleftarrow i + 1$
  - $\mathbf{depth}_t \longleftarrow i - 1$

Output: $y_1, y_2, \ldots, y_n$

Figure 7.1: Pseudocode for the adversary

Lemma 7.5.1. *Let $c \geq 1$ be an arbitrary constant and n, m be large enough integers such that $m < (n+1)^c$. Let $\mathcal{A}$ be a lazy randomized online labeling algorithm with the range m. Let $y_1, y_2, \ldots, y_n$ be the output of* **Adversary**$(\mathcal{A}, n, m)$ *(Figure 7.1). Then the cost satisfies:*

$$\chi_{\mathcal{A}}\left((y_1, y_2, \ldots, y_n), m\right) \geq \frac{1}{96}\left(\frac{1}{6}\right)^{512c^2}(n+1)\log(n+1) - \frac{n}{4}.$$

The proof of this lemma has two main steps. The first step is to bound the cost $\chi_{\mathcal{A}}\left((y_1, y_2, \ldots, y_n), m\right)$ from below by the minimum cost of a variant of the prefix-bucketing game. The variant of the game we consider is called *tail-bucketing*. The second step is to give a lower bound on the cost of tail-bucketing.

To prove the first step we will need two properties of **Adversary**$(\mathcal{A}, n, m)$. **Adversary**$(\mathcal{A}, n, m)$ determines $y_1, y_2, \ldots, y_n$ and the critical levels $q_1, \ldots, q_n$.

Lemma 7.5.2. *For any $t \in \{1, \ldots, n\}$,* $\mathbf{depth}_t \leq 4\log(m+1)$.

Lemma 7.5.3. *The cost of $\mathcal{A}$ on $y_1, y_2, \ldots, y_n$ satisfies:*

$$\chi_{\mathcal{A}}\left((y_1, y_2, \ldots, y_n), m\right) \geq \frac{1}{40}\sum_t |I_{t-1}(q_t) \setminus I_{t-1}(q_t+1)|,$$

where the sum ranges over steps $t \in \{1, \ldots, n\}$ for which $q_t < \mathbf{depth}_{t-1}$.

For the proofs of these two lemmas we need certain random variables associated with the execution of $\mathcal{A}$ on $y_1, y_2, \ldots, y_n$. Since all of the randomness comes from the distribution over $\mathcal{A}$, the value of each random variable is determined by the random selection of $\mathcal{A}$, and we sometimes subscript random variables by $\mathcal{A}$ to emphasize this dependence. (We think of randomized algorithms as a probability distribution of deterministic algorithms.) Furthermore, we replace $\mathcal{A}$ by a deterministic algorithm A in the subscript to indicate the value of the random variable when $\mathcal{A} = A$. We make the following definitions.

- For a pair (i, t) such that $i < \mathbf{depth}_t$, $\mathbf{shrink}_{\mathcal{A},t}(i)$ is the 0-1 indicator of the event that
$$\mathbf{span}_{\mathcal{A},t}(I_t(i+1)) \leq \mathbf{span}_{\mathcal{A},t}(I_t(i))/2.$$
- Define $\mathbf{shrink}_{\mathcal{A},t} = \sum_{i=1}^{\mathbf{depth}_t - 1} \mathbf{shrink}_{\mathcal{A},t}(i)$.

Proof of Lemma 7.5.2. For $t = 1$ the claim is trivial so assume $t > 1$. For any algorithm A, $\mathbf{span}_{A,t}(I_t(1)) = m+1$ and $\mathbf{span}_{A,t}(I_t(\mathbf{depth}_t)) \geq 2$, and $\mathbf{span}_{A,t}(I_t(i)) > \mathbf{span}_{A,t}(I_t(i+1))$ for $i \in \{1, \ldots, \mathbf{depth}_{t-1}\}$. Therefore $\mathbf{shrink}_{A,t}(i)$ can be 1 for at most $\log(m+1) - 1$ values of i. Thus $\mathbf{shrink}_{A,t} \leq \log(m+1) - 1$.

Next we claim and prove below that for $i \in \{1, \ldots, \mathbf{depth}_t - 1\}$, $\mathbf{Pr}[\mathbf{shrink}_{\mathcal{A},t}(i) = 1] \geq 1/4$. This claim implies $\mathbf{E}[\mathbf{shrink}_{\mathcal{A},t}] \geq (\mathbf{depth}_t - 1)/4$ which then gives $\mathbf{depth}_t \leq 4\log(m+1)$ to complete the proof of the lemma.

So it remains to prove the claim. Consider first the case that $i+1 > q_t$. Intervals $T_t(i+1)$ and $I_t(i+1)$ are rebuilt at step t. By definition of the adversary $T_t(i+1)$ is either $\mathbf{left\text{-}half}(I_t(i))$ or $\mathbf{right\text{-}half}(I_t(i))$. Furthermore this choice is made so that $\mathbf{span}_t(T_t(i+1)) \leq \mathbf{span}_t(I_t(i))/2$ with probability at least $1/2$ and since $I_t(i+1) \subseteq T_t(i+1)$, $\mathbf{Pr}[\mathbf{shrink}_t(i) = 1] \geq 1/2$.

Next consider the case that $i+1 \leq q_t$ so that $T_t(i+1)$ and $I_t(i+1)$ are preserved at step t. These intervals were most recently rebuilt at step $s = \mathbf{birth}_t(i+1) = \mathbf{birth}_{t-1}(i+1)$ and the endpoints of $T_u(i+1)$ are the same for all $u \in \{s, \ldots, t\}$. Since $i+1 > 1$, $s > 1$. Since $i+1 > q_s$, $\mathbf{Pr}[\mathbf{shrink}_s(i) = 1] \geq 1/2$. We now claim and prove below that if both $\mathbf{shrink}_s(i)$ and $\mathbf{stable}_t(i+1)$ happen then $\mathbf{shrink}_t(i)$ happens. From this claim, and the assumption that $i+1 \leq q_t$ we deduce: $\mathbf{Pr}[\mathbf{shrink}_t(i)] \geq \mathbf{Pr}[\mathbf{shrink}_s(i) \cap \mathbf{stable}_t(i+1)] \geq \mathbf{Pr}[\mathbf{shrink}_s(i)] + \mathbf{Pr}[\mathbf{stable}_t(i+1)] - 1 \geq 1/2 + 3/4 - 1 = 1/4$, as required.

To see the final claim, assume that the event $\mathbf{stable}_t(i+1)$ occurred. For each endpoint of $T_t(i+1)$, its label remained the same under each of the functions $\mathbf{f}_s, \ldots, \mathbf{f}_t$, and by the laziness of the algorithm, it also happened that for each endpoint of $I_t(i)$, its label remained the same under each of the functions $\mathbf{f}_s, \ldots, \mathbf{f}_t$. Thus if, in addition, $\mathbf{shrink}_s(i)$ happens then so does $\mathbf{shrink}_t(i)$. □

Proof of Lemma 7.5.3. An *item-step pair* (y, u) is a pair where $y \in I_u(1)$. For each step t such that $q_t < \mathbf{depth}_{t-1}$ we will define a set W_t of item-step pairs. The sets W_t will be disjoint for different steps t and will consist of some set of item-step pairs (y, u) with $u \leq t$. Say that the item-step pair (y, u) is a *relabel event* if $\mathbf{f}_u(y) \neq \mathbf{f}_{u-1}(y)$. Define $\mathbf{relabs}_t$ be the (random) number of relabel events in W_t. It follows that the cost of the algorithm is at least $\sum_{t:q_t<\mathbf{depth}_{t-1}} \mathbf{E}[\mathbf{relabs}_t]$. We will show that $\mathbf{E}[\mathbf{relabs}_t] \geq \frac{1}{40}|I_t(q_t) \setminus I_t(q_t+1)|$, which will suffice to prove the lemma.

We now define W_t for each t such that $q_t < \mathbf{depth}_{t-1}$. Let $t_b = \mathbf{birth}_{t-1}(1+q_t)$. For all steps $u \in \{t_b+1, \ldots, t-1\}$ the intervals $T_u(1+q_t)$ are preserved and also the intervals $I_u(1+q_t)$ are preserved and so from step $u-1$ to u they each change only by the addition of y_u. Defining for all steps s and levels i, $\Delta_s(i) = T_s(i) \setminus I_s(i)$, we have that the sets $\Delta_u(1+q_t)$ are all the same for each $u \in \{t_b, \ldots, t-1\}$. We define W_t to be the set of pairs (y, u) with $y \in \Delta_{u-1}(1+q_t)$ and $u \in \{t_b+1, \ldots, t\}$, i.e., $W_t = \Delta_{t-1}(1+q_t) \times \{t_b+1, \ldots, t\}$.

We now show that the sets W_t and $W_{t'}$ are disjoint for all pairs of steps $t < t'$. Suppose for contradiction that $W_t \cap W_{t'} \neq \emptyset$. Let $t'_b = \mathbf{birth}_{t'-1}(1+q_{t'})$. Then $\{t_b+1, \ldots, t\} \cap \{t'_b+1, \ldots, t'\} \neq \emptyset$ and so $\mathbf{birth}_{t'-1}(1+q_{t'}) = t'_b < t$. This means that level $1+q_{t'}$ is not rebuilt at step t but level $1+q_t$ is rebuilt at step t, so $q_t > q_{t'}$.

But then this contradicts $\Delta_{t-1}(1+q_t) \cap \Delta_{t'-1}(1+q_{t'}) \neq \emptyset$ since $\Delta_{t-1}(1+q_t) \subset T_{t-1}(1+q_t) \subset I_{t-1}(1+q_{t'}) \subset I_{t'-1}(1+q_{t'})$ while $\Delta_{t'-1}(1+q_{t'}) \cap I_{t'-1}(1+q_{t'})) = \emptyset$.

Finally, let us bound $\mathbf{E}[\mathbf{relabs}_t]$ from below. By the definition of the adversary $\Delta_{t-1}(1+q_t)$ is the union of the two equal-sized intervals $\mathbf{left\text{-}third}(T_{t_b}(1+q_t)) \cup \mathbf{right\text{-}third}(T_{t_b}(1+q_t))$. By the definition of q_t, the probability that both $\min(T_{t_b}(1+q_t))$ and $\max(T_{t_b}(1+q_t))$ have stable label during $\{t_b, \ldots, t\}$ is at most $3/4$. By the laziness of the algorithm, on any run in which the left (resp. right) endpoint of $T_{t-1}(1+q_t)$ has unstable label during $\{t_b, \ldots, t\}$ all items in $\mathbf{left\text{-}third}(T_{t_b}(1+q_t))$ (resp. $\mathbf{right\text{-}third}(T_{t_b}(1+q_t))$) have unstable label during $\{t_b, \ldots, t\}$ and so at least half the items of $\Delta_{t-1}(1+q_t)$ have unstable label during $\{t_b, \ldots, t\}$. Since this occurs with probability at least $1/4$, thus the expected number of relabel events is at least $|\Delta_{t-1}(1+q_t)|/8$.

To complete the proof of the lemma, we show that $|\Delta_{t-1}(1+q_t)| \geq \frac{1}{5}|I_{t-1}(q_t) \setminus I_{t-1}(1+q_t)|$. The sets $I_u(q_t) \setminus I_u(1+q_t)$ are the same for all $u \in \{b_t, \ldots, t-1\}$ and the same is true for the sets $\Delta_u(1+q_t)$. We compare these two sets for $u = t_b$. Letting $c = |I_{t_b}(q_t)|$ we have $c \geq 7$ since q_t is not the last level at step t_b. Since $T_{t_b}(1+q_t)$ and $I_{t_b}(1+q_t)$ are rebuilt, $|T_{t_b}(1+q_t)| \geq \lceil c/2 \rceil$ and $|\Delta_{b_t}(1+q_t)| \geq 2\lfloor(\lceil c/2 \rceil)/3\rfloor \geq c/5$ (where the final inequality uses $c \geq 7$, and is tight for $c = 10$). □

7.6 Prefix Bucketing and Tail Bucketing

In addition to prefix bucketing we have seen in Chapter 6 that was originally defined by Dietz, Seiferas and Zhang [13] we will need several other variants of the bucketing game. In a bucketing game we have k buckets numbered $1, \ldots, k$ in which items are placed. A *bucket configuration* is an arrangement of items in the buckets; formally it is a mapping $C : \{1, \ldots, k\}$ to the nonnegative integers, where $C(i)$ is the number of items in bucket i. It will sometimes be convenient to allow the range of the function C to be the nonnegative real numbers, which corresponds to allowing a bucket to contain a fraction of an item.

A *bucketing game* is a one player game in which the player is given a sequence of groups of items of sizes $n_1, \ldots, n_\ell$ and must sequentially place each group of items into a bucket. The case that $n_1 = \cdots = n_\ell = 1$ is called *simple bucketing*. The placement is done in ℓ steps, and the player selects a sequence $p_1, \ldots, p_\ell \in [1, k]^\ell$, called an (ℓ, k)-*placement sequence* which specifies the bucket into which each group is placed.

Bucketing games vary depending on two ingredients, the *rearrangement rule* and the *cost functions*.

When a group of m items is placed into bucket p, the items in the configuration are rearranged according to a specified rearrangement rule, which is not under the

control of the player. Formally, a rearrangement rule is a function R that takes as input the current configuration C, the number m of new items being placed and the bucket p into which they are placed, and determines a new configuration $R(C, m, p)$ with the same total number of items.

The *prefix rearrangement rule* is as follows: all items currently in buckets below p are moved to bucket p. We say that items are *merged into bucket* p. Formally, the new configuration $C' = R(C, m, p)$ satisfies $C'(i) = 0$ for $i < p$, $C'(p) = C(1) + \cdots + C(p) + m$ and $C'(i) = C(i)$ for $i > p$. Most of the bucketing games we'll discuss use the prefix rearrangement function, but in Section 7.7 we'll need another rearrangement rule.

The *cost function* specifies a cost each time a placement is made. For the cost functions we consider the cost of placing a group depends on the current configuration C and the selected bucket p but not on the number m of items being placed. We consider four cost functions

- In *cheap* bucketing, the cost is the number of items in bucket p before the placement:
$$\mathbf{cost_{cheap}}(C, p) = C(p).$$

- In *expensive* bucketing, the cost is the number of items in buckets p or higher before the placement:
$$\mathbf{cost_{exp}}(C, p) = \sum_{i=p}^{k} C(i).$$

- For $\gamma \in [0, 1]$, in the γ-*discounted* bucketing, the cost is:
$$\mathbf{cost}_{\gamma-\mathbf{disc}}(C, p) = \sum_{i=p}^{k} C(i)\gamma^i.$$
(Note that $\mathbf{cost}_{1-\mathbf{disc}} = \mathbf{cost_{exp}}$.)

- For $b \in \mathbb{N}$, in the b-*block* bucketing, the cost of step t is
$$\mathbf{cost}_{b-\mathbf{block}}(C, p) = \sum_{i=p}^{s(p)} C(i),$$
where $s(p)$ is the least multiple of b larger or equal to p. (Note that $\mathbf{cost}_{1-\mathbf{block}} = \mathbf{cost_{cheap}}$ and $\mathbf{cost}_{k-\mathbf{block}} = \mathbf{cost_{exp}}$.)

For completeness we remark that the cost function used in previous work [13, 2] is the number of items in buckets $1, \ldots, p$ before the placement:

$$\mathbf{cost}(C, p) = \sum_{i=1}^{p} C(i).$$

Fix a rearrangement rule R and a cost function c. A placement sequence $p_1, \ldots, p_\ell$ and a load sequence $n_1, \ldots, n_\ell$ together determine a sequence of configurations $B = (B_0, B_1, \ldots, B_\ell)$, called a *bucketing* where B_0 is the empty configuration and for $i \in [1, \ell]$, $B_i = R(B_{i-1}, n_i, p_i)$. Each of these ℓ placements is charged a cost according to the cost rule c. We write $c[R](p_1, \ldots, p_\ell | n_1, \ldots, n_\ell)$ for the sum $\sum_{i=1}^{\ell} c(B_{i-1}, p_i)$, which is the sum of the costs of each of the ℓ rearrangements that are done during the bucketing. If R is the prefix rule, we call B a *prefix bucketing* and denote the cost simply by $c(p_1, \ldots, p_\ell | n_1, \ldots, n_\ell)$. In the case of simple bucketing, $n_1 = \ldots = n_\ell = 1$, we write simply $c[R](p_1, \ldots, p_\ell)$ or $c(p_1, \ldots, p_\ell)$ in the case of simple prefix bucketing.

7.7 Tail Bucketing and the Online Labeling

We will also need an alternative rearrangement function, called the *tail rearrangement rule*. The bucketing game with this rule is called *tail bucketing*. The tail rearrangement rule $Tail_\beta$ with parameter β acts on configuration C, bucket p and group size m by first moving all items below bucket p to bucket p so that $w = C(1) + \cdots + C(p) + m$ items are in bucket p (as with the prefix rule), but then for j from p down to 1, β fraction of the items in bucket j are passed to bucket $j - 1$, until we reach bucket 1. (Here we allow the number of items in a bucket to be non-integral.) So the number of items in bucket j for $j \in [2, p]$ is $(1 - \beta)\beta^{p-j}w$ and the number of items in bucket 1 is $\beta^{p-1}w$.

A bucketing B produced with the tail bucketing rearrangement rule is called a *tail bucketing*.

We will consider tail bucketing with the cheap cost function. We will now relate the expected cost of randomized online labeling algorithm $\mathcal{A}$ on the sequence $y_1, y_2, \ldots, y_n$ which was produced by our adversary $\mathbf{Adversary}(\mathcal{A}, n, m)$ to the cost of a specific tail bucketing instance.

For a lazy online labeling algorithm $\mathcal{A}$ and $t = 1, \ldots, n$, let $\mathbf{f}_{\mathcal{A},t}, I_t(i), q_t, y_t$ be as defined by the $\mathbf{Adversary}(\mathcal{A}, n, m)$ and the algorithm $\mathcal{A}$. Denote $Y = \{y_1, y_2, \ldots, y_n\}$. Set $k = \lfloor 4 \log(m + 1) \rfloor$. Let $q_1, \ldots, q_n$ be the sequence of critical levels produced by the algorithm. For integer $i \in [k]$ define $\bar{i}$ to be $\bar{i} = (k + 1) - i$. Define the placement sequence $p_1 = \bar{q}_1, \ldots, p_n = \bar{q}_n$, and consider the tail bucketing $B_0, \ldots, B_n$ determined by this placement sequence with parameter $\beta = 1/6$, and

all group sizes 1 (so it is a simple bucketing). The following lemma is used to relate the cost of online labeling to the tail bucketing.

Lemma 7.7.1. *Let $\{I_t(i) : 1 \le t \le n, 1 \le i \le d_t\}$ be the interval chain computed by* **Adversary**$(\mathcal{A}, n, m)$ *and $B_{\mathcal{A}} = (B_0, \ldots, B_n)$ be the corresponding tail-bucketing. Then for any $t \in [0, n]$ and any $j \in [1, d_t]$:*

$$|I_t(j) \setminus I_t(j+1)| \ge B_t(\bar{j}) - 3.$$

Here, for the case $j = d_t$, we take $I_t(j+1)$ to be $\emptyset$.

Proof. We will actually prove:

$$\left\lceil \sum_{i \le \bar{j}} B_t(i) \right\rceil + 2 \ge |I_t(j)| \ge \left\lceil \sum_{i \le \bar{j}} B_t(i) \right\rceil. \tag{7.1}$$

Given this we get:

$$|I_t(j) \setminus I_t(j+1)| \ge \left\lceil \sum_{i \le \bar{j}} B_t(i) \right\rceil - \left(\left\lceil \sum_{i \le \bar{j}-1} B_t(i) \right\rceil + 2 \right) \ge B_t(\bar{j}) - 3,$$

as required.

We prove (7.1) by induction on t. For $t = 0$ we have $d_0 = 1$, so we only need to check the case $j = 1$. We have $|I_0(1)| = 2$, and $\bar{j} = k$ and $\sum_{i \le k} B_0(i) = 0$.

Let $t \ge 1$ and assume the claim is true for $t - 1$. Let $j \in [1, k]$. Suppose first $j \le q_t$. By the definition of the critical level, $q_t \le d_{t-1}$. Therefore $j \le d_{t-1}$ and we may apply the induction hypothesis with $t - 1$ and j. Since $j \le q_t$ $|I_t(j)| = |I_{t-1}(j)| + 1$. The conclusion then follows by induction if we can show that $\sum_{i \le \bar{j}} B_t(i) - \sum_{i \le \bar{j}} B_{t-1}(i) = 1$. This holds because $p_t = \bar{q}_t$ and so $\bar{j} \ge p_t$ and therefore B_t is obtained from B_{t-1} by adding a single item at position p_t and redistributing items among the first p_t buckets, so that the difference in the two sums is indeed 1.

Now assume $j > q_t$. We hold t fixed and prove the equality by induction on j, where we use the already proved case $j = q_t$ as the basis. Suppose that $d_t \ge j > q_t$ and that the desired equality holds for $(t, j-1)$.

Define $w(j) = \sum_{i \le \bar{j}} B_t(i)$. For $d_t \ge j > q_t$ we have $\bar{j} < p_t$ and the tail-bucketing rule implies $w(j) = w(j-1)/6$. Also, the rebuilding rule for $I_t(j)$ implies $|I_t(j)|$ is between $\lceil I_t(j-1)/6 \rceil$ and $\lceil I_t(j-1)/6 \rceil + 1$ (which is verified by case analysis depending on $|I_t(j-1)| \mod 6$).

Thus:

$$\begin{aligned}
|I_t(j)| &\leq \left\lceil \frac{1}{6}|I_t(j-1)| \right\rceil + 1 \\
&\leq \left\lceil \frac{1}{6}(\lceil w(j-1) \rceil + 2) \right\rceil + 1 \\
&\leq \left\lceil \frac{1}{6} w(j-1) \right\rceil + 2 \\
&= \lceil w(j) \rceil + 2,
\end{aligned}$$

where the second inequality uses the induction hypothesis and the third is a simple arithmetic fact. This proves the first inequality of (7.1). Similarly for the second inequality:

$$\begin{aligned}
|I_t(j)| &\geq \left\lceil \frac{1}{6}|I_t(j-1)| \right\rceil \\
&\geq \left\lceil \frac{1}{6}(\lceil w(j-1) \rceil) \right\rceil \\
&\geq \left\lceil \frac{1}{6} w(j-1) \right\rceil \\
&= \lceil w(j) \rceil .
\end{aligned}$$

□

Corollary 7.7.2. *The cost of randomized labeling algorithm $\mathcal{A}$ with label space $[1, m]$ on $y_1, \ldots, y_n$ satisfies:*

$$\chi_{\mathcal{A}}(y_1, y_2, \ldots, y_n) \geq \frac{1}{40}(\min \mathbf{cost}_{\mathbf{cheap}}[Tail_{1/6}](p_1, \ldots, p_n) - 10n),$$

where the minimum is over all placement sequences $(p_1, \ldots, p_n)$ into $\lfloor 4 \log(m+1) \rfloor$ buckets.

Proof. Consider the placement sequence p derived from the sequence of critical levels as in Lemma 7.7.1. The total cost is $\sum_t B_{t-1}(p_t) = \sum_t B_{t-1}(\bar{q}_t)$, which by Lemma 7.7.1 is bounded above by $\sum_t |I_{t-1}(q_t) \setminus I_{t-1}(1+q_t)| + 3n$. Split this latter sum according to $q_t < d_{t-1}$ or $q_t = d_{t-1}$. The terms for which $q_t = d_{t-1}$ are each at most 7 (since $|I_{t-1}(d_{t-1})| \leq 7$) and so:

$$\sum_t B_{t-1}(\bar{q}_t) - 10n \leq \sum_{t : q_t < d_{t-1}} |I_{t-1}(q_t) \setminus I_{t-1}(1+q_t)|.$$

Now apply Lemma 7.5.3. □

7.8 Lower Bounds on Tail Bucketing

Armed with Corollary 7.7.2, it now suffices to prove a lower bound on the cheap cost of simple tail bucketing when the number of items is n and the number of buckets is $\lfloor 4\log(m+1)\rfloor$.[2]

The first step is to bound the cost of (simple) tail bucketing by the cost of (simple) prefix bucketing under the cost function $\mathbf{cost}_{\gamma-\mathbf{disc}}$.

Lemma 7.8.1. *Let $k \geq 1$ be an integer and $p_1, \ldots, p_\ell$ be the placement sequence into k buckets. Then:*

$$\mathbf{cost}_{\mathbf{cheap}}[Tail_\beta](p_1, \ldots, p_\ell) \geq (1-\beta) \cdot \mathbf{cost}_{\beta-\mathbf{disc}}(p_1, \ldots, p_\ell)..$$

Proof. Refer to the item loaded in step j as item j. We can partition the cost of step s as the sum of the contributions due to each of the items $1, \ldots, s-1$. We now show that for each item j and each step $s > j$, the contribution of item j to the cost at step s using $\mathbf{cost}_{\mathbf{cheap}}$ with the tail rearrangement rule is at least $1-\beta$ times the contribution of item j to the cost at step s using $\mathbf{cost}_{\beta-\mathbf{disc}}$.

Let h be an index in $\{j, j+1, \ldots, s-1\}$ such that p_h is maximum. After step $s-1$, under the prefix rearrangement rule, j is located in bucket p_h. If $p_s \leq p_h$ then the contribution to $\mathbf{cost}_{\beta-\mathbf{disc}}$ by item j is $\beta^{p_h-p_s}$, otherwise the contribution is 0.

Under tail rearrangement j is split among buckets $1, \ldots, p_h$ with $(1-\beta)\beta^{p_h-i}$ of j in bucket i for $2 \leq i \leq p_h$ and β^{p_h-1} located in bucket 1. If $p_s > p_h$ then under $\mathbf{cost}_{\mathbf{cheap}}$ the contribution of item j to step s is 0. If $1 < p_s \leq p_h$ then under $\mathbf{cost}_{\mathbf{cheap}}$ the contribution is $(1-\beta)\beta^{p_h-p_s}$ and for $p_s = 1$ the contribution is $\beta^{p_h-p_s}$. This is at least $1-\beta$ times the contribution to $\mathbf{cost}_{\beta-\mathbf{disc}}$ under prefix bucketing. □

The next step is an easy reduction from $\mathbf{cost}_{\gamma-\mathbf{disc}}$ to $\mathbf{cost}_{b-\mathbf{block}}$.

Lemma 7.8.2. *Let $\gamma \in (0,1]$ and $1 \leq b$. Let $p_1, \ldots, p_\ell$ be a placement sequence. Then:*

$$\mathbf{cost}_{\gamma-\mathbf{disc}}(p_1, \ldots, p_\ell) \geq \gamma^b \mathbf{cost}_{b-\mathbf{block}}(p_1, \ldots, p_\ell).$$

Proof. Since in both games we are using the prefix rearrangement rule, the configuration after each step in the two games is the same. Consider the contribution of the tth step of the bucketing to each side. Items are loaded into bucket p_t. Let

[2] *Fun fact:* Before deriving the lower bound we expended several CPU-days (on AMD Phenom II X4 955 3.2GHz with 16GB of RAM) to calculate the optimal cost of tail-bucketing for upto 30 buckets and 500 items. This provided us with confidence that the cost grows in non-linear fashion.

s be the least multiple of b with $s \geq p_t$ and let $r = s - p_t$. In b-block bucketing we pay only for items that at step $t-1$ were in buckets of the form $p_t + i$ where $0 \leq i \leq r$. Since $r \leq b$, in γ-discounted bucketing we pay at least γ^b for each of these items. □

Applying this lemma with $b = 1$ gives $\mathbf{cost}_{\gamma-\mathbf{disc}}(p_1, \ldots, p_n) \geq \mathbf{cost}_{\mathbf{cheap}}(p_1, \ldots, p_n)$. This lower bound does not help us directly because it can be shown that for $k = \log(n+1)$ buckets there is an (n,k)-placement sequence with $\mathbf{cost}_{\mathbf{cheap}}(p_1, \ldots, p_n) = 0$. This follows from the following lemma, which we state in greater generality so that we can use it later:

Lemma 7.8.3. *For any ℓ, k and for any load sequence $n_1, \ldots, n_\ell$ there is an (ℓ, k)-placement sequence $r_1, \ldots, r_\ell$ into k buckets satisfying:*

$$\mathbf{cost}_{\mathbf{cheap}}(r_1, \ldots, r_\ell | n_1, \ldots, n_\ell) = \sum_{j=1}^{\ell - 2^k + 1} n_j(m + 1 - j),$$

where $m = \max(\ell - 2^k + 1, 0)$.

In particular, if $k \geq \log(\ell + 1)$ then $\mathbf{cost}_{\mathbf{cheap}}(r_1, \ldots, r_\ell | n_1, \ldots, n_\ell) = 0$.

Proof. The sequence consists of loading all items into bucket 1 for the first m steps. For all steps $m + j$ for $j \leq 2^k - 1$ load new items in step j in bucket $\alpha(j) + 1$ where $\alpha(j)$ is the largest power of 2 dividing j.

It is easy to prove by induction on j that after step $m + j$ the set of occupied buckets are exactly those whose positions correspond to the 1's in the binary expansion of j. Furthermore, for all $j \geq 2$, $\alpha(j) + 1$ is empty at the end of step $j - 1$. It follows that during the last $2^k - 2$ steps there is no cost incurred.

It remains to bound the total cost during the first $m + 1$ steps. Each item loaded at step $j \leq m$ is charged $m + 1 - j$ steps (at each step in $j + 1, \ldots, m + 1$). Thus the total charge is $\sum_{j=1}^{m+1} n_j(m + 1 - j)$. □

As mentioned, this gives an upper bound of 0 if the number of buckets is at least $\log(\ell + 1)$. We now show that a small reduction in the number of buckets is enough to give a good lower bound on $\mathbf{cost}_{\mathbf{cheap}}$.

Lemma 7.8.4. *For any (ℓ, k)-placement sequence $p_1, \ldots, p_\ell$,*

$$\mathbf{cost}_{\mathbf{cheap}}(p_1, \ldots, p_\ell) \geq (\ell + 1)(\log(\ell + 1) - 2k).$$

Proof. We lower bound $\mathbf{cost}_{\mathbf{cheap}}(p_1, \ldots, p_\ell)$ by induction on ℓ, where the base case $\ell = 0$ is trivial. Let $m_1 < m_2 < .... < m_r$ be the indices such that $p_{m_i} = k$. Also define $m_0 = 0$ and $m_{r+1} = \ell + 1$. For $i \in [1, r + 1]$, the interval $[m_{i-1} + 1, m_i - 1]$ is called *phase i*. Each phase consists only of placements to buckets $k - 1$ or

lower and (except possibly the last phase) is followed immediately by a placement to bucket k. We define $\ell_i = m_i - m_{i-1} - 1$ to be the length of the phase. Let $\gamma_i = (\ell_i + 1)/(\ell + 1)$ so that $\sum_{i=1}^{r+1} \gamma_i = 1$.

Let us now analyze the cost of the sequence phase by phase. At the beginning of phase i there are no items in any bucket below k. The phase itself is an $(\ell_i, k-1)$ bucketing so by induction has cost at least $(\ell_i + 1)(\log(\ell_i + 1) - 2(k-1)) = (\ell + 1)\gamma_i(2 + \log(\gamma_i) + \log(\ell + 1) - 2k)$. Except for $i = r + 1$, the placement p_{m_i} immediately following the phase costs $m_{i-1} = (\ell + 1)(\sum_{j=1}^{i-1} \gamma_j)$ since that is the number of items in bucket k prior to that placement. Summing over phases and rearranging gives:

$$\begin{aligned}\mathbf{cost_{cheap}}(p_1, \ldots, p_\ell) \quad \geq \quad & (\ell + 1)\left(\sum_{j=1}^{r}(r-j)\gamma_j + 2 + \sum_{i=1}^{r+1}\gamma_i \log(\gamma_i)\right) \\ & +(\ell + 1)(\log(\ell + 1) - 2k)\end{aligned}$$

Note that the final term is the lower bound we are aiming for so it suffices to show:

$$\sum_{j=1}^{r}(r-j)\gamma_j + 2 \geq \sum_{i=1}^{r+1}\gamma_i \log(1/\gamma_i).$$

Since $\sum_{i=1}^{r+1} \gamma_i = 1$ the lefthand side is at least $\sum_{j=1}^{r+1}(r-j+2)\gamma_j$. Observing that $\sum_{i=1}^{r+1} 2^{-(r-j+2)} \leq 1$, the desired inequality follows from:

Proposition 7.8.5. *Let $\alpha_1, \ldots, \alpha_s$ be nonnegative reals summing to 1. Then for all choices of $x_1, \ldots, x_s$ of nonnegative reals with sum at most 1, the function $\sum_i \alpha_i \log(1/x_i)$ is minimized when $(x_1, \ldots, x_s) = (\alpha_1, \ldots, \alpha_s)$.*

This is essentially equivalent to the well known fact that the KL-divergence of two distributions is always nonnegative and is easily proved by first noting that we may assume $\sum_i x_i = 1$, and then using Lagrange multipliers, or induction on s. □

7.9 From $\mathbf{cost}_{b-\mathbf{block}}$ to $\mathbf{cost}_{\mathbf{cheap}}$

So far we have shown that the cost of online labeling can be bounded below by the cheap cost of tail-bucketing, which can be bounded below by the $\mathbf{cost}_{b-\mathbf{block}}$ for simple bucketing.

Below we will prove Lemma 7.9.3 which shows that $\mathbf{cost}_{b-\mathbf{block}}$ can be bounded below by $\mathbf{cost}_{\mathbf{cheap}}$ with fewer buckets. In preparation, we begin by bounding $\mathbf{cost}_{\mathbf{exp}}$ from below by $\mathbf{cost}_{\mathbf{cheap}}$ with fewer buckets.

Lemma 7.9.1. *Let $k \geq 1$ and $b = 2^k - 1$. Let $n_1, \ldots, n_\ell$ be an arbitrary load sequence. Then for any placement sequence $p_1, \ldots, p_\ell$ into b buckets there is a placement sequence $r_1, \ldots, r_\ell$ into k buckets such that*

$$\mathbf{cost}_{\mathbf{cheap}}(r_1, \ldots, r_\ell | n_1, \ldots, n_\ell) \leq \mathbf{cost}_{\mathbf{exp}}(p_1, \ldots, p_\ell | n_1, \ldots, n_\ell).$$

Proof. If $\ell \leq b$ then $k \geq \log(\ell+1)$ so by Lemma 7.8.3 there is a placement sequence $r_1, \ldots, r_\ell$ with zero cheap cost and the lemma follows. Hence, assume $\ell > b$. We begin with a lower bound on $\mathbf{cost}_{\mathbf{exp}}(p_1, \ldots, p_\ell | n_1, \ldots, n_\ell)$. At step j, any item inserted before j that is in bucket p_j or higher incurs a charge of 1. Any previously loaded item that is in a bucket less than p_j incurs no charge, but is moved to bucket p_j. Thus, once an item is loaded, in every step it incurs a charge of 1 or increases its bucket number. An item loaded at step j incurs no cost at step j and incurs a cost of 1 in every step that it does not move, which means that it incurs a cost of one in at least $(\ell - j) - (b - 1)$ steps. Summing over the first $\ell - b$ items we get.

$$\mathbf{cost}_{\mathbf{exp}}(p_1, \ldots, p_\ell | n_1, \ldots, n_\ell) \geq \sum_{j=1}^{\ell-b} n_j(\ell - j - b + 1).$$

Now, setting $b = 2^k - 1$, Lemma 7.8.3 completes the proof of the lemma. □

For a step i let $c^i(p_1, \ldots, p_\ell | n_1, \ldots, n_\ell)$ be the cost of the placement into p_i at step i. For $I \subseteq [1, \ell]$, let

$$c^I(p_1, \ldots, p_\ell | n_1, \ldots, n_\ell) = \sum_{i \in I} c^i(p_1, \ldots, p_\ell | n_1, \ldots, n_\ell). \tag{7.2}$$

Lemma 7.9.2. *Let $p_1, \ldots, p_\ell$ be a placement sequence with b buckets. Let $\theta \in [1, b]$ and let $I = \{i_1 < \cdots < i_h\}$ be the indices in $[1, \ell]$ such that $p_{i_j} > \theta$. Let $s_1, \ldots, s_h$ be the placement sequence into $b - \theta$ buckets given by $s_j = p_{i_j} - \theta$ and let $n_1, \ldots, n_h$ be given by $n_1 = i_1$ and for $j > 1$, $n_j = i_j - i_{j-1}$. Then for cost function $c \in \{\mathbf{cost}_{\mathbf{cheap}}, \mathbf{cost}_{\mathbf{exp}}\}$,*

$$c^I(p_1, \ldots, p_\ell) = c(s_1, \ldots, s_h | n_1, \ldots, n_h).$$

Proof. It suffices to show that for each $j \in [1, h]$, $c^{i_j}(p_1, \ldots, p_\ell) = c^j(s_1, \ldots, s_h | n_1, \ldots, n_h)$. Let $B_1, \ldots, B_\ell$ be the bucketing sequence associated to $(p_1, \ldots, p_\ell)$, and let $\tilde{B}_1, \ldots, \tilde{B}_h$ be the bucketing sequence associated to $(s_1, \ldots, s_h | n_1, \ldots, n_h)$.

We claim that for each $j \in [1,h]$ the configuration B_{i_j} restricted to $[\theta+1,b]$ is identical to the configuration $\tilde{B}_j$ restricted to $[1,b-\theta]$. This is easily shown by induction on j. The base case $j=0$ is trivial. Assume $j>0$. The result holds for $j-1$ so $B_{i_{j-1}}$ restricted to $[\theta+1,b]$ is identical to B_{j-1} restricted to $[1,b-\theta]$.

For the sequence $s_1,\ldots,s_h$, at step j, all buckets above s_j are unchanged, all buckets below s_j are emptied, and s_j increases by the number of items that were in buckets below s_j, together with the load of n_j.

Now consider the change in the configuration B from $B_{i_{j-1}}$ to B_{i_j}. For each $s \in i_{j-1}+1$ to i_j-1, $p_s \leq \theta$, which implies that B restricted to $[\theta+1,b]$ is unchanged. Next consider the placement p_{j_i} at step j_i. All buckets above $p_{j_i} = s_j+\theta$ are unchanged and all buckets below p_j are emptied, and bucket p_j gets all of the items that were in buckets $[\theta+1,p_{i_j}-1]$ after step i_{j-1} together with all of the n_j new items that arrived since i_{j-1} of the buckets in B. This exactly matches the change in bucket s_j at step j in the other bucketing, as required to establish the claim.

By the claim, the cost of step i_j for $p_1,\ldots,p_\ell$ is the same as the cost of step j for $s_1,\ldots,s_h|n_1,\ldots,n_h$ as required to prove the lemma. □

Next we come to a crucial reduction which lower bounds $\mathbf{cost}_{b-\mathbf{block}}$ in terms of $\mathbf{cost}_{\mathbf{cheap}}$ with a fewer number of buckets.

Lemma 7.9.3. *Let $k \geq 1$, $m \geq 1$ and $b = 2^k-1$. Let $p_1,\ldots,p_\ell$ be a placement sequence into bm buckets. There exists a placement sequence $s_1,\ldots,s_\ell$ for km buckets such that*

$$\mathbf{cost}_{\mathbf{cheap}}(s_1,\ldots,s_\ell) \leq \mathbf{cost}_{b-\mathbf{block}}(p_1,\ldots,p_\ell).$$

Proof. Fix $p_1,\ldots,p_\ell$. We first describe the construction of the sequence $s_1,\ldots,s_\ell$ and then prove the properties.

To specify the sequence $s_1,\ldots,s_\ell$ we will define a partition of $[1,\ell]$ into (generally non-consecutive) subsequences, and for each set $\widehat{\mathbf{h}}$ in the partition separately specify s_i for $i \in \widehat{\mathbf{h}}$.

The definition of the partition takes a few steps. Define the *level of a bucket w for block size b* to be the largest λ such that $\lambda b < w$, and the *remainder of w* to be $w-\lambda b$. For $i \in [1,n]$, define λ_i to be the level of p_i and r_i to be the remainder of p_i. By the hypotheses of the lemma each $\lambda_i \in [0,m-1]$ and each remainder is in $[1,\ldots,b]$. We also define $\lambda_0 = \lambda_{\ell+1} = \infty$.

A chain of level j and order v is a sequence $\mathbf{h}$ of indices $\mathbf{h}_0 < \mathbf{h}_1 < \cdots < \mathbf{h}_v < \mathbf{h}_{v+1}$ (with possibly $\mathbf{h}_0 = 0$ or $\mathbf{h}_{v+1} = \ell+1$) satisfying the following properties:

- $\lambda_{\mathbf{h}_0} > j$ and $\lambda_{\mathbf{h}_{v+1}} > j$,

- $\lambda_{\mathbf{h}_1} = \cdots = \lambda_{\mathbf{h}_v} = j$,

- For any index i belonging to $[\mathbf{h}_0, \mathbf{h}_{v+1}] \setminus \{\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{v+1}\}$, $\lambda_i < j$.

The indices $\mathbf{h}_0, \mathbf{h}_{v+1}$ are the *endpoints* of $\mathbf{h}$, and the other indices are the *interior indices*. We write $\widehat{\mathbf{h}} = \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_v\}$ for the interior of $\mathbf{h}$. Thus the order of $\mathbf{h}$ equals to $|\widehat{\mathbf{h}}|$. A chain of order 0 is *trivial*, others are non-trivial. Every chain of level j can be obtained in the following way: consider the sequence $0 = g_0 < g_1 < \cdots < g_{w-1} < g_w = \ell + 1$ consisting of all indices at level higher than j. Then between each consecutive pair g_i and g_{i+1} from the sequence there is a unique chain of level j. The interiors of these chains partition the set of indices at level j. The collection of all nontrivial chains is denoted $\mathcal{H}$, and the set of interiors of these chains partitions $[1, \ell]$.

We write $\lambda(\mathbf{h})$ for the level of $\mathbf{h}$ and $v(\mathbf{h})$ for the order of $\mathbf{h}$.

For a chain $\mathbf{h}$ as above, we define its *difference sequence* to be the sequence $\Delta^{\mathbf{h}}_1, \ldots, \Delta^{\mathbf{h}}_{v(\mathbf{h})}$ given by $\Delta^{\mathbf{h}}_i = \mathbf{h}_i - \mathbf{h}_{i-1}$. (We could also define $\Delta^{\mathbf{h}}_{v(\mathbf{h})+1}$ but we won't need it.) The sum of the difference sequence is just $\mathbf{h}_{v(\mathbf{h})} - \mathbf{h}_0$. Finally we define the remainder sequence of $\mathbf{h}$ to be the subsequence $r_{\mathbf{h}_1}, \ldots, r_{\mathbf{h}_{v(\mathbf{h})}}$ of the remainder sequence $r_1, \ldots, r_{v(\mathbf{h})}$ corresponding to the interior indices of $\mathbf{h}$.

At last we are ready to define s_i. Fix $\mathbf{h} \in \mathcal{H}$; we define s_i for $i \in \widehat{\mathbf{h}}$. Now view the remainder sequence $r_{\mathbf{h}_1}, \ldots, r_{\mathbf{h}_v}$ of $\mathbf{h}$ as a placement for the load sequence $\Delta^{\mathbf{h}}_1, \ldots, \Delta^{\mathbf{h}}_{v(\mathbf{h})}$. All of these placements are in the range $[1, 2^k - 1]$ so by Lemma 7.9.1, there is a placement $u_1, \ldots, u_{v(\mathbf{h})}$ into buckets in the range $[1, k]$ such that:

$$\mathbf{cost}_{\mathbf{cheap}}(u_1, \ldots, u_{v(\mathbf{h})} | \Delta^{\mathbf{h}}_1, \ldots, \Delta^{\mathbf{h}}_{v(\mathbf{h})}) \leq \mathbf{cost}_{\mathbf{exp}}(r_1, \ldots, r_{v(\mathbf{h})} | \Delta^{\mathbf{h}}_1, \ldots, \Delta^{\mathbf{h}}_{v(\mathbf{h})}).$$

Now for each $i \in [1, v(\mathbf{h})]$ let $s_{\mathbf{h}_i} = \lambda(\mathbf{h})k + u_i$. This defines the values s_i for $i \in \widehat{\mathbf{h}}$, and by doing this for all $\mathbf{h} \in \mathcal{H}$ we get the sequence $s_1, \ldots, s_n$.

Since $\lambda(\mathbf{h}) \in [0, m-1]$ and $u_i \in [1, k]$ we have that all s values are in $[1, km]$. When we refer to the level of an s_i we mean its level with respect to block size k. Observe that the sequence λ_i of levels (with respect to block size b) corresponding to the placement sequence p is the same as the sequence of levels (with respect to block size k) corresponding to placements in s.

To prove the inequality of the lemma, we need a bit more notation. Write $p^{\mathbf{h}}$ for the consecutive subsequence of p of length $\mathbf{h}_v - \mathbf{h}_0$ starting with $p_{\mathbf{h}_0+1}$. Thus $p^{\mathbf{h}}_i = p_{\mathbf{h}_0+i}$. Define $s^{\mathbf{h}}$ analogously. Also let $\widehat{\mathbf{h}}_{\mathrm{Rel}} = \{\mathbf{h}_1 - \mathbf{h}_0, \mathbf{h}_2 - \mathbf{h}_0, \ldots, \mathbf{h}_{v(\mathbf{h})} - \mathbf{h}_0\}$. Thus $\widehat{\mathbf{h}}_{\mathrm{Rel}}$ is the set of indices of $p^{\mathbf{h}}$ corresponding to $\widehat{\mathbf{h}}$.

The inequality of the lemma is obtained from the following chain (where we use the notation from (7.2)) of relations:

$$
\begin{aligned}
\mathbf{cost}_{b-\mathbf{block}}(p_1,\ldots,p_\ell) &\overset{(A1)}{=} \sum_{\mathbf{h}\in\mathcal{H}} \mathbf{cost}^{\widehat{\mathbf{h}}}_{b-\mathbf{block}}(p_1,\ldots,p_\ell)\\
&\overset{(A2)}{=} \sum_{\mathbf{h}\in\mathcal{H}} \mathbf{cost}^{\widehat{\mathbf{h}}_{\mathrm{Rel}}}_{\mathbf{exp}}(p^{\mathbf{h}}_1,\ldots,p^{\mathbf{h}}_{\mathbf{h}_{v(\mathbf{h})}-\mathbf{h}_0})\\
&\overset{(A3)}{=} \sum_{\mathbf{h}\in\mathcal{H}} \mathbf{cost}_{\mathbf{exp}}(r_{\mathbf{h}_1},\ldots,r_{\mathbf{h}_{v(\mathbf{h})}}|\Delta^{\mathbf{h}}_1,\ldots,\Delta^{\mathbf{h}}_{v(\mathbf{h})})\\
&\overset{(A4)}{\geq} \sum_{\mathbf{h}\in\mathcal{H}} \mathbf{cost}_{\mathbf{cheap}}(u_{\mathbf{h}_1},\ldots,u_{\mathbf{h}_{v(\mathbf{h})}}|\Delta^{\mathbf{h}}_1,\ldots,\Delta^{\mathbf{h}}_{v(\mathbf{h})})\\
&\overset{(A5)}{=} \sum_{\mathbf{h}\in\mathcal{H}} \mathbf{cost}^{\widehat{\mathbf{h}}_{\mathrm{Rel}}}_{\mathbf{cheap}}(s^{\mathbf{h}}_1,\ldots,s^{\mathbf{h}}_{\mathbf{h}_{v(\mathbf{h})}-\mathbf{h}_0})\\
&\overset{(A6)}{=} \sum_{\mathbf{h}\in\mathcal{H}} \mathbf{cost}^{\widehat{\mathbf{h}}}_{\mathbf{cheap}}(s_1,\ldots,s_\ell)\\
&\overset{(A7)}{=} \mathbf{cost}_{\mathbf{cheap}}(s_1,\ldots,s_\ell).
\end{aligned}
$$

We now justify each of these steps. We work from both ends to the middle. Equalities (A1) and (A7) follow from the fact that $\mathcal{H}$ is a partition of $[1,\ell]$. For all of the other relations, we fix an $\mathbf{h}\in\mathcal{H}$ and show it holds term by term. For (A2) observe first that after step $\mathbf{h}_0$, items stored during steps $[1,\mathbf{h}_0]$ are in buckets higher than level j and so contribute nothing to the $\mathbf{cost}_{b-\mathbf{block}}$ during steps $[\mathbf{h}_0+1,\mathbf{h}_{v(\mathbf{h})}]$ so in accounting for the cost of steps of $\widehat{\mathbf{h}}$ we can omit all placements prior to $\mathbf{h}_0$. During $[\mathbf{h}_0+1,\mathbf{h}_{v(\mathbf{h})}]$ there are no placements above block j so $\mathbf{cost}_{\mathbf{exp}}$ coincides with $\mathbf{cost}_{b-\mathbf{block}}$. This proves (A2) and a similar argument gives (A6). For (A3), we apply Lemma 7.9.2 with $\theta=jb$, and for (A5) we apply the same lemma with $\theta=jk$. Finally Lemma 7.9.1 implies (A4). □

Lemma 7.9.4. *Let $c\geq 1$ be an arbitrary constant. For any large enough integers n,m satisfying $m<(n+1)^c$ and any placement sequence $p_1,\ldots,p_n$ into $\lfloor 4\log(m+1)\rfloor$ buckets the following is true:*

$$
\mathbf{cost}_{\mathbf{cheap}}[Tail_{1/6}](p_1,\ldots,p_n)\geq \frac{5}{12}\left(\frac{1}{6}\right)^{512c^2}(n+1)\log(n+1).
$$

Now Lemma 7.5.1 follows from this lemma and Corollary 7.7.2.

Proof of Lemma 7.9.4. Choose $b\in[256c^2,512c^2]$ such that $b=2^k-1$ for some integer k. By Lemmas 7.8.1 and 7.8.2 we have:

$$
\begin{aligned}
\mathbf{cost}_{\mathbf{cheap}}[Tail_{1/6}](p_1,\ldots,p_n) &\geq (5/6)\cdot\mathbf{cost}_{1/6-\mathbf{disc}}(p_1,\ldots,p_n)\\
&\geq (5/6)(1/6)^b\cdot\mathbf{cost}_{b-\mathbf{block}}(p_1,\ldots,p_n).
\end{aligned}
$$

Set $k_1 = \lfloor 4\log(m+1) \rfloor$ and $k_2 = k \cdot \lceil k_1/b \rceil$. By Lemma 7.9.3, for any (n, k_1)-placement sequence $p_1, \ldots, p_n$ there is a (n, k_2)-placement sequence $s_1, \ldots s_n$ such that:

$$(5/6)(1/6)^b \cdot \mathbf{cost}_{b-\mathbf{block}}(p_1, \ldots, p_n) \;\; \geq \;\; (5/6)(1/6)^b \cdot \mathbf{cost}_{\mathbf{cheap}}(s_1, \ldots, s_n).$$

We want to apply Lemma 7.8.4 to this final expression. Notice,

$$\begin{aligned} k_2 &= \log(b+1) \cdot \left\lceil \frac{\lfloor 4\log(m+1) \rfloor}{b} \right\rceil \\ &\leq \log(b+1) + \frac{\log(b+1)}{b} \cdot 4c \cdot \log(n+1) \\ &\leq \frac{\log(n+1)}{4} \end{aligned}$$

provided that n is large enough and $\log(b+1)/b < 1/16c$. Indeed, since $\log(x+1)/x$ is a decreasing function, $\log(b+1)/b \leq \log(256c^2+1)/256c^2 \leq (11/16)(1/16c)$ as can be easily verified. Lemma 7.8.4 applied on $s_1, \ldots s_n$ implies the lower bound. □

8. Online Labeling Problem with Small Label Space

8.1 Introduction

In this chapter we prove an $\Omega\left(n \cdot \frac{\log^2(n)}{2+\log(m)-\log(n)}\right)$ lower bound on the number of moves for inserting n items. This lower bound is valid for m between cn ($c > 1$) and $n^{1+\varepsilon}$ ($\varepsilon < \frac{1}{16}$) for any deterministic online labeling algorithm, matching the known upper bound up to constant factors.

This chapter is an extension of [10] obtained together with Martin Babka and Vladimír Čunát. It is based on the original simpler proof of [10]. In Chapter 9 we provide [10] which was the first lower bound for general algorithms in the linear space regime. Previously the only known general lower bound was by Dietz et al. [13] who proved an $\Omega(n \cdot \log(n))$ lower bound for polynomial size arrays. Thus our bound is a significant improvement. For the case of $m = O(n)$, Dietz et al. [12, 21] proved an $\Omega(n \log^2(n))$ lower bound for the restricted case of so-called *smooth algorithms*, however, this didn't give tight bound beyond the linear case.

This chapter uses ideas similar to Chapter 9. Chapter 9 provides lower bounds for the linear case and arrays of size close to n. It also gives lower bounds for the case of limited item universe. It would be possible to extend proofs of Chapter 9 to handle also the case considered in this chapter. However, the main ideas of the proof would disappear. Thus we present the lower bound proof for the superlinear arrays independently. We hope that this presentation makes the proof easier to follow and also helps in understanding of Chapter 9.

Recall we use the notation from section 1.2 and section 5.2.

8.2 Hard Sequence Construction

In this section we sketch an adversary which given an algorithm $\mathcal{A}$, the number of inserted items n and the size of the array m (such that they satisfy some requirements we point out later) it produces an input sequence $y_1, y_2, \ldots, y_n$ that is costly for the algorithm $\mathcal{A}$.

As a guide to picking each successive item, the adversary maintains a sequence (*chain*) of nested intervals of already inserted items similar to previous chapters. There are however differences to chains in Chapters 6 and 7. While in these chapters we build chain from the very first step, here we start after roughly $n/2$ items were inserted. The reason for this is that then we can fix the depth of the chain to be the same during the argument. The first $n/2$ items are therefore chosen

almost arbitrarily, we only have to ensure that between any two inserted items, there remain enough unused items in the universe.

The other difference is that for this proof the density control is crucial. Ideally, we should build the chain so that the density of successive intervals of the chain is nondecreasing (which is not necessary in Chapters 6 and 7). This is easy to achieve if we can choose each successive subinterval arbitrarily. However we have to choose the subinterval so that it has significant buffers of items surrounding it. This allows us to charge the algorithm when it moves the boundaries of some interval of the chain (recall we may assume that algorithm is lazy). Unfortunately when the buffers are introduced it is infeasible to ensure increasing density. Thus we have to relax this condition so that the density does not decrease *too much*. Still the density decrease may be *too large*. Therefore we distinguish *good* and *bad* intervals of the chain (levels of the chain). We say that an interval is *good*, if it contains a large subinterval surrounded with large enough buffers and the density of the subinterval is close to the density of the whole interval. The interval is *bad* otherwise. We show that in the bad case we can find large subinterval whose density is significantly higher then the density of the whole interval. By very careful choice of "almost the same" and "significantly higher" we can build a chain which is suitable for a charging scheme which we describe later. In particular we will be able to limit the number of bad intervals in the chain after inserting each item.

We will need to rebuild the chain after each insert. The basic idea is similar to previous chapters. We first determine so called *critical level* and then starting from this level we *rebuild* the chain while all intervals above the critical level are *preserved* (i.e. they are unchanged, except for the newly inserted item which is added to them). The choice of the critical level is however slightly tricky. Before a good interval is rebuilt we need to ensure that enough items were moved. This is easy to ascertain for intervals whose borders were crossed by the lazy algorithm (i.e. their leftmost or rightmost item was relabeled). However if the borders were not crossed, a good interval could technically become bad by redistributing the items within the interval. Here we crucially use an additional property of good intervals. If the interval does not contain a large enough interval with much higher density, then all subintervals are almost as dense as the whole interval. Using this we can show, that unless a significant number of items which are in buffers of the interval are moved, a subinterval with almost the same density can be found.

8.3 Charging Scheme

Charging scheme defines a way we assign the cost of the algorithm among the inserted items. This scheme is used to lower bound the cost of the algorithm.

First we show that the cost of the algorithm at each step can be lower bound by the number of items in buffers of good intervals which were rebuilt in that step. Notice that we completely ignore the moves of items which occur in bad intervals of the chain, still the remaining cost will be large enough.

Now we focus on each interval that was rebuilt separately. We distribute the cost which equals to the size of buffers of each interval among the last half of items inserted to this interval since the interval was rebuilt the last time. We say that these items were charged at this step. The reason for not charging all of the items will be explained later. This is correct as from construction of the adversary it will be obvious that buffers of different intervals are disjoint. Notice that items may be charged on multiple levels in a single step.

Next we lower bound the total cost assigned to certain items. We choose those items which were charged enough times (i.e., at least on constant factor of all levels). First we estimate the charge assigned to the item at certain level to be (roughly) the fraction of the density of the interval on that level and the density of the interval on the next level. We consider the densities with respect to the step when the item was inserted. This approximates the size of the interval buffers divided by the number of items inserted to the level since the last rebuilt. This is also the reason why we only consider the last half of the items inserted since last rebuilt, because for them we can estimate their charge well.

However the reason we insist on estimating charge using the densities at one particular step is that the way we build interval chain puts some limits on densities of intervals in it at each step (but not between the steps). In particular, the densities of intervals in which one particular item was charged cannot be growing that much. We use this to show that the number of newly inserted items among all the intervals in which the item was charged is limited. Thus the number of items among which the cost is distributed is relatively small and we can find sufficient lower bound on cost charged to the items.

Finally we show, that the number of items which are charged on sufficient number of levels is at least constant fraction of all items. This follows from the fact that the number of good levels at each step is a constant fraction of depth of the chain.

8.4 The Main Theorem

In this section, we state the main result of this chapter.

Theorem 8.4.1. *There are positive constants C_0, and C_1 so that the following holds. Let $\mathcal{A}$ be a deterministic algorithm with parameters (n, m, r), such that $C_0 \leq n \leq m \leq \frac{1}{4}n^{1+1/16}$ and $r \geq 2^n - 1$. Then $\chi_{\mathcal{A}}(n, m, r) \geq C_1 \cdot \frac{n \log^2(n)}{2+\log m - \log n}$.*

This theorem will be an immediate consequence of Lemma 8.5.2 in the next section.

8.5 Adversary Construction

In this section we specify the adversary **Adversary**$(\mathcal{A}, n, m)$ which given an online labeling algorithm $\mathcal{A}$, a length n, and label space size m, constructs an item sequence $y_1, y_2, \ldots, y_n$ from the universe $U = \{1, \ldots, 2^n - 1\}$. We state Lemma 8.5.2 which implies Theorem 8.4.1.

In what follows, $\mathbf{f}_t$ is the allocation of Y_t by the algorithm $\mathcal{A}$. As was explained in Section 5.3 the central problem is to construct the sequence of items $y_1, \ldots, y_n$ which cause the algorithm to perform the desired number of relabellings. To do so we want to track the dense segments of the array. Therefore the adversary maintains a sequence of nested intervals of inserted items Y_t,[1]

$$I_t(\mathbf{depth}) \subseteq \cdots \subseteq I_t(2) \subseteq I_t(1) = Y_t \cup \{\min_U, \max_U\},$$

updating them after each time step t.

Our proofs use the following assumptions.

Assumption 8.5.1.

(1) The number of inserted items n is at least 512

(2) The size of the array m is at most $\frac{1}{4}n^{1+1/16}$

Consider an arbitrary interval I of Y_t. The density of interval I at step t, $\rho_t(I)$, is defined as $\rho_t(I) = \frac{|I|}{\mathbf{span}_t(I)+1}$. Notice that $\rho_t(I) \leq 1$. Let $\mathbf{densify}_t(I)$ be the subinterval T of I such that

1. $\lceil \frac{I}{16} \rceil \leq |T| \leq \lceil \frac{I}{8} \rceil$
2. $\rho_t(T)$ is maximal among all possible T's.

If there are more such subintervals T we choose the one with the minimal smallest item. Hence, $\mathbf{densify}_t(I)$ is the densest subinterval T of I that contains the appropriate number of items.

Figure 8.1 presents the pseudocode of the **Adversary**$(\mathcal{A}, n, m)$.

[1]Recall, we use subscript to denote step and we use $(\cdot)$ notation to denote a particular coordinate of such a vector or sequence at that step.

Adversary$(\mathcal{A}, n, m)$

- $\mathbf{depth} \longleftarrow \lfloor \log_{16} \frac{n}{2} \rfloor$, $\boldsymbol{\delta} \longleftarrow \left(\frac{4 \cdot m}{n}\right)^{\frac{2}{\mathbf{depth}}}$, $\mathbf{t_0} \longleftarrow \left\lfloor \frac{n}{2} \right\rfloor$
- $I_{\mathbf{t_0}-1}(1) \longleftarrow \{y_t = 2^{n-t}; t \in \{1, \ldots, \mathbf{t_0} - 1\}\} \cup \{\min_U, \max_U\}$
- For $t = \mathbf{t_0}, \ldots, n$ do
  - if $t = \mathbf{t_0}$ then $y_t \longleftarrow 2^{n-\mathbf{t_0}}$
  - else $y_t \longleftarrow \min(I_{t-1}(\mathbf{depth})) + 2^{n-t}$
  - Run $\mathcal{A}$ on $(y_1, y_2, \ldots, y_t)$ to set $\mathbf{f}_t$ and $\mathbf{Rel}_t$.

  {***Choose Critical Level***}
  - if $t \neq \mathbf{t_0}$
    - $q_t \longleftarrow$ maximal i such that $\min(I_{t-1}(i)), \max(I_{t-1}(i)) \notin \mathbf{Rel}_t$
    - If $\mathbf{bad}_{\mathrm{t}-1}(\mathrm{q_t})$ is *true* or $((L_{t-1}(q_t) \cup R_{t-1}(q_t)) \cap \mathbf{Rel}_t \neq \emptyset$ then
      - $q_t \longleftarrow q_t - 1$
  - else $q_t \longleftarrow 0$

  {***Preservation Rule***}
  - for $i = 1, \ldots, q_t$ do:
    - $I_t(i) \longleftarrow I_{t-1}(i) \cup \{y_t\}$
    - $L_t(i) \longleftarrow L_{t-1}(i)$, $R_t(i) \longleftarrow R_{t-1}(i)$, $\mathbf{bad}_{\mathrm{t}}(\mathrm{i}) \longleftarrow \mathbf{bad}_{\mathrm{t}-1}(\mathrm{i})$

  {***Rebuilding Rule***}
  - $I_t(q_t + 1) \longleftarrow I_{t-1}(q_t + 1) \cup \mathbf{Rel}_t$
  - for $i = q_t + 1, \ldots, \mathbf{depth} - 1$ do:
    - $L_t(i) \longleftarrow \left\lceil \frac{I_{t-1}(i)}{16} \right\rceil$ smallest items of $I_{t-1}(i)$
    - $R_t(i) \longleftarrow \left\lceil \frac{I_{t-1}(i)}{16} \right\rceil$ greatest items of $I_{t-1}(i)$
    - if $\rho_t(\mathbf{densify}_t(I_t(i))) > \boldsymbol{\delta}\rho_t(I_t(i))$ then
      - $\mathbf{bad}_{\mathrm{t}}(\mathrm{i}) \longleftarrow true$
      - $I_t(i+1) \longleftarrow \mathbf{densify}_t(I_t(i))$
    - else
      - $\mathbf{bad}_{\mathrm{t}}(\mathrm{i}) \longleftarrow false$
      - $I_t(i+1) \longleftarrow I_t(i) \backslash \left\{ \left\lceil \frac{I_{t-1}(i)}{8} \right\rceil \text{ smallest and greatest items from } I_t(i) \right\}$
  - $L_t(\mathbf{depth}) \longleftarrow R_t(\mathbf{depth}) \longleftarrow \emptyset$
  - $\mathbf{bad}_{\mathrm{t}}(\mathbf{depth}) \longleftarrow true$
- *Output:* $y_1, y_2, \ldots, y_n$.

Figure 8.1: Pseudocode for the Adversary

Let us explain some details of the adversary. As already mentioned the adversary generates the sequence of items $y_1, y_2, \ldots, y_n$ using the interval chain at each step to determine the next item.

The first $\mathbf{t_0} = \left\lfloor \frac{n}{2} \right\rfloor$ items are chosen almost arbitrarily since these items are only needed for the initial construction of the chain. We only have to ensure that the minimum difference between two inserted items is at least $2^{n-\mathbf{t_0}}$. This is necessary since a crucial assumption for the adversary is that $y_1, \ldots, y_n$ are distinct. It is easy to see by induction on t that the items belonging to $Y_t \cup \{\min_U, \max_U\}$ are multiples of 2^{n-t}, and it follows that y_t is strictly between the smallest and second smallest elements of $I_{t-1}(\mathbf{depth})$.

Now we need to specify how we determine the chain at step $t \geq \mathbf{t_0}$.

We build the intervals for the chain at step t in order of increasing level (i.e., decreasing size). Intervals are either *preserved* (carried over from the previous chain, with the addition of y_t) or *rebuilt*. To specify which intervals are preserved, we define a *critical level q_t for step t*, which is at least 0 and at most **depth** which is the length of the chain at each step t, $t \geq \mathbf{t_0}$. We'll explain the choice of q_t below. At step t, the intervals $I_t(i)$ for $i \leq q_t$ are *preserved*, which means that $I_t(i)$ is obtained simply by adding y_t to $I_{t-1}(i)$, and the rest are *rebuilt*. The rule for rebuilding the chain for $i > q_t$ is defined by induction on i as follows: If subinterval $\mathbf{densify}_t(I_t(i-1))$ of $I_t(i-1)$ has density at least δ times greater than the density of $I_t(i-1)$ we say that $I_t(i-1)$ is bad and we set $I_t(i)$ to be a $\mathbf{densify}_t(I_t(i-1))$. Otherwise, we say that $I_t(i-1)$ is good and we choose middle half of items of $I_t(i-1)$ to be $I_t(i)$. If $I_t(i-1)$ is good we also define $L_t(i-1)$ and $R_t(i-1)$ which play the role of buffers and are unchanged until i-th level is rebuilt again. On the other hand, the first line of Rebuilding Rule allows the subinterval $I_t(i)$ on i-th level to absorb additional items of $I_t(i-1)$ during next steps even if $I_t(i-1)$ is not rebuilt (i.e., items from subinterval between $L_t(i-1)$ and $I_t(i)$ and between $I_t(i)$ and $R_t(i-1)$). In particular it absorbs relocated elements. In this way, $I_t(i)$ may absorb number of items comparable to the size of $L_t(i-1)$ and $R_t(i-1)$. Once it absorbs enough elements, we rebuild $I_t(i)$.

It remains to explain how the critical level q_t is selected. First we find the greatest level ℓ such that the smallest and the greatest items of $I_t(\ell)$ were not moved. If $I_t(\ell)$ is good and its buffers were not affected we set $q_t = \ell$. Notice, that this is the only case when the first line of Rebuilding Rule may have non-trivial impact since $\mathbf{Rel}_t$ does not have to be a subset of $I_t(\ell+1) \cup \{y_t\}$.

If $I_t(\ell)$ is good and its buffers were affected we set $q_t = \ell - 1$ since we have to rebuild $I_t(\ell)$. The reason is that enough work has been done and we cannot ensure the requested minimal density of $I_t(\ell+1)$ after the rebuild. Similarly, if $I_t(\ell)$ is bad we set $q_t = \ell - 1$ since its internal structure changed and it may become good.

Now we can state the main lemma of the section which together with Lem-

ma 5.4.2 (we can consider only lazy algorithms) implies Theorem 8.4.1 immediately.

Lemma 8.5.2. *Under Assumptions 8.5.1 let $\mathcal{A}$ be a lazy online labeling algorithm with parameters (n, m). Let $y_1, y_2, \ldots, y_n$ be the output of* **Adversary**$(\mathcal{A}, n, m)$ *(Figure 8.1). Then it holds that*

$$\chi_{\mathcal{A}}\left((y_1, y_2, \ldots, y_n), m\right) \geq \frac{n}{32} \cdot \left(\frac{\lfloor \log_{16}(\frac{n}{2}) \rfloor^2}{2^{52} \cdot (\log m - \log n + 2)} - \frac{\lfloor \log_{16}(\frac{n}{2}) \rfloor}{2^5} \right).$$

8.6 Interval Chain Properties

In this section we prove a few useful claims about the chain properties. By $\mathbf{birth}_t(i)$ we denote t' greatest such $t' \leq t$ and $q_{t'} < i$. Similarly we define $\mathbf{death}_t(i)$ as t' where $t' > t$ is the smallest such that $q_{t'} < i$ or n if there is not such t'.

We start with claim which limits **depth** and $\boldsymbol{\delta}$.

Claim 8.6.1. *Under Assumptions 8.5.1, we have:*

(1) $\mathbf{depth} \geq 2$.

(2) $1 \leq \boldsymbol{\delta} < 2$.

Proof. Proof of Part (1) is immediate implication of Assumption 8.5.1.1. Thus we focus on Part (2).

Recall that

(i) $\boldsymbol{\delta} = \left(\frac{4m}{n}\right)^{\frac{2}{\mathbf{depth}}}$

(ii) $\frac{1}{2}\log_{16}(n) < \mathbf{depth}$ (since $\mathbf{depth} \geq 2$)

(iii) $m \leq \frac{1}{4} \cdot n^{1+1/16}$ (Assumption 8.5.1.2)

Thus we have

$$\log(\boldsymbol{\delta}) = 2 \cdot \frac{\log \frac{4m}{n}}{\mathbf{depth}} < \frac{4 \cdot \log \frac{4m}{n}}{\log_{16} n} = \frac{16 \cdot \log \frac{4m}{n}}{\log n} \leq \frac{16 \cdot \log n^{\frac{1}{16}}}{\log n} = 1,$$

and obviously $\log(\boldsymbol{\delta}) = \frac{2 \cdot \log \frac{4m}{n}}{\mathbf{depth}} \geq 0$ which finishes the proof. □

In the next claim we show, that not only all intervals in the chain are nonempty at each time $t \geq \mathbf{t_0}$, but also we find lower bounds on their size at each level. We only need this lower bound for the very last interval of the chain, however Claim 8.6.2 provides us better understanding of how the Adversary works.

Claim 8.6.2. *Under Assumptions 8.5.1 for each* $i \in \{1, \ldots, \mathbf{depth}\}$, $t \in \{\mathbf{t_0}, \ldots, n\}$ *it holds that* $|I_t(i)| \geq 16^{\mathbf{depth}-i+1}$.

Proof. First we prove that for $\mathbf{t_0}$. From construction, it follows that $q_{\mathbf{t_0}} = 1$ and $|I_{\mathbf{t_0}}(1)| = \lfloor n/2 \rfloor + 2 \geq \frac{n}{2}$. Since $\mathbf{depth} = \lfloor \log_{16} \frac{n}{2} \rfloor$, the claim is clearly true for $I_{\mathbf{t_0}}(1)$.

Now we proceed by induction on $i \in \{2, \ldots, \mathbf{depth}\}$, assuming that claim is true for $I_{\mathbf{t_0}}(i-1)$. From the Rebuilding Rule it follows that $I_{\mathbf{t_0}}(i)$ may be constructed in two ways.

First assume that $\mathbf{bad_{t_0}}(\mathrm{i}-1)$ is true. Then $|I_{\mathbf{t_0}}(i)| \geq |I_{\mathbf{t_0}}(i-1)|/16$ thus the claim is true.

If $\mathbf{bad_{t_0}}(\mathrm{i}-1)$ is false then notice that

$$|I_{\mathbf{t_0}}(i)| \geq |I_{\mathbf{t_0}}(i-1)| - 2\left\lceil \frac{|I_{\mathbf{t_0}}(i-1)|}{8} \right\rceil \geq |I_{\mathbf{t_0}}(i-1)|/16$$

assuming that $|I_{\mathbf{t_0}}(i-1)| \geq 16$. But it is obviously true for each $I_{\mathbf{t_0}}(i)$ if $i < \mathbf{depth}$.

Now we proceed by induction on t, assuming that claim is true for each $I_{t-1}(i)$. We consider two cases, first let us assume that $i \leq q_t$. In such case we have $I_t(i) = I_{t-1}(i) \cup y_t$ and thus the claim is obviously true.

For $i > q_t$ we proceed as in the case of $\mathbf{t_0}$, the only difference is that we start from level q_t and not from level 1. The proof follows. □

Claim 8.6.3. *Under Assumptions 8.5.1*

(i) for any $t \in \{\mathbf{t_0}+1, \ldots, n\}$ *and* $i \in \{1, \ldots, q_t\}$, *it holds that*

$$\mathbf{Rel}_t \subseteq I_t(i) \setminus \{\min(I_t(i)), \max(I_t(i))\}$$

(ii) for any $t \in \{\mathbf{t_0}+1, \ldots, n\}$ *such that* $q_t < \mathbf{depth}$ *and* $\mathbf{bad}_{\mathrm{t}-1}(\mathrm{q_t})$ *is true it holds that*

$$\mathbf{Rel}_t \subseteq I_t(q_t+1) \setminus \{\min(I_t(q_t+1)), \max(I_t(q_t+1))\}.$$

Proof. We start with Part (i) which is simpler and then we show how to use the same approach for Part (ii). If we prove Part (i) for $i = q_t$ we are done since for $i < q_t$ it holds that $I_t(q_t) \subseteq I_t(i)$.

From the adversary construction we can infer that $y_t > \min(I_{t-1}(\mathbf{depth}))$. To prove $y_t < \max(I_{t-1}(\mathbf{depth}))$ we used $|I_{t-1}(\mathbf{depth})| \geq 2$ (Claim 8.6.2). Since $I_{t-1}(\mathbf{depth}) \subseteq I_{t-1}(q_t)$ we also obtain $y_t > \min(I_{t-1}(q_t))$ and $y_t < \max(I_{t-1}(q_t))$.

Finally, we have to combine the fact from the previous paragraph, the fact that $\mathcal{A}$ is lazy and the fact that $\min(I_{t-1}(q_t)) \notin \mathbf{Rel}_t$ and $\max(I_{t-1}(q_t)) \notin \mathbf{Rel}_t$ and $I_t(q_t) = I_{t-1}(q_t) \cup y_t$ which proves Part (i).

To prove Part (ii) we have to use the fact implied by the Critical Level Choice that if $\mathbf{bad}_t(q_t)$ is *true* then $\min(I_{t-1}(q_t+1)) \notin \mathbf{Rel}_t$ and $\max(I_{t-1}(q_t+1)) \notin \mathbf{Rel}_t$. Otherwise we would choose $q_t - 1$ to be q_t. However using the very same arguments as for Part (i) we again obtain $y_t > \min(I_{t-1}(q_t+1))$ and $y_t < \max(I_{t-1}(q_t+1))$.

Since $I_t(q_t+1) = I_{t-1}(q_t+1) \cup \mathbf{Rel}_t$ proof of Part (ii) follows. □

Claim 8.6.4. *Under Assumptions 8.5.1, for each* $i \in \{1, \ldots, \mathbf{depth}-1\}$, $t \in \{\mathbf{t_0}+1, \ldots, n\}$, *if* $t_b = \mathbf{birth}_t(i)$, *then*

$$I_{t_b}(i+1) \cup \{y_{t_b+1}, \ldots, y_t\} \cup \bigcup_{t'=t_b+1}^{t} \mathbf{Rel}_{t'} = I_t(i+1), \text{ and}$$
$$I_{t_b}(i) \cup \{y_{t_b+1}, \ldots, y_t\} = I_t(i).$$

Proof. We proceed by induction on t. For $t = t_b$ both equations are true trivially. We start with the first equation. For $t > t_b$ recall that $q_t \geq i$ thus two cases may occur:

1. $i+1 \leq q_t$, then Adversary definition implies $I_t(i+1) = y_t \cup I_{t-1}(i+1)$ and $\mathbf{Rel}_t \subseteq I_t(i+1)$ (Claim 8.6.3)
2. $i+1 = q_t+1$, then Rebuilding Rule definition implies $I_t(i+1) = \mathbf{Rel}_t \cup I_{t-1}(i+1)$

Both cases implies $I_t(i+1) = \mathbf{Rel}_t \cup y_t \cup I_{t-1}(i+1)$ which implies the wanted equation immediately.

Second equation is even simpler as for $t > t_b$ we have $q_t \geq i$. □

Claim 8.6.5. *Under Assumptions 8.5.1 for each* $i \in \{1, \ldots, \mathbf{depth}\}$, $t \in \{\mathbf{t_0}, \ldots, n\}$, *if* $t_b = \mathbf{birth}_t(i)$, *then it holds that* $\frac{|I_t(i+1)|}{|I_t(i)|} \geq \frac{|I_{t_b}(i+1)|}{|I_{t_b}(i)|}$.

Proof. From Claim 8.6.4 we have $|I_t(i+1)| \geq |I_{t_b}(i+1)| + t - t_b$ and $|I_t(i)| = |I_{t_b}(i)| + t - t_b$. Thus we can proceed

$$\frac{|I_t(i+1)|}{|I_t(i)|} \geq \frac{|I_{t_b}(i+1)| + t - t_b}{|I_{t_b}(i)| + t - t_b} \geq \frac{|I_{t_b}(i+1)|}{|I_{t_b}(i)|},$$

where the last inequality follows from $|I_{t_b}(i+1)| < |I_{t_b}(i)|$ since $I_{t_b}(i+1) \subset I_{t_b}(i)$. □

Next two claims shows that "badness" and the span of the interval is preserved if the interval is not rebuilt. Preservation Rule immediately implies the following claim.

Claim 8.6.6. *For each* $i \in \{1, \ldots, \mathbf{depth}\}$, $t \in \{\mathbf{t_0}, \ldots, n\}$, *if* $t_b = \mathbf{birth}_t(i)$ *then* $\mathbf{bad}_{\mathrm{t_b}}(\mathrm{i}) = \mathbf{bad}_{\mathrm{t}}(\mathrm{i})$.

Claim 8.6.7. *Under Assumptions 8.5.1 for each* $i \in \{1, \ldots, \mathbf{depth} - 1\}$, $t \in \{\mathbf{t_0}, \ldots, n\}$, *if* $t_b = \mathbf{birth}_t(i)$ *then:*

(i) $\mathbf{span}_{t_b}(I_{t_b}(i)) = \mathbf{span}_t(I_t(i))$

(ii) if additionally $\mathbf{bad}_{\mathrm{t}}(\mathrm{i})$ *is true then* $\mathbf{span}_{t_b}(I_{t_b}(i+1)) = \mathbf{span}_t(I_t(i+1))$.

Proof. We start with Part (i). We proceed by induction on t. For $t = t_b$ the equation is trivial. Now assume that equations are true for $t-1$. Since $i \leq q_t$ we use Claim 8.6.3 Part (i) and we obtain $\mathbf{span}_{t-1}(I_{t-1}(i)) = \mathbf{span}_t(I_t(i))$. Using the induction hypothesis the claim follows.

For Part (ii) we proceed similarly using Claim 8.6.3 Part (ii) together with Claim 8.6.6. □

Lemma 8.6.8. *Under Assumptions 8.5.1 for each* $i \in \{1, \ldots, \mathbf{depth}\}$, $t \in \{\mathbf{t_0}, \ldots, n\}$, *if* $\mathbf{bad}_{\mathrm{t}}(\mathrm{i})$ *is true then* $\rho_t(I_t(i)) < \rho_t(I_t(i+1))/\boldsymbol{\delta}$.

Proof. Let $t_b = \mathbf{birth}_t(i)$. Now we distinguish two situations. If $t = t_b$ then the claim follows immediately from Rebuilding Rule.

Otherwise we proceed with the following chain of inequalities.

$$\begin{aligned}\frac{1}{\boldsymbol{\delta}} > \frac{\rho_{t_b}(I_{t_b}(i))}{\rho_{t_b}(I_{t_b}(i+1))} &= \frac{|I_{t_b}(i)| \cdot (\mathbf{span}_{t_b}(I_{t_b}(i+1)) + 1)}{|I_{t_b}(i+1)| \cdot (\mathbf{span}_{t_b}(I_{t_b}(i)) + 1)} \\ &\overset{(Claim\ 8.6.7)}{=} \frac{|I_{t_b}(i)| \cdot (\mathbf{span}_t(I_t(i+1)) + 1)}{|I_{t_b}(i+1)| \cdot (\mathbf{span}_t(I_t(i)) + 1)} \\ &\overset{(Claim\ 8.6.5)}{\geq} \frac{|I_t(i)| \cdot (\mathbf{span}_t(I_t(i+1)) + 1)}{|I_t(i+1)| \cdot (\mathbf{span}_t(I_t(i)) + 1)} = \frac{\rho_t(I_t(i))}{\rho_t(I_t(i+1))}.\end{aligned}$$

The lemma follows immediately. □

To prove the similar lower bound in case $\mathbf{bad}_{\mathrm{t}}(\mathrm{i})$ is $false$ we start with the following lemma which limits the density of the middle part of the segment given the upper bounds on the remaining parts.

Lemma 8.6.9. *Let* $\mathbf{f}$ *be an allocation of items from a set* Y *by an algorithm* $\mathcal{A}$. *Let density* ρ *and* $\mathbf{span}$ *be measured with respect to this allocation. Let* I *be an interval of* Y. *Let* L, M, R *partition* I *into subintervals of* I, *so that all elements in* L *are smaller than in* M *and all elements in* M *are smaller than in* R. *Let* c *be arbitrary such that* $c > 1$ *and* $\rho(L), \rho(R) < c \cdot \rho(I)$. *If* $|M| \geq |L| + |R|$ *then*

$$\rho(M) \geq \frac{\rho(I)}{c}.$$

Proof. Let $\boldsymbol{\ell} = \mathbf{span}(I) - \mathbf{span}(L) - \mathbf{span}(R) - 2$ and $\mathbf{d} = \frac{|M|}{\boldsymbol{\ell}+1}$. We only prove that $\mathbf{d} \geq \frac{\rho(I)}{c}$ since $\rho(M) \geq \mathbf{d}$ which follows trivially from the fact $\boldsymbol{\ell} \geq \mathbf{span}(M)$. Using this notation we obtain the following chain of equalities and inequalities.

$$\begin{aligned}
\mathbf{d} &= \frac{|M|}{\boldsymbol{\ell}+1} = \frac{|I| - |R| - |L|}{\boldsymbol{\ell}+1} \\
&= \frac{|I| - \rho(R)(\mathbf{span}_t(R)+1) - \rho(L)(\mathbf{span}_t(L)+1)}{\boldsymbol{\ell}+1} \\
&> \frac{|I| - c \cdot \rho(I)(\mathbf{span}(R) + \mathbf{span}(L) + 2)}{\boldsymbol{\ell}+1} \\
&= \frac{|I| - c \cdot \rho(I)(\mathbf{span}(I) - \boldsymbol{\ell})}{\boldsymbol{\ell}+1} \\
&= \frac{(1-c)|I|}{\boldsymbol{\ell}+1} + c\rho(I) \geq 2(1-c) \cdot \mathbf{d} + c \cdot \rho(I),
\end{aligned}$$

where the last inequality follows from the fact that $c > 1$ and $2|M| \geq |I_t(i)|$. Now we can deduce the following

$$\begin{aligned}
\mathbf{d} - 2(1-c)\mathbf{d} &\geq c \cdot \rho(I) \\
\mathbf{d} &\geq \frac{c \cdot \rho(I)}{2c-1} \geq \frac{\rho(I)}{c}.
\end{aligned}$$

Last inequality follows from the fact that $c^2 - 2c + 1 \geq 0$ which implies $\frac{c}{2c-1} \geq \frac{1}{c}$ and thus finishes the proof. □

Lemma 8.6.10. *Under Assumptions 8.5.1, for each* $i \in \{1, \ldots, \mathbf{depth} - 1\}$, $t \in \{\mathbf{t_0}, \ldots, n\}$ *if* $\mathbf{bad}_{\mathrm{t}}(\mathrm{i})$ *is* $false$ *then* $\frac{\rho_t(I_t(i))}{\boldsymbol{\delta}} \leq \rho_t(I_t(i+1))$.

Proof. Let $t_b = \mathbf{birth}_t(i)$, $L' = \{y \in I_t(i); y < \min(I_t(i+1))\}$ and $R' = \{y \in I_t(i); y > \max(I_t(i+1))\}$. From Claim 8.6.4 it follows that for each $y \in I_t(i) \setminus I_t(i+1)$, $\mathbf{f}_{t_b}(y) = \mathbf{f}_t(y)$. Thus from the construction of the adversary and since $\mathbf{bad}_{\mathrm{t}}(\mathrm{i})$ is $false$ we can derive that $\rho_t(L') \leq \boldsymbol{\delta} \cdot \rho_{t_b}(I_{t_b}(i))$ and $\rho_t(R') \leq \boldsymbol{\delta} \cdot \rho_{t_b}(I_{t_b}(i))$. But since $\rho_{t_b}(I_{t_b}(i)) \leq \rho_t(I_t(i))$ (Claim 8.6.4, Claim 8.6.7), the lemma follows from Lemma 8.6.9. □

Now we can prove one of the most important lemmas of the section. We show that the number of good levels is at least a constant fraction of **depth**.

Lemma 8.6.11. *Under Assumptions 8.5.1, for each* $t \in \{\mathbf{t_0}, \ldots, n\}$ *if we define* $\mathbb{S}$ *as follows:*

$$\mathbb{S} = \{i \in \{1, \ldots, \mathbf{depth}\}, \textit{ such that } \mathbf{bad}_{\mathrm{t}}(\mathrm{i}) \textit{ is } false\},$$

then $|\mathbb{S}| \geq \left\lceil \frac{1}{4} \cdot \mathbf{depth} \right\rceil$.

Proof. For the sake of contradiction, let us assume that $|\mathbb{S}| \leq \left\lceil \frac{1}{4} \cdot \mathbf{depth} \right\rceil - 1$. Recall that $\rho_t(I_t(i)) > \boldsymbol{\delta} \cdot \rho_t(I_t(i-1))$ (Lemma 8.6.8) if $\mathbf{bad}_t(i-1)$ is *true* and $\rho_t(I_t(i)) > \frac{\rho_t(I_t(i-1))}{\boldsymbol{\delta}}$ otherwise. In addition, $t \geq \mathbf{t_0}$ ensures that $\rho_t(I_t(1)) = \frac{|Y_t|+2}{m+2} \geq \frac{n}{2m}$ (i.e the density of all items inserted so far). Since $I_t(\mathbf{depth})$ is bad by definition of the Adversary and $\mathbf{depth} \geq 2$ we can estimate the $\rho_t(I_t(\mathbf{depth}))$ as follows:

$$\begin{aligned}
\rho_t(I_t(\mathbf{depth})) &\geq \rho_t(I_t(1)) \cdot \boldsymbol{\delta}^{\frac{3}{4}\mathbf{depth}-\frac{1}{4}\mathbf{depth}-1} \\
&= \frac{n}{2m} \cdot \left(\frac{4m}{n}\right)^{\frac{2}{\mathbf{depth}}\cdot(\frac{2}{4}\mathbf{depth}-1)} \\
&= \frac{n}{2m} \cdot \frac{4m}{n} \cdot \frac{1}{\boldsymbol{\delta}} \overset{(Claim\ 8.6.1.2)}{>} 1.
\end{aligned}$$

Thus we obtain a contradiction since $\rho_t(I_t(\mathbf{depth}))$ is at most 1. □

The last property of the chain we need to showd is that the span of a successive subinterval of an interval is at most constant fraction of the span of the interval.

Claim 8.6.12. *Under Assumptions 8.5.1, for each* $i \in \{1, \ldots, \mathbf{depth} - 1\}$, $t \in \{\mathbf{t_0}, \ldots, n\}$ *it holds*

$$\mathbf{span}_t(I_t(i+1)) + 1 \leq \frac{15}{16} \cdot (\mathbf{span}_t(I_t(i)) + 1).$$

Proof. Let $t_b = \mathbf{birth}_t(i)$. We distinguish two cases:

If $\mathbf{bad}_t(i)$ is *true* then $\mathbf{span}_t(I_t(i+1)) = \mathbf{span}_{t_b}(I_{t_b}(i+1))$ (Claim 8.6.7). However, for $t = t_b$ the claim follows from construction since $|I_{t_b}(i+1)| \leq \frac{1}{2}|I_{t_b}(i)|$ and $\rho_{t_b}(I_{t_b}(i+1)) \geq \rho_{t_b}(I_{t_b}(i))$.

If $\mathbf{bad}_t(i)$ is $false$ we focus on the length of $L_t(i)$. Construction of Adversary implies that $L_{t_b}(i) = L_t(i)$ and since none of the items of $L_{t_b}(i)$ was relabeled since t_b we know that $\mathbf{span}_{t_b}(L_{t_b}(i)) = \mathbf{span}_t(L_t(i))$. The construction of Adversary also implies that $|L_{t_b}(i)| \geq \frac{|I_{t_b}(i)|}{16}$. Thus we obtain:

$$\begin{aligned}
\mathbf{span}_t(L_t(i)) + 1 = \mathbf{span}_{t_b}(L_{t_b}(i)) + 1 &\geq \frac{|L_{t_b}(i)|}{\boldsymbol{\delta} \cdot \rho_{t_b}(I_{t_b}(i))} \\
&\geq \frac{|I_{t_b}(i)|}{16 \cdot \boldsymbol{\delta} \cdot \rho_{t_b}(I_{t_b}(i))} \\
&\overset{(Claim\ 8.6.1.2)}{\geq} \frac{\mathbf{span}_{t_b}(I_{t_b}(i)) + 1}{32}.
\end{aligned}$$

Similarly we proceed for $R_t(i)$. Finally since

$$\mathbf{span}_t(I_t(i+1)) \leq \mathbf{span}_t(I_t(i)) - \mathbf{span}_t(L_t(i)) - \mathbf{span}_t(R_t(i)) - 2$$

and $\mathbf{span}_t(I_t(i)) = \mathbf{span}_{t_b}(I_{t_b}(i))$ (Claim 8.6.7 Part (i)) the proof follows. □

8.7 Relating Charge to Online Labeling Cost

In this section we define a charging scheme which assigns charge to inserted items. Then we show that sum of these charges lower bounds the cost of the corresponding labeling using the Adversary we defined. Finally we lower bound the sum of these charges by showing that there is a constant fraction of items which have charge bigger than some function which depends on **depth**.

Definition 8.7.1. *For $t \in \{\mathbf{t_0}+1,\ldots,n\}$ we say that item y_t was* charged *at level $i \in \{1,\ldots,\mathbf{depth}\}$ iff $t \in \left\{\left\lceil \frac{t_b+t_d}{2}\right\rceil,\ldots,t_d\right\}$ where $t_b = \mathbf{birth}_{t-1}(i)$ and $t_d = \mathbf{death}_{t-1}(i)$ and $\mathbf{bad}_{\mathrm{t}-1}(\mathrm{i})$ is $false$. In such case we set*

$$\mathbf{charge}^i(y_t) = \frac{|I_{t_b}(i)|}{16(t_d - t_b)},$$

otherwise we say that y_t was not charged *and we set $\mathbf{charge}^i(y_t) = 0$.*

First we relate the actual cost of the corresponding labeling to the charge of items. We partition $\mathbf{Rel}_t$ into levels. For each $i \in \{1,\ldots,\mathbf{depth}\}$, $t \in \{\mathbf{t_0}+1,\ldots,n\}$ we define $\mathbf{Rel}_{\mathrm{t}}(i)$ as follows:

1. for $i < \mathbf{depth}$ we set $\mathbf{Rel}_{\mathrm{t}}(i) = \{y \in I_{t-1}(i) \setminus I_{t-1}(i+1); y \in \mathbf{Rel}_t\}$.
2. for $i = \mathbf{depth}$ we set $\mathbf{Rel}_{\mathrm{t}}(i) = \{y \in I_{t-1}(i); y \in \mathbf{Rel}_t\}$.

Following properties of $\mathbf{Rel}_{\mathrm{t}}(i)$ are immediate:

1. $\bigcup_{i=1}^{\mathbf{depth}} \mathbf{Rel}_{\mathrm{t}}(i) \cup y_t = \mathbf{Rel}_t$
2. for each $i,j \in \{1,\ldots,\mathbf{depth}\}$, $i \neq j$ it holds that $\mathbf{Rel}_{\mathrm{t}}(i) \cap \mathbf{Rel}_{\mathrm{t}}(j) = \emptyset$.

These properties imply immediately the following claim:

Claim 8.7.2. *It holds that $\chi_{\mathcal{A}}((y_1,y_2,\ldots,y_n),m) \geq \sum_{t=\mathbf{t_0}+1}^{n} \sum_{i=1}^{\mathbf{depth}} \mathbf{Rel}_{\mathrm{t}}(i)$.*

Now we focus on good intervals for a while.

Claim 8.7.3. *Let $t_b \in \{\mathbf{t_0},\ldots,n\}$ and $i \in \{1,\ldots,\mathbf{depth}\}$ such that $t_b = \mathbf{birth}_{t_b}(i)$ and let $t_d = \mathbf{death}_{t_b}(i)$. If $\mathbf{bad}_{\mathrm{t_b}}(\mathrm{i})$ is $false$ then it holds that*

$$\sum_{t'=t_b+1}^{t_d} |\mathbf{Rel}_{\mathrm{t}'}(i)| \geq \lceil |I_{t_b}(i)|/16 \rceil.$$

Proof. Since $t_d = \mathbf{death}_{t_d-1}(i)$ then $L_{t_d-1}(i) \cap \mathbf{Rel}_{t_d} \neq \emptyset$ or $R_{t_d-1}(i) \cap \mathbf{Rel}_{t_d} \neq \emptyset$ (or both). Let us assume that $L_{t_d-1}(i) \cap \mathbf{Rel}_{t_d} \neq \emptyset$ as the other case is symmetric. Let $L' = \{y \in I_{t_b}(i) \setminus L_{t_b}(i); y < \min(I_{t_b}(i+1))\}$.

First we show that for each $t' \in \{t_b, \ldots, t_d\}$ it holds that for each $y \in L'$ either $y \in I_{t'}(i) \setminus I_{t'}(i+1)$ or $y \in \bigcup_{t=t_b+1}^{t'} \mathbf{Rel}_{\mathrm{t}}(i)$. For $t' = t_b$ this is true immediately as it follows from the construction of the adversary.

Now consider $t' \in \{t_b + 1, \ldots, t_d\}$ such that for $t' - 1$ the above claim is true. Thus we can distinguish the following cases for each $y \in L'$:

(i) $y \in \bigcup_{t=t_b+1}^{t'-1} \mathbf{Rel}_{\mathrm{t}}(i)$: it is obviously true that y is also in $\bigcup_{t=t_b+1}^{t'} \mathbf{Rel}_{\mathrm{t}}(i)$

(ii) $y \notin \bigcup_{t=t_b+1}^{t'-1} \mathbf{Rel}_{\mathrm{t}}(i)$: then two cases may occur

- *(a)* $y \in \mathbf{Rel}_{t'}$: then $y \in \bigcup_{t=t_b+1}^{t'} \mathbf{Rel}_{\mathrm{t}}(i)$ since $y \in I_{t'-1}(i) \setminus I_{t'-1}(i+1)$
- *(b)* $y \notin \mathbf{Rel}_{t'}$: then Claim 8.6.4 implies that $y \in I_{t'}(i) \setminus I_{t'}(i+1)$ since $y \in I_{t'-1}(i) \setminus I_{t'-1}(i+1)$

Now we show that all items $y \in L'$ which are not int $y \in \bigcup_{t=t_b+1}^{t_d-1} \mathbf{Rel}_{\mathrm{t}}(i)$ are in $\mathbf{Rel}_{\mathrm{t_d}}(i)$. Notice that for all such y's it holds that $\max(L_{t_d-1}(i)) < y < \min(I_{t_d-1}(i))$. Together with the facts that $\mathcal{A}$ is lazy, $L_{t_d-1}(i) \cap \mathbf{Rel}_{t_d} \neq \emptyset$ and y_{t_d} was inserted "into" $I_{t_d-1}(i)$ it follows that such y's are in $\mathbf{Rel}_{\mathrm{t_d}}(i)$.

Thus we can infer that $L' + \{\max(I_{t_d-1}(i))\} \subseteq \bigcup_{t_b+1}^{t_d} \mathbf{Rel}_{\mathrm{t_d}}(i)$ and the claim follows since $|L'| = \lfloor |I_{t_b}(i)|/16 \rfloor$ which follows from the Rebuilding Rule for good levels.

□

Lemma 8.7.4. *Under Assumptions 8.5.1 let $\mathcal{A}$ be a lazy online labeling algorithm with the parameters (n, m). If $y_1, y_2, \ldots, y_n$ is the output of* $\mathbf{Adversary}(\mathcal{A}, n, m)$ *then* $\chi_{\mathcal{A}}((y_1, y_2, \ldots, y_n), m) \geq \sum_{i=1}^{\mathbf{depth}} \sum_{t=\mathbf{t_0}+1}^{n} \mathbf{charge}^i(y_t)$.

Proof. Let us fix some $i \in \{1, \ldots, \mathbf{depth}\}$. Now we define a sequence b_i as follows

1. $b_0 = \mathbf{t_0}$
2. for $i > 0$, $b_i = \mathbf{death}_{b_{i-1}}(i)$
3. The sequence ends for j, such that $b_j = n$.

Now consider the sequence of time intervals T_k where $T_k = \{b_k + 1, \ldots, b_{k+1}\}$. By $\mathbf{charge}(T_k)$ we denote $\sum_{t \in T_k} \mathbf{charge}^i(y_t)$. If $\mathbf{bad}_{\mathrm{b_k}}(\mathrm{i})$ is $false$ we can infer from **charge** definition

$$\mathbf{charge}(T_k) \leq \frac{|I_{b_k}(i)|}{16(b_{k+1} - b_k)} \cdot \left\lceil \frac{b_{k+1} - b_k}{2} \right\rceil,$$

as the charge of items $y_{\lfloor (b_k+b_{k+1})/2 \rfloor}, y_{b_k+1}, \ldots, y_{b_{k+1}}$ at level i is the same for all of them.

If $\mathbf{bad}_{\mathrm{b_k}}(\mathrm{i})$ is *true* we obtain $\mathbf{charge}(T_k) = 0$.

Now we use Claim 8.7.3 to proceed as follows

$$\sum_{t \in T_k} \mathbf{Rel}_{\mathrm{t}}(i) \geq \frac{|I_{b_k}(i)|}{16} \geq \mathbf{charge}(T_k),$$

if $\mathbf{bad}_{\mathrm{b_k}}(\mathrm{i})$ is $false$, and $\sum_{t \in T_k} \mathbf{Rel}_{\mathrm{t}}(i) \geq 0$ otherwise.

Let $\mathbb{T}_j$ denote the set of all time intervals for $j \in \{1, \ldots, \mathbf{depth}\}$. Combining above facts together with Claim 8.7.2 we obtain

$$\chi_{\mathcal{A}}\left((y_1, y_2, \ldots, y_n), m\right) \geq \sum_{i=1}^{\mathbf{depth}} \sum_{T \in \mathbb{T}_i} \mathbf{charge}(T) = \sum_{i=1}^{\mathbf{depth}} \sum_{t=\mathbf{t_0}+1}^{n} \mathbf{charge}^i(y_t),$$

where the first inequality follows from the fact that intervals for the same level are distinct as well as charges of particular item among all levels. □

8.8 Estimating the Charge

In this section we prove a lower bound on the charge for items which are charged on many levels. We proceed in two major steps. First we show that there are many items which are charged on many levels. Then we estimate charge for such items.

First we define $\mathbf{charge}_i^{\#}(y)$ to be one if y is charged at level i and 0 otherwise. We say that item y is *heavily charged* if $\sum_{i=1}^{\mathbf{depth}} \mathbf{charge}_i^{\#}(y) \geq \left\lceil \frac{\mathbf{depth}}{16} \right\rceil$. Now we can lower bound the overall number of charges.

Claim 8.8.1. *Under Assumptions 8.5.1 there are at least* $\lceil \frac{n}{32} \rceil$ *items among* $\{y_{\mathbf{t_0}+1}, y_{\mathbf{t_0}+2}, \ldots, y_n\}$ *which are heavily charged.*

Proof. First we show that $\mathbf{count} = \sum_{t=\mathbf{t_0}+1}^{n} \sum_{i=1}^{\mathbf{depth}} \mathbf{charge}_i^{\#}(y_t) \geq \frac{1}{2} \cdot \frac{\mathbf{depth}}{4} \cdot \frac{n}{2} = \frac{n \cdot \mathbf{depth}}{16}$.

Notice that $|\{y_{\mathbf{t_0}+1}, y_{\mathbf{t_0}+2}, \ldots, y_n\}| \geq \frac{n}{2}$. Obviously at least one half of items inserted to particular level when it was good was charged and the number of considered items is at least $\frac{n}{2}$. In addition, no item can be charged twice on any level. Putting this together with the Lemma 8.6.11 we obtain requested lower bound on **count**.

Now for the sake of contradiction, let us assume that there is at most $\lceil \frac{n}{32} \rceil - 1$ items such that they are charged on at least $\lceil \frac{\mathbf{depth}}{16} \rceil$ levels while remaining $\frac{n}{2} -$

$\left\lceil \frac{n}{32} \right\rceil + 1$ items are charged on at most $\left\lfloor \frac{\mathbf{depth}}{16} \right\rfloor$ levels. Under this assumption we obtain, that the number of charged items is at most

$$\left(\left\lceil \frac{n}{32} \right\rceil - 1\right) \cdot \mathbf{depth} + \left(\frac{n}{2} - \left\lceil \frac{n}{32} \right\rceil + 1\right) \cdot \left\lfloor \frac{\mathbf{depth}}{16} \right\rfloor < \frac{n}{32}\mathbf{depth} + \frac{n}{32}\mathbf{depth}.$$

Thus we have that number of charged items is smaller than $\frac{n \cdot \mathbf{depth}}{16}$ and therefore we obtain a contradiction. □

Before we state next claim we introduce $\hat{\rho}_t(I_t(i))$ to be equal to $\frac{|I_t(i)|+1}{\mathbf{span}_t(I_t(i))}$. This differs from $\rho_t(I_t(i))$ by increasing size of $I_t(i)$ by one. Now we lower bound the charge for each charged item.

Lemma 8.8.2. *Under Assumptions 8.5.1 let y_t be item charged on level i. Then*

$$\mathbf{charge}^i(y_t) \geq \frac{1}{512} \frac{\boldsymbol{\delta}\hat{\rho}_{t-1}(I_{t-1}(i))}{\boldsymbol{\delta}^2\hat{\rho}_{t-1}(I_{t-1}(i+1)) - \hat{\rho}_{t-1}(I_{t-1}(i))} - \frac{1}{32 \cdot \boldsymbol{\delta}^2}.$$

To prove this lemma we first prove a couple of claims. The proof of the lemma itself is on page 93. We start by showing some properties of $\hat{\rho}$ similar to properties of ρ in Lemma 8.6.8 and Lemma 8.6.10.

Claim 8.8.3. *Under Assumptions 8.5.1 for each $i \in \{1, \ldots, \mathbf{depth} - 1\}$, $t \in \{\mathbf{t_0}, \ldots, n\}$ it holds that $\hat{\rho}_t(I_t(i)) < \boldsymbol{\delta} \cdot \hat{\rho}_t(I_t(i+1))$.*

Proof. If $\hat{\rho}_t(I_t(i+1)) \geq \hat{\rho}_t(I_t(i))$ the claim is trivial thus we focus on the other case. From Lemma 8.6.8 and Lemma 8.6.10 we can deduce that $\frac{1}{\boldsymbol{\delta}} \leq \frac{\rho_t(I_t(i+1))}{\rho_t(I_t(i))}$. Now since

$$\frac{\hat{\rho}_t(I_t(i+1))}{\hat{\rho}_t(I_t(i))} = \frac{\rho_t(I_t(i+1)) + \frac{1}{\mathbf{span}_t(I_t(i+1))}}{\rho_t(I_t(i)) + \frac{1}{\mathbf{span}_t(I_t(i))}}, \text{ and}$$

$$\frac{1}{\mathbf{span}_t(I_t(i))} \leq \frac{1}{\mathbf{span}_t(I_t(i+1))}$$

proof follows immediately. □

Next we continue with the claim which shows an upper bound on $\mathbf{birth}_t(i) - \mathbf{death}_t(i)$ (which equals to denominator of charge of y_t).

Claim 8.8.4. *Under Assumptions 8.5.1 let y_t be the item charged at level i. Let $t_b = \mathbf{birth}_{t-1}(i)$ and $t_d = \mathbf{death}_{t-1}(i)$. Then*

$$t_b - t_d \leq 2 \cdot (|I_{t-1}(i)| - |I_{t_b}(i)| + 1).$$

Proof. Since y_t was charged at level i it holds that $t \geq \left\lceil \frac{t_d+t_b}{2} \right\rceil$. Thus we can infer

$$2 \cdot (t - t_b) \geq 2 \cdot \left(\left\lceil \frac{t_d + t_b}{2} \right\rceil - t_b \right) \geq t_d + t_b - 2 \cdot t_b = t_d - t_b.$$

It remains to realize that $t - t_b = |I_{t-1}(i)| - |I_{t_b}(i)| + 1$ (Claim 8.6.4). □

Now we can estimate $\rho_{t_b}(I_{t_b}(i))$.

Claim 8.8.5. *Under Assumptions 8.5.1 let* $t \in \{\mathbf{t_0} + 1, \ldots, n\}$ *and* $i \in \{1, \ldots, \mathbf{depth} - 1\}$ *be such that* $\mathbf{bad}_t(i)$ *is false. Let* $t_b = \mathbf{birth}_{t-1}(i)$*. It holds*

$$\rho_{t_b}(I_{t_b}(i)) \geq \frac{|I_{t-1}(i)| - |I_{t-1}(i+1)|}{\boldsymbol{\delta} \cdot (\mathbf{span}_{t-1}(I_{t-1}(i)) - \mathbf{span}_{t-1}(I_{t-1}(i+1)))}.$$

Proof. Let $L' = \{y \in I_{t-1}(i); y < \min(I_{t-1}(i+1))\}$ and $R' = \{y \in I_{t-1}(i); y > \max(I_{t-1}(i+1))\}$. From Claim 8.6.4 we can infer that none of items of L' and R' was relabeled since t_b. Thus we can infer from the definition of the adversary that $\rho_t(L') \leq \boldsymbol{\delta} \cdot \rho_{t_b}(I_{t_b}(i))$ and $\rho_t(R') \leq \boldsymbol{\delta} \cdot \rho_{t_b}(I_{t_b}(i))$. Thus we obtain

$$\begin{aligned} \frac{|I_{t-1}(i)| - |I_{t-1}(i+1)|}{\boldsymbol{\delta}(\mathbf{span}_{t-1}(I_{t-1}(i)) - \mathbf{span}_{t-1}(I_{t-1}(i+1)))} &\leq \frac{|R'| + |L'|}{\boldsymbol{\delta} \cdot (\mathbf{span}_{t_b}(R') + \mathbf{span}_{t_b}(L') + 2)} \\ &\leq \rho_{t_b}(I_{t_b}(i)). \end{aligned}$$

The last inequality follows from the fact that $|R'| \leq \boldsymbol{\delta} \cdot \rho_{t_b}(I_{t_b}(i)) \cdot (\mathbf{span}_{t_b}(R') + 1)$ and also $|L'| \leq \boldsymbol{\delta} \cdot \rho_{t_b}(I_{t_b}(i)) \cdot (\mathbf{span}_{t_b}(L') + 1)$. □

Merging these claims together we can finish the proof of Lemma 8.8.2.

Proof of Lemma 8.8.2. To make the computations easier to follow we introduce the following notations. We set $I = I_{t-1}(i), I' = I_{t-1}(i+1), t_b = \mathbf{birth}_t(i), t_d = \mathbf{death}_t(i), \ell = \mathbf{span}_{t-1}(I) + 1, \ell' = \mathbf{span}_{t-1}(I') + 1, \hat{\rho} = \hat{\rho}_{t-1}(I)$ and $\hat{\rho}' = \hat{\rho}_{t-1}(I')$.

The proof is done by the following chain of inequalities

$$\begin{aligned}
\mathbf{charge}^i(y_t) &= \frac{|I_{t_b}(i)|}{16(t_b - t_d)} \geq \frac{|I_{t_b}(i)|}{16 \cdot 2 \cdot (|I| - |I_{t_b}(i)| + 1)} \text{ (Claim 8.8.4)} \\
&\geq \frac{1}{32} \cdot \frac{\ell \cdot \rho_{t_b}(I_{t_b}(i))}{\ell \cdot \hat{\rho} - \ell \cdot \rho_{t_b}(I_{t_b}(i))} \text{ (Lemma 8.6.7)} \\
&\geq \frac{1}{32} \cdot \frac{\frac{|I|-|I'|}{\boldsymbol{\delta}(\ell-\ell')}}{\boldsymbol{\delta}\hat{\rho}' - \frac{|I|-|I'|}{\boldsymbol{\delta}(\ell-\ell')}} \text{ (Claim 8.8.5, Claim 8.8.3)} \\
&= \frac{1}{32} \cdot \frac{|I| + 1 - |I'| - 1}{\boldsymbol{\delta}^2 \cdot \hat{\rho}'\ell - \boldsymbol{\delta}^2 \cdot \hat{\rho}'\ell' - |I| - 1 + |I'| + 1} \\
&= \frac{1}{32} \cdot \frac{\hat{\rho}\ell - \hat{\rho}'\ell'}{\boldsymbol{\delta}^2 \cdot \hat{\rho}'\ell - \boldsymbol{\delta}^2 \cdot \hat{\rho}'\ell' - \hat{\rho}\ell + \hat{\rho}'\ell'} \geq \frac{1}{32} \cdot \frac{\hat{\rho} - \hat{\rho}'\frac{\ell'}{\ell}}{\boldsymbol{\delta}^2\hat{\rho}' - \hat{\rho}} \text{ (Claim 8.6.1.2)} \\
&= \frac{1}{32} \cdot \frac{\hat{\rho} - \hat{\rho}'\frac{\ell'}{\ell} + \hat{\rho}'\frac{\ell'}{\boldsymbol{\delta}^2\ell} - \hat{\rho}'\frac{\ell'}{\boldsymbol{\delta}^2\ell}}{\boldsymbol{\delta}^2\hat{\rho}' - \hat{\rho}} = \frac{1}{32} \cdot \frac{\ell'(-\boldsymbol{\delta}^2\hat{\rho}' + \hat{\rho})}{\boldsymbol{\delta}^2\ell \cdot (\boldsymbol{\delta}^2\hat{\rho}' - \hat{\rho})} + \frac{1}{32} \cdot \frac{\hat{\rho} - \hat{\rho}\frac{\ell'}{\ell\boldsymbol{\delta}^2}}{\boldsymbol{\delta}^2\hat{\rho}' - \hat{\rho}} \\
&\geq -\frac{1}{32\boldsymbol{\delta}^2} + \frac{1}{32 \cdot 16} \cdot \frac{\hat{\rho}}{\boldsymbol{\delta}^2\hat{\rho}' - \hat{\rho}} \text{ (Claim 8.6.1.2, Lemma 8.6.12)}
\end{aligned}$$

□

Now we focus on heavily charged items.

Lemma 8.8.6. *For each heavily charged item y_t it holds that*

$$\sum_{i=1}^{\mathbf{depth}} \mathbf{charge}^i(y_t) \geq \frac{\mathbf{depth}^2}{2^{45} \cdot (\log m - \log n + 2)} - \frac{\mathbf{depth}}{2^{10}}.$$

Proof. During the proof we fix time t when y_t was inserted. We define $\mathbb{C}$ to be a set of i's such that y_t was charged on the level i. Obviously

$$\begin{aligned}
\sum_{i=1}^{\mathbf{depth}} \mathbf{charge}^i(y_t) &= \sum_{i \in \mathbb{C}} \mathbf{charge}^i(y_t) \\
&\geq \sum_{i \in \mathbb{C}} -\frac{1}{32 \cdot \boldsymbol{\delta}^2} + \frac{1}{512} \cdot \frac{\hat{\rho}_{t-1}(I_{t-1}(i))}{\boldsymbol{\delta}^2\hat{\rho}_{t-1}(I_{t-1}(i+1)) - \hat{\rho}_{t-1}(I_{t-1}(i))}.
\end{aligned}$$

We focus on $\sum_{i \in \mathbb{C}} \frac{\hat{\rho}_t(I_t(i))}{\boldsymbol{\delta}^2\hat{\rho}_t(I_t(i+1)) - \hat{\rho}_t(I_t(i))}$ as the summation of the remaining parts is simple. Let $r_i = \frac{\hat{\rho}_t(I_t(i+1))}{\hat{\rho}_t(I_t(i))}$. Notice that Claim 8.8.3 implies that $r_i \geq \frac{1}{\boldsymbol{\delta}}$. We use this fact in the following claim.

Claim 8.8.7. *Let $\mathbf{r} = \prod_{i \in \mathbb{C}} r_i$. Then $\mathbf{r} \leq \frac{4m}{n} \cdot \boldsymbol{\delta}^{\mathbf{depth}}$.*

Proof. First notice that $\prod_{i\in\mathbb{C}} r_i \leq \boldsymbol{\delta}^{\mathbf{depth}-|\mathbb{C}|} \cdot \prod_{i=1}^{\mathbf{depth}-1} r_i$. It follows from the fact that each r_i, it holds that $r_i\boldsymbol{\delta} \geq 1$ (Claim 8.8.3). Also we have $\prod_{i=1}^{\mathbf{depth}-1} r_i \leq \frac{4m}{n}$ since for each $t \geq \mathbf{t_0}$ we have $\hat{\rho}_t(I_t(1)) \geq \rho_t(I_t(1)) \geq \frac{n+2}{2m+4} \geq \frac{n}{2m}$ and $\hat{\rho}_t(I_t(\mathbf{depth})) \leq \rho_t(I_t(\mathbf{depth})) + 1 \leq 2$. Finally since $\boldsymbol{\delta} \geq 1$ (Claim 8.6.1.2) the proof is finished. □

Previous claim implies the following claim.

Claim 8.8.8. *It holds that* $(\log_2 \mathbf{r})/|\mathbb{C}| \leq 24 \log_2 \boldsymbol{\delta}$.

Proof. First we rewrite the expression as follows using the Claim 8.8.7 and Claim 8.8.1.

$$\log(\mathbf{r})/|\mathbb{C}| = \frac{\log\left(\frac{4m}{n} \cdot \boldsymbol{\delta}^{\mathbf{depth}}\right)}{|\mathbb{C}|} \leq 16 \cdot \log \boldsymbol{\delta} + 16 \cdot \frac{\log \frac{4m}{n}}{\mathbf{depth}}.$$

Recall that $\log \boldsymbol{\delta} = 2 \cdot \frac{\log \frac{4m}{n}}{\mathbf{depth}}$ and therefore we can use Claim 8.6.1.2 to obtain the following

$$16 \cdot \log \boldsymbol{\delta} + 16 \cdot \frac{\log \frac{4m}{n}}{\mathbf{depth}} = 24 \log \boldsymbol{\delta},$$

which finishes the proof. □

Now we use these claims to finish the proof of the Lemma 8.8.6. First notice that

$$\begin{aligned}
\sum_{i\in\mathbb{C}} \frac{\hat{\rho}_t(I_t(i+1))}{\boldsymbol{\delta}^2\hat{\rho}_t(I_t(i+1)) - \hat{\rho}_t(I_t(i))} &= \sum_{i\in\mathbb{C}} \frac{r_i}{\boldsymbol{\delta}^2 r_i - 1} \\
&\geq \frac{1}{\boldsymbol{\delta}} \sum_{i\in\mathbb{C}} \frac{1}{\boldsymbol{\delta}^2 r_i - 1} \\
&= \frac{1}{\boldsymbol{\delta}} \sum_{i\in\mathbb{C}} \frac{1}{\boldsymbol{\delta}^2 2^{\log r_i} - 1}.
\end{aligned}$$

To minimize this sum we use Jensen's inequality. Consider the function $f(x) = \frac{1}{\boldsymbol{\delta}^2 2^x - 1}$ and logarithm of $\prod_{i\in\mathbb{C}} r_i$ which is $\sum_{i\in\mathbb{C}} \log r_i$. Since $f(x)$ is convex for $x > 2 \cdot \log \frac{1}{\boldsymbol{\delta}}$ it follows from Jensen's inequality that our sum is minimized when all r_i are equal. Thus we choose $r_i = \mathbf{r}^{1/|\mathbb{C}|}$ and we continue as follows

$$\frac{1}{\boldsymbol{\delta}} \cdot \sum_{i\in\mathbb{C}} \frac{1}{\boldsymbol{\delta}^2 r_i - 1} \geq \frac{1}{\boldsymbol{\delta}} \cdot \frac{|\mathbb{C}|}{\boldsymbol{\delta}^2 \mathbf{r}^{1/|\mathbb{C}|} - 1} = \frac{1}{\boldsymbol{\delta}} \cdot \frac{|\mathbb{C}|}{2^{2\cdot\log_2 \boldsymbol{\delta} + (\log_2 \mathbf{r})/|\mathbb{C}|} - 1}.$$

Now Claim 8.6.1.2 and Claim 8.8.8 imply that

$$2 \cdot \log_2 \boldsymbol{\delta} + (\log_2 \mathbf{r})/|\mathbb{C}| \leq 26 \cdot \log_2 \boldsymbol{\delta} \leq 26.$$

Thus we can use the fact that for x, $0 \leq x \leq 26$ it holds that $2^x - 1 \leq 2^{26} \cdot x$. Applying this to the fraction we have, we obtain

$$\begin{aligned}\frac{1}{\boldsymbol{\delta}} \cdot \frac{|\mathbb{C}|}{2^{\log_2 \boldsymbol{\delta} + (\log_2 \mathbf{r})/|\mathbb{C}|} - 1} &\geq \frac{1}{\boldsymbol{\delta}} \cdot \frac{|\mathbb{C}|}{2^{26} \cdot 26 \cdot \log \boldsymbol{\delta}} \\ &\geq \frac{|\mathbb{C}|}{\boldsymbol{\delta} \cdot 2^{26} \cdot 26 \cdot 16 \cdot \frac{\log \frac{4m}{n}}{|\mathbb{C}|}} \geq \frac{|\mathbb{C}|^2}{\boldsymbol{\delta} \cdot 2^{35} \cdot \log \frac{4m}{n}}.\end{aligned}$$

To finish the proof of Lemma 8.8.6 we plug this into the original equation so that we obtain

$$\begin{aligned}\sum_{i \in \mathbb{C}} \mathbf{charge}^i(y_t) &\geq -\frac{\mathbb{C}}{32 \cdot \boldsymbol{\delta}} + \frac{\boldsymbol{\delta}}{512} \cdot \frac{|\mathbb{C}|^2}{\boldsymbol{\delta} \cdot 2^{35} \cdot \log \frac{4m}{n}} \\ &\geq \frac{\mathbf{depth}^2}{2^{52} \cdot (\log m - \log n + 2)} - \frac{\mathbf{depth}}{2^5},\end{aligned}$$

where the last inequality follows from the Claim 8.8.1 and Claim 8.6.1.2. □

Now we have all the necessary ingredients to prove the main lemma of the section.

Proof of Lemma 8.5.2. The lemma is an immediate consequence of the Claim 8.8.1 and Lemma 8.8.6 □

9. Online Labeling with Small Label Space and Universe

9.1 Introduction

In this chapter we prove an $\Omega(n \log^2(n))$ lower bound on the number of moves for inserting n items into an array of size $m = O(n)$ for any deterministic online labeling algorithm, matching the known upper bound up to constant factors. For the case of array of size $m \leq n + n^{1-\varepsilon}$ (where ε is a positive constant) we prove the asymptotically optimal lower bound $\Omega(n \log^3(n))$.

All upper bounds from Part I work for any input domain size (arbitrary r). As noted before, if $r \leq m$ then there is a trivial solution of cost n. A natural question (which has not, to our knowledge, been addressed previously in the literature) is whether it might be possible to improve the $O(n \log^2(n))$ upper bound if r is much larger than m but still restricted. Our lower bounds rule out such an improvement: we obtain an $\Omega(n \log^2(n))$ lower bound provided that r is at least a sufficiently large constant times m.

Our lower bounds extend to slightly superlinear array size. For example, in the case $m = O(n \log^{1-\varepsilon}(n))$ our results give an $\Omega(n \log^{1+\varepsilon/3}(n))$ lower bound (provided that the range size r is large enough). This, however, is inferior to what we showed in Chapter 8. On the other hand, in Chapter 8 we do not handle optimally arrays of size $m \leq n + n^{1-\varepsilon}$ (i.e. very small arrays) and we do not consider the case of small item universe.

This chapter is based on joint work with Michal Koucký and Michael Saks [10] which was the first lower bound for general algorithms in the small space regime. Previously Dietz et al. [12, 21] proved a $\Omega(n \log^2(n))$ lower bound for the restricted case of so-called *smooth algorithms*. These lower bounds are especially interesting because the best known algorithms for the problem are smooth. Nevertheless, the restriction to smooth algorithms is significant. The lower bound for the small space regime of smooth algorithms is obtained by considering the trivial adversary that inserts items in decreasing order. This lower bound clearly relies heavily on the smoothness of the algorithm; a non-smooth algorithm can easily handle the given adversary with constant amortized time per item. It was not known whether a non-smooth algorithm could give a significant advantage over a smooth algorithm on general inputs. Our lower bound rules this out.

There is some confusion in the literature regarding the lower bounds of [12, 21]; the fact that they apply only to smooth algorithms is sometimes not mentioned ([7]), creating the impression that the general lower bound was already established.

9.2 Proof Techniques

We will describe our results in the language of the file maintenance problem, in which arriving items are placed in an array, rather than the online labeling problem. Our main lemma (Lemma 9.4.1) gives a lower bound on the cost of inserting n additional items into an array that is already partially full with $n_0 \geq n$ items. The $\Omega(n \log^2(n))$ lower bound for file maintenance problem for the case that $m = \Theta(n)$ is obtained by applying this lemma to bound the cost of inserting the second half of the items given that the first half of the items are initially in the array. The $\Omega(\log^3(n))$ lower bound for an array of size $m \leq n + n^{1-\varepsilon}$ is obtained by iterative application of the main lemma $\Theta(\log n)$ times where we initially start with half of the items in the array, and in each iteration we insert half of the remaining items. The main lemma shows that each iteration costs $\Theta(n \log^2(n))$. This parallels the structure of the iterative algorithm of Chapter 3 that gives a matching upper bound.

The general idea for proving the lemma builds heavily on the prior work [12, 21, 13] and is similar to ideas in Chapter 8. Adversary constructions in the previous chapters use a chain of intervals of items as a guide to selecting the next item. In this section we use a dual approach, the adversary will maintain a chain of nested segments of the array. Recall, that the key challenge is to give an adversary procedure for building segment chains that ensures large buffers, but the density degrades very slowly. The previous chapter builds on the observation that if for a given segment every subsegment having large buffers has density significantly smaller than the given segment, then there must be a large subsegment (located near the boundary of the given segment) having substantially higher density than the given segment. This allows us to build a chain of $\Theta(\log(n))$ segments, such that (1) a constant fraction of the segments have large buffers with respect to their predecessors (good segments), (2) the segments that don't have large buffers have significantly higher density than their predecessor segments (bad segments), and (3) the degradation of density along the entire chain can be bounded by a constant factor. (To give a rough idea of the choice of parameters, when $m = \Theta(n)$, we allow decrease in density by a factor of at most $(1 - O(1/\log(n)))$ in a single step.)

The adversary used in this chapter doesn't work exactly like this. Unlike in previous chapter we collapse consecutive bad segments into one and then we omit them from the chain. Thus, the chain of segments is built so that every segment in the chain (not just a constant fraction) has left and right buffers whose sizes are a constant fraction of the length of the segment. We do this by relaxing the requirement that the length of each segment in the chain is at least a constant fraction of the length of its predecessor. If we encounter a segment whose items are concentrated near the boundary then the next segment will be a small subsegment of high enough density whose distance from the boundary is large relative to its own

size, even if this distance is small relative to the size of the predecessor segment. This raises a new problem: once we allow successive subsegments to shrink by more than a constant fraction we face the problem that the length of the segment chain d may not be $\Omega(\log(n))$. This is important because the lower bound that comes out of the analysis is proportional to d^2. So we need to ensure that the chains have length $\Omega(\log(n))$ even though we allow the length of segments to drop significantly. This is accomplished by a procedure (see Lemma 9.11.2) that allows us to construct a sequence of segments where each selected segment satisfies a strong uniformity property called *lower balance*: It has no subsegment of length at least 1/4 of its length that has density significantly smaller than the given segment. (In the lower bound for smooth algorithms mentioned earlier, the ability to find such a sequence is essentially built into the smoothness restriction. Our construction allows us to dispense with this assumption.) To find the successor segment S' of a given segment S we first restrict to the middle third T of the segment and choose S' to be a subsegment of T. The restriction to a subsegment of T ensures that S' has large buffers relative to S. Furthermore, the uniformity property of S ensures that the density of T is close to that of S. We want to choose S' inside T having density at least that of T, having size not much smaller than T, and having the desired uniformity property. To identify S' we maximize a certain quality function of the form $\rho(I)|I|^\kappa$ (where $\rho(I)$ is the density of items stored in I and κ is a small positive parameter). This balances the requirement that S' have high density and large size. Furthermore, choosing S' in this way guarantees that S' has the needed uniformity property (since the presence of a subsegment that violates the uniformity property would imply that there is a subsegment of S' that has a higher quality). Maximizing the quality function implicitly captures the process of successively choosing subsegments of significantly higher density until one arrives at a subsegment for which no such selection is possible.

After identifying a segment chain with the required properties at each step, the item selected by the adversary to insert is one whose value is between two items stored in the final segment of the chain. Whenever the maintenance algorithm rearranges some portion of the array the adversary rebuilds the affected portion of the segment chain. To obtain our lower bound $\Omega(n \log^2(n))$ we use a careful accounting argument (see Lemma 9.7.1) that encapsulates and extends the clever argument used in [12, 21] to obtain an $\Omega(n \log^2(n))$ bound for smooth algorithms. (Which are similar to Chapter 8)

There is one additional complication that arises because we want our lower bounds to apply even in the case that the range r of items is relatively small. In a given step, after selecting the chain, the adversary is supposed to choose the next item to insert to be an item that is between two items currently stored in the final segment of the chain. However, if the set of items stored in the final segment

are a consecutive subset of the set of possible values, the adversary is unable to choose an item to insert. Of course this is not a problem if the range of values is, for example, all rationals in a given interval but it is a potential problem if the range is a bounded subset of the integers. For example, we noted earlier that if the range of possible items is small enough, $r \leq m$, then there is a trivial algorithm that incurs only unit cost per item, so our lower bound proof must fail. How large does r have to be so that the adversary described above can avoid this problem? It is not hard to show that $r \geq 2^n$ is sufficient, but in fact when $m = O(n)$ we only need r to be a (sufficiently large) constant multiple of n. To carry out the argument for such small r, we modify the definition of density of segments by weighting more recent items with a smaller (but still non-negligible) weight than older items. When our adversary selects a segment chain, the decreased weight on recent items will tilt the adversary to prefer segments that are crowded mainly with older items over segments crowded mainly with newer items (unless the latter is significantly more crowded than the former). The reason we want to do this is that if the adversary continues to place items in a segment that mainly has newer items there is a risk (because of the limited range size r) that the adversary will end up with a segment where the items are consecutive integers, and not have another item to insert. By giving more recent items a smaller (but not too small) weight we can avoid this possibility; see Lemma 9.6.1.

9.3 The Main Results

In this section, we state our lower bound results for $\chi(n, m, r)$. We divide our results into two theorems, corresponding to the relation between the array size and the number of items. In formulating the theorems, we use N for the number of items rather, to avoid confusion with the parameter n appearing in the Main Lemma (Lemma 9.4.1 below), which is used in the proof of the theorems, and in which n stands for the number of items inserted during a portion of the game.

The first theorem applies whenever $2N \leq m$, and gives interesting results provided that m is not too large (slightly superlinear function of N). In the first part of the theorem, the range $\{1, \ldots, r\}$ of possible items has size exponential in N. In the second part, r is at most a constant times m. Despite this strong limitation, the lower bound is only slightly worse.

Theorem 9.3.1. *There is a (sufficiently large) constant C_2 so that the following holds. Let m, N be integers satisfying $C_2 \leq N$ and $2N \leq m$. Let $\delta = N/m$. Then*

1. *If $r \geq N2^{N-1}$ then $\chi(N, m, r) \geq N \log^2(N) \frac{\delta}{C_2(\log(1/\delta))^2}$.*
2. *If $r \geq C_2 m$ then $\chi(N, m, r) \geq N \log^2(N) \frac{\delta^2}{C_2(\log(1/\delta))^2}$.*

Recall, the base of logarithms is 2.

In both parts, if $m = O(N)$ so that $\delta = O(1)$ then the lower bound obtained is $\Omega(N \log^2(N))$. The messy dependence on δ tells how our bound degrades in the case of $\delta = o(1)$, i.e., the array is much larger than the number of items. The first bound gives a nontrivial $\omega(n)$ bound for m up to $o(n \log^2(N)/\log^2(\log(N)))$, while the second bound is nontrivial for $m = o(N \log(N)/\log\log(N))$.

In the next result we consider array size satisfying $N < m < 2N$:

Theorem 9.3.2. *There are (sufficiently large) constants C_2, C_3 so that the following holds. Let m, N be integers satisfying $C_3 \leq N < m < 2N$ and let $\delta = N/m$. Assume $r \geq (\frac{1}{1-\delta})^{C_2} N$. Then:*

$$\chi(N, m, r) \geq \frac{1}{C_3} N \log^2(N) \log\left(\frac{1}{1-\delta}\right). \tag{9.1}$$

In this theorem, the array size is assumed to be at most twice the number of items. As long as $m \geq n(1+c)$ for a fixed $c > 0$ we again get an $\Omega(N \log^2(N))$ bound. As $m-n$ gets smaller, the bound improves. In particular, for $m \leq N + N^{1-\varepsilon}$ this gives a tight lower bound of $\Omega(N \log^3(N))$. For this lower bound we only need the range of items to be polynomial in m. (We believe that it is possible to refine the analysis to obtain an asymptotically similar lower bound when the range size is only $N + N^{1-O(\varepsilon)}$ but have not included this analysis so as not to lengthen an already lengthy proof.)

9.4 Reduction of the Theorems to the Main Lemma

As the game has been defined, every cell is initially unoccupied. For the proofs of the main theorems, it will be convenient to consider a generalization of the game, in which the array is initially partially full. This version of the game is specified by the parameters n, m (but not r) and additionally takes a set Y_0 of items, whose size is denoted by n_0 and is required to be at least 2. The subset Y_0 is given to the algorithm who selects the initial storage function $\mathbf{f}_0$ (at no cost). The game then proceeds as before, except that the adversary is restricted to inserting items in the range $(\min(Y_0), \max(Y_0))$ (where we assume $|Y_0| \geq 2$). During the game, items $y_1, y_2, \ldots, y_n$ are inserted and the algorithm defines allocations $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_n$. The set Y_t is defined as $Y_0 \cup \{y_1, \ldots, y_t\}$. Let us emphasize, that this is different to previous chapter.

Note that the parameter r does not appear in this formulation.

We denote the game by $G(n, m|Y_0)$ and write $\chi_{\mathcal{A}}(n, m|Y_0)$ for the minimum cost that can be achieved by the algorithm $\mathcal{A}$ against the best adversary. We assume that $m \geq n_0 + n$, otherwise there is not enough room to insert all of the

items. For a set Y of items, we define mingap(Y) to be the minimum absolute difference between pairs of items in Y.

Now we state the central result of Chapter 9:

Lemma 9.4.1. *(*The Main Lemma*) There are positive constants C_0, C_4 so that the following holds. Let m, n, n_0 be integers and let $\delta_0 = n_0/m$ where:*

$$C_0 \leq n \leq n_0 \tag{9.2}$$

$$n + n_0 \leq m \tag{9.3}$$

$$\delta_0 \in (\log^{-2}(n), 1 - n^{-1/5}). \tag{9.4}$$

Let Y_0 be any set of n_0 items and let $\mu_0 = \text{mingap}(Y_0)$.

1. *If $\mu_0 \geq 2^n$ then*

$$\chi(n, m|Y_0) \geq n(\log^2(n))\frac{\delta_0(1-\delta_0)}{C_4 \log^2(1/\delta_0)}.$$

2. *If $\mu_0 \geq 1 + 12/\delta_0$, then*

$$\chi(n, m|Y_0) \geq n(\log^2(n))\frac{{\delta_0}^2(1-\delta_0)}{C_4 \log^2(1/\delta_0)}.$$

We point out that the condition $n_0 \geq n$ means that the new items being added at most doubles the number of items in the array.

We now prove Theorems 9.3.1 and 9.3.2 using Lemma 9.4.1. In the proof of the first theorem we apply the main lemma once with $\delta_0 = 1/2$, and in the proof of the second theorem we'll apply the main lemma multiple times, with δ_0 getting closer and closer to 1 (which is why the dependence on δ_0 in the lower bound is important.) In every application of the main lemma, the array size parameter m of the lemma will be the same as the array size parameter m in the theorem being proved, but the number of items n in the lemma will vary and won't be the same as the number N in the theorem being proved.

Proof of Theorem 9.3.1. In the argument below we choose C_2 for the theorem large enough depending on C_4 in Lemma 9.4.1.

Given N, m, r for the theorem, let $n_0 = \lceil N/2 \rceil$ and $n = N - n_0$. Let B be the largest integer such that $n_0 B \leq r$. Let $Y_0 = \{B \cdot t : t \in \{1, \ldots, n_0\}\}$. Note that mingap($Y_0$) $= B$. Consider the adversary for $G(n, m, r)$ that during the first n_0 steps inserts Y_0 and then follows the optimal adversary strategy for the game $G(n, m|Y_0)$.

For the first part of Theorem 9.3.1, the hypothesis that $r \geq N2^{N-1}$ implies mingap(Y_0) $\geq 2^n$ so the first part of Lemma 9.4.1 applies. Under the hypothesis

of Theorem 9.3.1, $1 - \delta_0 \geq 1/2$, and so the conclusion of Lemma 9.4.1 yields the conclusion of the first part of Theorem 9.3.1.

For the second part of Theorem 9.3.1, the hypothesis $r \geq C_2 m$ and our freedom to choose C_2 to be a sufficiently large constant imply $B \geq \lfloor C_2 m/n_0 \rfloor \geq C_2/\delta_0 - 1 \geq 12/\delta_0 + 1$ and so part (2) of Lemma 9.4.1 gives the desired lower bound. □

We next turn to the proof of Theorem 9.3.2. The idea of the proof is simple: The array size m is not much larger than N. We bound the cost as the sum of the costs of $p = \Theta(\log(1/(1-\delta)))$ "subgames" where the first subgame starts when the array is (roughly) half full and consists of inserting the next (roughly) $m/4$ items. After subgame $i-1$ the number of empty spaces in the array is (roughly) $m/2^i$ and in the ith subgame we insert (roughly) $m/2^{i+1}$ additional items. Corollary 9.4.2 below (deduced from Lemma 9.4.1) says that that each phase has cost $\Omega(m \log^2(m))$; even though the number of items inserted per phase is decreasing by a factor of 2, this is counterbalanced by the increased crowding of the array.

The following consequence of Lemma 9.4.1 is what we need to analyze a single subgame.

Corollary 9.4.2. *There are positive constants C_0, C_5 so that the following holds. Let m, n be integers satisfying $C_0 \leq (m/2)^{5/6} \leq n \leq m/3$. Let Y_0 be any set of $m - 2n$ items such that $\mu_0 = \text{mingap}(Y_0) \geq 37$. Then:*

$$\chi(n, m|Y_0) \geq \frac{1}{C_5} m \log^2(m).$$

Furthermore, there is an adversary that achieves this bound and has the additional property that the mingap of the items in the array after inserting the n items is at least $\lfloor \mu_0/37 \rfloor$.

Proof. For the first conclusion, we apply the second part of the main lemma with $n_0 = m - 2n$ and $\delta_0 = n_0/m = 1 - \frac{2n}{m}$. We need to check the hypotheses of the lemma. Hypothesis (9.2) and (9.3) are immediate. Since $n \leq m/3$ and $m \leq 2n^{6/5}$ we have $1/3 \leq \delta_0 \leq 1 - n^{-1/5}$ as required for (9.3), and also that $\mu_0 \geq 37$ is at least $1 + 12/\delta_0$. Therefore the hypotheses of part 2 of the lemma hold, and from the conclusion we get:

$$\begin{aligned}
\chi(n, m|Y_0) &\geq n \log^2(n) \frac{\delta_0^2(1-\delta_0)}{C_4 \log^2(1/\delta_0)} \\
&\geq \frac{m(1-\delta_0)}{3} \log^2((m/3)^{5/6}) \frac{1}{4^2 \cdot C_4} \frac{1-\delta_0}{\log^2(1/\delta_{j-1})} \\
&\geq m \log^2(m) \cdot \frac{1}{C_5},
\end{aligned}$$

where the final inequality uses the (numerical) fact that for $\delta_0 \in \{1/4, \ldots, 1\}$, $\log(1/\delta_0) \leq 3(1-\delta_0)$, and the fact that C_5 can be chosen to be a large enough constant.

For the second conclusion, for ease of notation we use B to represent the number 37. We first the case that $r = Bn_0$ and Y_0 is the set $\{B, 2B, \ldots, n_0B\}$. Thus $\mu_0 = B$. By part 2 of Lemma 9.4.1 here is an adversary strategy Γ for inserting the next n items that forces the claimed lower bound on $\chi(n, m|Y_0)$. The mingap is at least 1, as required.

Now consider the general case that r is arbitrary and Y_0 is a set of size n_0 with mingap at least B. Let $v_1 < \cdots < v_{n_0}$ be the items in Y_0. Let $G = \lfloor \mu_0/B \rfloor$. For $1 \leq j < n_0$, let V_j be the set of items of the form $v_j + iG$ where $1 \leq i \leq B-1$ and let $V = V_1 \cup \cdots \cup V_{n_0-1}$. The smallest item of V_j is $v_j + G$ and the largest is at most at most $v_{j+1} - G$. Thus the set V contains exactly $B-1$ items between each pair v_j, v_{j+1} and so $V \cup Y_0$ is combinatorially equivalent to the above case that Y_0 is the set $\{B, 2B, \ldots, n_0B\}$. By the obvious adaptation of the strategy for that case, we can carry out the adversary strategy while only inserting items from V, so that at the conclusion of the game, the set of inserted items is a subset of $Y_0 \cup V$, and the mingap of this set is at least $G = \lfloor \mu_0/B \rfloor$. □

We now fill in the (routine) details of the above sketch of the proof of Theorem 9.3.2.

Proof of Theorem 9.3.2. Let m, N and δ be given as in the theorem. Suppose first that $\delta = N/m$ is bounded above by $1-c$ for some positive constant c. Then Theorem 9.3.2 follows from the second part of Theorem 9.3.1, since the quantity $\delta/\log(1/\delta)$ appearing in the conclusion of Theorem 9.3.1(2) can be bounded below by a positive constant and the quantity $\log(1/(1-\delta))$ appearing in the conclusion of Theorem 9.3.2(1) can be bounded above by a positive constant.

We are left with the (main) case that $\delta > 1-c$ for a constant $c > 0$ of our choice; for convenience we assume $c \leq 1/16$ which implies, in particular that $\log(1/(1-\delta)) \geq 4$. Let $\Delta = m - N$ and let p be the largest integer such that $2^p\Delta < m/2$, so:

$$p = \left\lfloor \log_2\left(\frac{m}{2}\Delta\right)\right\rfloor \geq \log\left(\frac{1}{1-\delta}\right) - 2 \geq \frac{1}{2}\log\left(\frac{1}{1-\delta}\right).$$

Define for $i \in \{0, \ldots, p\}$, $z_i = m - 2^{p-i}\Delta$ and let $n_i = 2^{p-i}\Delta = z_i - z_{i-1}$.

For the game of inserting N items into an empty array of size m, we identify p subgames where the subgrames involve disjoint sets of steps. Subgame i starts after z_{i-1} were inserted and consists of inserting the next n_i items, so it ends after a total of z_i items are inserted.

Each subgame satisfies the hypothesis of the corollary, and so we can bound the cost from below by $\Theta(m\log^2(m)) = \Theta(N\log^2(N))$. So the total cost can be

bounded below by this times $p = \Omega(\log(1/(1-\delta)))$. □

The remainder of the chapter is devoted to proving Lemma 9.4.1. Throughout the rest of the chapter, the input parameters to the lemma are fixed.

m**:** The array size.

Y_0**:** The set of initial items.

n_0**:** The size of Y_0.

μ_0**:** The mingap of Y_0.

δ_0**:** The initial density n_0/m.

n**:** The number of items to be inserted.

The rest of the chapter is organized as follows.

- Section 9.5 gives some additional notation.
- Section 9.6 gives a full description of the adversary. The adversary is not too difficult to describe but the choices made may strike the reader as somewhat arbitrary. The subsequent discussion will (we hope) demystify the adversary.
- In Section 9.7, we formulate seven (parameterized) properties of the adversary. We then state two main lemmas. Lemma 9.7.1 asserts that any adversary that satisfies these properties forces any algorithm to pay a high cost. Lemma 9.7.2 asserts that our adversary has these seven properties. We show how the main lemma follows easily from these two lemmas.
- In Section 9.9 we prove Lemma 9.7.1, showing that an adversary satisfying these seven properties give a good lower bound on any algorithm. We start with a sketch of the main idea of the proof and then follow with the full proof.
- In Section 9.10 we establish that our adversary has these seven properties by proving Lemma 9.7.2. We begin the section with an informal discussion of how these properties led us to the chosen adversary.

Lemma 9.4.1 was formulated in sufficient generality (in terms of n, n_0, m and μ_0, so as to be able to prove both Theorems 9.3.2 and 9.3.1. These include both cases that the range of possible numbers is very large, or rather small and also very dense case where $\delta_0 = n_0/m$ is very close to 1. The reader may wish to focus on the following restrictions:

- $n_0 = n = \Theta(m)$. (We'll refer to this as the case of *small constant density.*)
- $r \geq 2^m$. (We'll refer to this as the case of *large initial mingap.*)

We refer to this restricted case as the *illustrative case*. This setting of parameters is enough to prove the first part of Theorem 9.3.2 in the case that $N = \Theta(m)$ and captures most of the difficulty of the proof. In organizing the proof, we considered first presenting the proof for this case, and only then doing the general proof, but decided against it because it would require either duplicating much of the proof of the special case when doing the general proof, or leaving the reader to extrapolate the general proof based on the special case and a sketch of the differences. We'll provide some guideposts in the proof that will allow the reader to simplify details of the general proof in order to focus on the illustrative case.

9.5 Some Notation and Preliminaries for the Proof of the Main Lemma

We introduce and recall some terminology:

- A *segment* is a subinterval of the set of cells $\{1, \ldots, m\}$.
- A *step interval* is a subinterval of $\{0, \ldots, n\}$ representing a sequence of consecutive steps of the game.
- We say that I is an *items interval* of Y if

$$I = Y \cap \{\min(I), \ldots, \max(I)\}.$$

Recall that at step t, $\mathbf{Rel}_t$ denotes the set of items moved at step t. For $y \in \mathbf{Rel}_t$ the *trail of y at step t* is the segment $Trail_t(y)$ between $\mathbf{f}_{t-1}(y)$ and $\mathbf{f}_t(y)$; for y_t it is just the location $\mathbf{f}_t(y_t)$. The *busy region* at step t, denoted B_t is the union over $y \in \mathbf{Rel}_t$ of $Trail_t(y)$.

9.6 A Description of the Adversary for Lemma 9.4.1

In this section we present our adversary strategy. The first subsection discusses the preliminary notion of a gap, which is a pair of items that have been inserted such that no item between them has been inserted yet. The second subsection describes the class of *segment chain strategies* which includes our adversary strategy. The third subsection specifies our strategy within this class.

9.6.1 Gaps and suitable gaps

During each step t the adversary must choose an item y_t to insert into the array. For a set Y of items, a *Y-gap* is a pair $y^L < y^R$ of items belonging to Y such that no item of Y has value in the item interval (y^L, y^R). The *gap length* is $y^R - y^L$. We emphasize that a gap refers to the set of possible item values between y^L and y^R and not to the region of the array in which the items are stored.

Provided that a gap has length at least 2, there is always an item between y^L and y^R that is available to be inserted. We call such a gap *suitable*. A *suitable segment* is one that contains a suitable gap. The condition of being a suitable segment is equivalent to: the set of items currently stored in the segment is not a consecutive sequence of integers.

Our adversary will choose a suitable segment, identify the longest suitable gap (y^L, y^R) stored in the segment and select the item $\lfloor (y^L + y^R)/2 \rfloor$, which is the midpoint of the gap rounded down to the nearest integer. The segment (resp., gap) chosen by the adversary at step t is referred to as the *chosen segment (resp., gap) at step t*.

When the adversary selects the segment S, we must ensure that S contains a suitable gap. For $\text{mingap}(Y_0) \geq 2^n$ (as in the illustrative setting mentioned at the end of Section 9.4), an easy induction shows that $\text{mingap}(Y_t) \geq 2^{n-t}$ for every $t < n$. Therefore any segment S that contains at least two items is suitable. We refer to this case as *large initial mingap*. In the case of *small initial mingap*, $\text{mingap}(Y_0) < 2^n$, we need to be more careful to ensure that the selected segment is suitable. This will complicate things a bit, but not in any significant way. The reader may wish to focus on the case of large initial mingap; we will identify the places in the argument where the arguments diverge.

The choice of the suitable segment at step t will depend on the configuration $(Y_{t-1}, \mathbf{f}_{t-1})$. Intuitively, the adversary will select a suitable segment that is currently located in an area of the array that is relatively "crowded". A natural notion of crowding for a segment S is the ratio of the number of items stored in the segment to the length of the segment. This notion of crowding is sufficient for the case of large initial mingap.

To handle both the case of large and small initial mingap, we need a notion of crowding that depends on a *weight parameter* $\lambda \in (0, 1]$. Each item in the array is assigned a weight which is 1 if the item belongs to the initial set Y_0 and is λ if it was inserted by the adversary. Given a configuration (Y, f), we define the following functions on segments $S \subseteq \{1, \ldots, m\}$:

- The weight $w(S) = w(S, f)$ is the sum of the weights of all items stored in S under $\mathbf{f}$.
- The density $\rho(S) = \rho(S, f)$ is $w(S)/|S|$. The density function provides a

natural measure of crowding of S.

The weight and density with respect to allocation $\mathbf{f}_t$ are denoted w_t and ρ_t respectively.

In the case of large initial mingap ($\text{mingap}(Y_0) \geq 2^n$) we set the weight parameter λ to 1. Thus the weight of a segment is just the number of items stored in it and the density is the fraction of occupied cells. This is the only difference between this case and the case of small initial mingap.

The following lemma shows that by choosing λ appropriately we can get a sufficient condition for a segment to contain a suitable gap. Readers who wish to concentrate only on the case of large mingap can skip to the next subsection.

Lemma 9.6.1. *(Suitable Gap Lemma) Suppose that $\lambda \in (0, 1/2)$. Let Y be a set of integer items with a specified set Y_0 of* initial *item, let $S \subseteq \{1, \ldots, m\}$ be a segment and $\mathbf{f}$ an arbitrary allocation. Let A be the set of items from Y_0 stored in S and B be the set of other items stored in S with respect to $\mathbf{f}$. Define the weight $w(S)$ to be $|A| + \lambda|B|$ and its density $\rho(S)$ to be $w(S)/|S|$. If:*

1. *The mingap of Y_0 is at least $1 + \frac{1}{2\lambda}$*
2. *$w(S) \geq 2$*
3. *$\rho(S) \geq 2\lambda$.*

Then S is suitable, i.e. the set of items stored in S is not consecutive.

The key hypothesis is $\rho(S) \geq 2\lambda$. Since non-initial items are given weight only λ, an interval S of density at least 2λ must have a substantial fraction of initial items. Since the mingap of Y_0 is not too small, we will be able to show that there is a pair $x < y$ of items stored in S that form a gap in Y_0 that have fewer than $\mu(Y_0)$ items are stored between them in S, which implies that there is at least one item between them that has not been inserted yet, so S is suitable.

Proof. Let $a = |A|$ and $b = |B|$. We first note that $a \geq b\lambda$. This follows from $w(S) = a + b\lambda = (a + b)\rho(S) \geq 2(a + b)\lambda$ so $(1 - 2\lambda)a > b\lambda$ which implies $a > b\lambda$ In particular, this implies $a \geq 2$, since a is an integer larger than $(a + b\lambda)/2 = w(S)/2 \geq 1$.

Let $\min_A$ and $\max_A$ be the smallest and largest items in A. Suppose for contradiction that there is no suitable gap between $\min_A$ and $\max_A$. Then all of the $\max_A - \min_A + 1$ items in the range $\{\min_A, \ldots, \max_A\}$ must have been inserted already. There are $a - 1$ gaps between items of A, each of size at least $\mu(Y_0)$ so by the hypothesis on λ, $b \geq (a - 1)(\mu_0 - 1) \geq 2(a - 1)/\lambda \geq a/\lambda$ (since $a \geq 2$) which contradicts $a > b\lambda$. □

When we design our adversary strategy to prove Lemma 9.4.1 we will ensure that (for the case of small mingap) the segment selected by the algorithm satisfies the hypotheses of the suitable gap lemma. We haven't explained our strategy yet so we can't show this, but we sketch how this is done. We'll choose λ to be $\delta_0/6$. Then the hypothesis $\mu(Y_0) \geq 1 + 12/\delta_0$ from part 2 of Lemma 9.4.1 gives the needed lower bound on $\mu(Y_0)$ in this lemma. Also our strategy will always choose a segment of weight at least 2, and density at least $\delta_0/3 = 2\lambda$, so the hypotheses of the suitable gap lemma will be satisfied.

9.6.2 Segment chain strategies

This section describes a general class of adversary strategies from which we will select our adversary.

At each step t, our adversary will identify a segment satisfying the conditions of Lemma 9.6.1 (and other conditions as well.) The adversary will actually specify a chain of segments $S_t(1) \supset S_t(2) \supset \cdots \supset S_t(d)$[1], where the parameter d will be fixed (later) to be $\Theta(\log(n))$. (Note, that in general the first segment $S_t(1)$ need not equal $\{1, \ldots, m\}$.) The segment $S_t(d)$ will satisfy the conditions of Lemma 9.6.1 and will be used as the selected suitable segment at step t.

We call this type of strategy for the adversary, where we close in on a suitable segment by taking a chain of segments, a *segment chain strategy.*

It is natural that the segment chain selected at step t should relate to the sequence selected at the previous step $t-1$ and the moves made by the algorithm. Recall that B_t is the busy region at step t, which is determined by the moves made by the algorithm in response to y_t. Also recall that by Lemma 5.4.2 we assume that the algorithm is lazy so that B_t is always a segment. Define the *critical index* j_{t-1} after step $t-1$ to be the smallest index $j \in \{1, \ldots, d+1\}$ for which B_{t-1} is not a subset of $S_{t-1}(j)$. In particular, $j_{t-1} = d+1$ if B_{t-1} is a subset of $S_{t-1}(d)$, and $j_0 = 1$. Our adversary will satisfy the following natural rule (which was also satisfied by the adversary of Dietz et al.):

Conservative Selection Rule. For every $t \geq 2$: The sequence $S_t(1), \ldots, S_t(d)$ is chosen so that $S_t(j) = S_{t-1}(j)$ for $j < j_{t-1}$.

This rule puts no restriction on the selection of $S_t(j)$ for $j \geq j_{t-1}$; in particular the adversary can use any information about the present or past configurations.

Assuming the use of a segment chain strategy with conservative selection, we now summarize the sequence of events that occur during step t of the game. Note that the configuration $(Y_{t-1}, \mathbf{f}_{t-1})$, the busy segment B_{t-1}, and the critical index j_{t-1} were determined during step $t-1$.

[1]Recall, we use subscript to denote step and we use $(\cdot)$ notation to denote a particular coordinate of such a vector or sequence at that step.

- The adversary selects the sequence $S_t(1) \supset \cdots \supset S_t(d)$. (We will specify how this is done below.) The selection will be done subject to the conservative selection rule and will be done in a way that ensures that $S_t(d)$ satisfies the hypothesis of Lemma 9.6.1 (with respect to $(Y_{t-1}, \mathbf{f}_{t-1})$), and therefore contains a suitable gap.

- The adversary chooses the longest gap in $S_t(d)$ and lets y_t be the approximate midpoint. Y_t is set to be $Y_{t-1} \cup \{y_t\}$.

- The algorithm selects the storage function $\mathbf{f}_t$ for Y_t.

- The choice of $\mathbf{f}_t$ together with the previous storage function $\mathbf{f}_{t-1}$ determines the busy segment B_t.

- The critical index j_t is determined by B_t and $S_t(1), \ldots, S_t(d)$.

9.6.3 Specifying the segment chain

It remains to describe how the adversary selects the segments $S_t(j)$ for $j \geq j_{t-1}$. Our strategy is not hard to describe but it is not so easy to motivate the (seemingly arbitrary) choices made. We'll present it first with a minimum of discussion. Hopefully the discussion in Section 9.10.1 and the proofs of correctness will demystify it.

Our strategy uses an auxiliary sequence of segments $T_t(1), \ldots, T_t(d)$ which is interleaved with $S_t(1), \ldots, S_t(d)$:

$$T_t(1) \supset S_t(1) \supset T_t(2) \supset S_t(2) \supset \cdots \supset T_t(d) \supset S_t(d).$$

Recall that j_{t-1} is the critical index after step $t-1$. For $1 \leq j < j_{t-1}$, we satisfy the conservative selection rule by setting $T_t(j) = T_{t-1}(j)$ and $S_t(j) = S_{t-1}(j)$.

The significant part of the specification consists of three parts:

- The choice of $T_t(j_{t-1})$ for the critical index j_{t-1}.

- For $i \in \{j_{t-1}, \ldots, d\}$, the choice of $S_t(i)$ given $T_t(i)$.

- For $i \in \{1 + j_{t-1}, \ldots, d\}$, the choice of $T_t(i)$ given $S_t(i-1)$.

To specify these, we will need some definitions. For a segment U:

- **middle**(U) is the subsegment of U defined as follows: Break U into three segments from left to right, L, M, R where $|L| = |R| = \lfloor |U|/3 \rfloor$. **middle**$(U)$ is the segment M (which is roughly the middle third of U).

Let (Y, f) be an arbitrary configuration with associated density ρ. Let $\kappa > 0$. For an arbitrary segment U:

- The *quality of* U *with respect to* (Y, f) is the real number $\phi(U) = \rho(U)^{1/\kappa}|U|$.
- **densify**(U) is the subsegment of V of U that maximizes $\phi(V)$ (breaking ties arbitrarily).
- **balance**(U) is the subsegment V of U given by **middle**(**densify**(U)).
- we write ρ_{t-1}, ϕ_{t-1}, $\mathbf{densify}_{t-1}$ and $\mathbf{balance}_{t-1}$ to be the functions based on the configuration $(Y_{t-1}, \mathbf{f}_{t-1})$

Our adversary depends on three parameters: the length d of the segment chain will be fixed to be $\Theta(\log(n))$ by (9.8) and the parameter κ used in the quality function (which will be fixed to be $\Theta(1/\log(n))$ in (9.7)) and the weight parameter λ which appears implicitly because it determines the functions w_t, ρ_t, and ϕ_t and therefore also the functions **densify** and **balance** defined above. The parameter λ is 1 in the case of large mingap and will be set to $\delta_0/6$ otherwise. We'll explain these choices later but for now we leave d, κ and λ as parameters.

The specification of adversary $\textsc{Adv}(d, \kappa, \lambda)$

Adv1 Specification of $T_t(j)$ for the critical index j_{t-1}:

Adv1.a If $j_{t-1} = 1$ then $T_t(j_{t-1})$ is the segment of length between $n/2$ and n of highest density (breaking ties by an arbitrary rule).

Adv1.b If $j_{t-1} > 1$ (and so necessarily $t > 1$) set

$$T_t(j_{t-1}) = T_{t-1}(j_{t-1}) \cup B_{t-1}.$$

Adv2 Specification of $S_t(i)$ given $T_t(i)$ for $i > j_{t-1}$: $S_t(i) = \mathbf{balance}_{t-1}(T_t(i))$.

Adv3 Specification of $T_t(i)$ given $S_t(i-1)$ for $i > j_{t-1}$: $T_t(i) = \mathbf{middle}(S_t(i-1))$.

It is not clear that the above adversary is well-defined, because we need that the final segment $S_t(d)$ in each segment chain be suitable as defined in Section 9.6.1; if it isn't then the adversary is unable to proceed with inserting an item according to the requirements. Indeed if d is chosen too large then the chain of segments will eventually degenerate to a segment of length 1 which will just be repeated until the chain ends. When we fix the parameters, one of the things we'll have to prove is that each of the segments $S_t(d)$ is indeed suitable (which is formulated as Property (P4) below).

Also, in our adversary we require that for all t and j, $T_t(j)$ and $S_t(j)$ are segments. This is not immediately apparent from the description of the adversary (because of Adv1.b) but can be proved by induction on t and for fixed t by induction on j. The only case that requires some discussion is the definition of $T_t(j)$ when j is critical and $j > 1$. By induction, $T_{t-1}(j)$ is a segment, and by laziness of the algorithm B_{t-1} is a segment that must intersect $T_{t-1}(j)$, so their union is a segment.

9.7 Important Properties of the Adversary

The rest of this chapter is devoted to proving that the above adversary, with suitably chosen parameters, gives the bounds of Lemma 9.4.1. In this subsection we state seven properties (P1) - (P7) that encapsulate what we need from our adversary. In subsequent sections we'll show that these properties imply a cost lower bound, and that our adversary satisfies these properties.

The first three properties are fairly mild technical "boundary" conditions on the sizes and densities of the segments appearing in a segment chain. We introduce parameters parameters σ and δ^* to represent these conditions.

(P1) For each segment chain the first segment in the chain (and therefore all others) have size at most $n/2$.

(P2(σ)) For each segment chain, the final segment (and therefore all others) have size at least σ. (The reader should think of σ as a function of n that grows slowly, but not too slowly. Later, we'll choose it to be $n^{1/4}$, but the exponent $1/4$ is fairly arbitrary.)

(P3(δ^*)) Every segment in any of the segment chains has density at least δ^*. (Later we set $\delta^* = \delta_0 2^{\delta_0 - 1}$ which is between $\delta_0/2$ and δ_0.) In the illustrative case in which $\delta_0 < 1/2$ the reader should think of δ^* as $c\delta^*$ for some constant c around $1/2$ where the constant is unimportant. (In the dense case that δ_0 is close to 1, the desired lower bound of $\Theta(n \log^2 n/(1-\delta_0))$ gets larger as δ_0 gets closer to 1. In this case it is not enough to take the δ^* to be a constant fraction of δ_0, instead it is roughly $(\delta_0)^2$.)

The next property is crucial since without it the adversary is not well-defined. However, it will be easy to satisfy.

(P4) For each t, $S_t(d)$ has a suitable gap. In the case of large mingap we only need $|S_t(d)| \geq 2$ (which will follow from (P2(σ))), but for the case of small mingap, we'll need to make sure that $S_t(d)$ satisfies the hypotheses of Lemma 9.6.1 (which will not be hard to do.)

The next simple property plays a significant role in the lower bound.

(P5) For each t and $i \geq 2$, $|S_t(i)| \leq |S_t(i-1)|/2$. (Segment sizes decrease by at least a factor of 2 along a segment chain.)

The next property is especially important to the argument. As discussed above, when we choose the segment chain we would like that (among other properties) each successive segment should have density at least that of the previous segment. We can't necessarily do this but it's enough that the density not decrease by too much. This is quantified by the following property, which depends on a *density degradation parameter* α:

(P6(α)) $\rho_{t-1}(S_t(1)) \geq \frac{1}{2^\alpha}\delta_0$ and for $i \geq 2$, $\rho_{t-1}(S_t(i)) \geq \frac{1}{2^\alpha}\rho_{t-1}(S_t(i-1))$.

The value of α we're able to achieve plays a crucial role in our lower bound. The heart of the lower bound argument (Lemma 9.7.1) will give a lower bound on $\chi(n, m|Y_0)$ of roughly $\Omega(nd^2/(\alpha d + 1))$. We will have $d = \Theta(\log(n))$ (it can't be larger because of (P5)) so this argument has the potential of giving a lower bound of $\Omega(n \log^2 n)$ if we can make α small enough. The argument of Dietz et al. [13] constructs the segment chain with $\alpha = O(1)$ which gives an $\Omega(n \log(n))$ lower bound; we'll be able to achieve $\alpha = O(1/\log(n))$ which will give the optimal $\Omega(n \log^2 n)$ lower bound.

The final property (P7) is the most technical, and is crucial for the analysis. It concerns the way that we'll account for the cost of the algorithm, and describing this property requires some additional terminology.

Each of the segment chains $S_t(1), \ldots, S_t(d)$ has length d. We say that the segment $S_t(j)$ is at *level* j. For each $j \in \{1, \ldots, d\}$, we define $\Sigma(j) \subseteq \{1, \ldots, n\}$ to be the set of steps $t \in \{1, \ldots, n\}$ such that B_t is not a subset of $S_t(j)$. According to the definition of the adversary, $\Sigma(i)$ is therefore the set of steps t such that $S_i(t)$ is not determined by the conservative selection rule, but rather is rebuilt. We use the set $\Sigma(j)$ to define a partition $\Pi(j)$ of $\{1, \ldots, n\}$ into intervals where the first interval starts at 1 and ends at the smallest element of $\Sigma(j)$ and each successive interval ends at the next smallest element of $\Sigma(j) \cup \{n\}$. These intervals are called the *epochs* at level j. We also define the partition $\Pi(0)$ to be the trivial partition consisting of the single epoch $\{1, \ldots, n\}$, and $\Pi(d+1)$ to be the partition into n singleton sets. We make a few observations:

- By the conservative selection rule, for each epoch E, all of the segments $S_t(j)$ are the same for all $t \in E$. We therefore define S_E to be this unique segment. For the epoch $\{1, \ldots, n\}$ at level 0 we define $S_{\{1,\ldots,n\}} = \{1, \ldots, m\}$, and for the singleton epochs at level $d+1$, S_E is not defined.

- The definition of $\Sigma(j)$ implies that $\Sigma(j+1) \subseteq \Sigma(j)$ and therefore the partition of $\{1, \ldots, n\}$ into epochs at level $j+1$ refines the partition into epochs at level j.

Each epoch E is an interval $\{s_E, \ldots, c_E\}$ where s_E is the *start time* of the epoch and c_E is the *closing time*. The closing times of the epochs at level j are the elements of $\Sigma(j) \cup \{n\}$ and the start times are each 1 more than the closing time of the previous epoch. We write $E_t(j)$ for the epoch at level j that contains step t.

An epoch is said to be *terminal* if it contains n, so it is the final epoch at its level. An epoch is *non-terminal* otherwise.

We now build a rooted tree, called the *epoch tree* whose nodes are the epochs at all levels together with one leaf for each $t \in \{1, \ldots, n\}$:

- The root is the level 0 epoch $\{1, \ldots, n\}$.
- For an epoch at level $j \geq 1$, its parent is the unique epoch at level $j-1$ that contains it.

Thus the tree has depth $d+1$ and it has n leaves corresponding to singleton subsets. We visualize the tree as ordered so the leaves are in order from left to right.

Observe that for each $t \in \{1, \ldots, n\}$ the path from the root $\{1, \ldots, n\}$ to the leaf $\{t\}$ traverses the sequence $E_t(1), \ldots E_t(d)$ of epochs and this sequence corresponds precisely to the sequence $S_t(1), \ldots, S_t(d)$ of segments selected by the algorithm at step t.

The epoch tree will provide a convenient way to account for the cost of relocations done by a given algorithm. Fix a segment chain strategy and an algorithm. Let χ denote the cost of the algorithm against that strategy.

We define a *move* to be a pair (y, t) where y is an item and t a step such that the algorithm moves item y at step t. The cost χ incurred by the algorithm is the total number of moves.

For accounting purposes, we assign each move (y, t) to the smallest epoch E such that $t \in E$ and $\mathbf{f}_{t-1}(y) \in S_E$. In terms of the epoch tree, we travel from the leaf t to the root and assign (y, t) to the first epoch encountered on the path for which S_E contains $\mathbf{f}_{t-1}(y)$. (By definition (y, t) is not assigned to a leaf.) Thus if (y, t) is assigned to epoch E at level i then $\mathbf{f}_{t-1}(y) \in S_t(i) - S_t(i+1)$. Denoting the cost of all moves assigned to E by q_E we have:

$$\chi = \sum_E q_E. \tag{9.5}$$

We are now ready to state the final property of the adversary.

(P7) For any non-terminal epoch E with start time s we have $q_E \geq \frac{1}{8} w_{s-1}(S_E)$, that is, the total cost of moves assigned to E is at least a $1/8$ fraction of the weight of the associated segment S_E at the start of the epoch.

We now formulate two lemmas concerning these properties. The first lemma gives a lower bound on the cost incurred by any algorithm against a segment chain strategy that satisfies the above properties, in terms of the parameters σ, α and λ in the properties. This lemma encapsulates and extends the main accounting argument of Dietz et al. [12, 21], which they used to prove an $\Omega(\log^2(n))$ amortized lower bound for the special case of *smooth* algorithms.

Lemma 9.7.1. *(*Properties imply lower bound*) Let m, n, n_0, δ_0 and Y_0 be as in Lemma 9.4.1. Let $\sigma \geq 1$ and α, λ be positive parameters. If a segment chain strategy produces a segment chain with d levels satisfying (P1)-(P7) then the cost incurred by the algorithm satisfies*

$$\chi\left(n, m | Y_0\right) \geq \frac{\lambda \delta^* n d^2}{128(\alpha d + \frac{\lambda}{\sigma} + 1 - \delta_0)}. \tag{9.6}$$

We next turn to the second lemma, Lemma 9.7.2, which shows that with suitable parameters our adversary satisfies (P1)-(P7). Before giving the specific parameter choices, we give some intuition for these choices. The choice of parameters is directly motivated by the expression in the lower bound. We would like the numerator to be large, while keeping the denominator small.

The main parameter in the numerator is the depth of the segment chains d. Property (P5), which requires segment sizes to at least halve each time, and property (P4) which requires that each $S_t(d)$ be suitable, limit d to be $O(\log(n))$. We will indeed be able to choose d to be $\Theta(\log(n))$ (here and elsewhere in this overview the $\Theta(\cdot)$ may have an implicit dependence on δ_0). The numerator also involves λ and δ^* which are both at most δ_0 and we'll pick them to be $\Omega(\delta_0)$. Thus the numerator will just be $\Theta(n \log^2(n))$ (where we again hide the dependence on δ_0).

The denominator must be at least $1 - \delta_0$ and we'll be able to achieve a bound of $\Theta(1 - \delta_0)$. To do this we'll need that the parameter α which bounds the density degradation along each chain by $O((1 - \delta_0)/d)$ so (ignoring the dependence on δ_0) we need α to be $O(\frac{1}{\log(n)})$. We'll also need that $\lambda/\sigma = O(1 - \delta_0)$ but ensuring it is a minor detail.

The density degradation parameter α is the crucial part of the denominator. The parameter α is closely related to the input parameter κ of the adversary (which determines the quality function). As we'll see later in Lemma 9.11.3, α is in fact $O(\kappa)$. Since we want α to be $O(\frac{1}{\log(n)})$ we'll want to choose κ to be $O(\frac{1}{\log(n)})$.

On the other hand, if we make κ too small then this has the effect of making the quality function depend mostly on the density and very little on the size. This

can be bad because, as the adversary builds a segment chain the segment sizes may shrink rapidly along the chain, which could force us to make the length d of the segment chains smaller than $\Theta(\log(n))$ in order to be sure that the final segment in each chain is suitable (as required by property (P4)).

Fortunately, as we'll show, we will be able to choose κ (and hence also α) to be $\Theta(1/\log(n))$ and d to be $\Theta(\log(n))$. Thus we'll get the $\Theta(n\log^2(n))$ lower bound.

With all of this in mind, we now specify the parameters.

We choose the input parameters to our adversary as follows:

$$\kappa = 2\log(1/\delta_0)\frac{1}{\log(n)} \tag{9.7}$$

$$d = \left\lfloor \frac{1-\delta_0}{8C_6\log(1/\delta_0)}\log(n) \right\rfloor, \tag{9.8}$$

$$\lambda = \frac{1}{6}\delta_0. \tag{9.9}$$

where

$$C_6 = 60. \tag{9.10}$$

We'll also need a lower bound on n:

$$C_0 = 2^{1000C_6} \tag{9.11}$$

(Both C_6 and C_0 are chosen large enough to satisfy Lemma 9.11.3 below.)

The auxiliary parameters needed to specify properties (P2),(P3) and (P6) are set as follows:

$$\sigma = n^{1/4} \quad \text{(segment size lower bound)} \tag{9.12}$$

$$\delta^* = \delta_0 2^{\delta_0 - 1} \quad \text{(segment density lower bound)} \tag{9.13}$$

$$\alpha = 2C_6\kappa = 4C_6\log(1/\delta_0)\frac{1}{\log(n)} \quad \text{(density degradation parameter)} \tag{9.14}$$

Here is the promised lemma:

Lemma 9.7.2. *Let $m, n, n_0, Y_0, \delta_0, \mu_0$ be as in Lemma 9.4.1. Let the parameters be set according to (9.7)-(9.14). Then* $\textsc{Adv}(d, \kappa, \lambda)$ *satisfies (P1)-(P7).*

9.8 Proof of Lemma 9.4.1

Before proving lemmas 9.7.2 and 9.7.1, we show how they combine to prove the main lemma.

Let m, n, n_0, Y_0, δ_0 be as in Lemma 9.4.1. Lemma 9.7.2 implies that, with the given setting of parameters, our adversary satisfies (P1)-(P7).

Lemma 9.7.1 gives a lower bound on $\chi(n, m|Y_0)$. The denominator of (9.6) is $\Theta(\alpha d + \frac{\lambda}{\sigma} + 1 - \delta_0)$. The settings given by (9.14) and (9.8) give $\alpha d \leq (1 - \delta_0)/2$. The setting $\sigma = n^{1/4}$ and $\lambda \leq 1$ and the hypothesis of the main lemma that $\delta_0 \leq 1 - n^{-1/5}$ give $\frac{\lambda}{\sigma} \leq 1 - \delta_0$. So the denominator of (9.6) is $\Theta(1 - \delta_0)$

For the numerator, the setting of d gives $d^2 = \Theta(\log^2(n))\lambda\delta^*(1-\delta_0)^2/\log^2(1/\delta_0)$. For large mingap $\lambda = 1$ and the fraction simplifies to:

$$\chi = \Theta\left(n \log^2(n) \frac{\delta_0(1 - \delta_0)}{\log^2(1/\delta_0)}\right),$$

while for small mingap $\lambda > \frac{\delta_0}{2e}$,

$$\chi = \Theta\left(n \log^2(n) \frac{{\delta_0}^2(1 - \delta_0)}{\log^2(1/\delta_0)}\right),$$

as required to prove Lemma 9.7.1.

It remains to prove Lemmas 9.7.1 and 9.7.2. Each of these lemmas is proved in its own section, and the two sections are completely independent so can be read in either order.

9.9 Proof of Lemma 9.7.1

The proof that the properties give a good lower bound is a careful accounting argument that is a reworking of the idea used in [12, 21] to obtain an $\Omega(n \log^2(n))$ bound for smooth algorithms.

9.9.1 Some preliminaries and an overview

For simplicity we write χ for $\chi(n, m|Y_0)$. By (9.5), $\chi \geq \sum_E q_E$. The critical property (P7) says that, for E non-terminal, q_E is bounded below by $\frac{1}{8}$ of the weight of the associated segment S_E at the beginning of the epoch. Using this together with the lower bound on the density of any segment from (P3(δ^*)) gives:

$$qE \geq \frac{1}{8} w_{s_E - 1}(S_E) \geq \frac{\delta^*}{8} |S_E|.$$

Letting $\mathcal{N}$ be the set of non-terminal epochs at level between 1 and d, we have:

$$\chi \geq \sum_{E \in \mathcal{N}} q_E \geq \frac{\delta^*}{8} \sum_{E \in \mathcal{N}} |S_E|. \tag{9.15}$$

So we are reduced to proving a lower bound on $\sum_{E \in \mathcal{N}} |S_E|$. To this end we start with two easy observations.

Proposition 9.9.1.

1. *For any epoch E, $|E| \leq |S_E|$.*
2. *For each level $i \in \{1, \ldots, d\}$ the sum of the lengths of all non-terminal epochs at level i is at least $n/2$.*

The first observation holds because during epoch E, $|E|$ new items are stored within the segment $|S_E|$. The second holds since the sum of the lengths of all epochs at level i is n and by (P1) the terminal epoch has length at most $n/2$.

Combining these observations with (9.15),

$$\chi \geq \frac{\delta^*}{8} \sum_{E \in \mathcal{N}} |S_E| \geq \frac{\delta^*}{8} \sum_{E \in \mathcal{N}} |E| \geq \frac{\delta^* nd}{16}.$$

This is a non-trivial lower bound, but we want a lower bound of $\Omega(nd^2)$. To improve the bound from $\Omega(nd)$ to $\Omega(nd^2)$ we try to show that the bound $|S_E| \geq |E|$ used above can typically be improved to $|S_E| = \Omega(|E|d)$. This is not universally true for all epochs, but the following weaker statement will suffice: for a constant fraction of steps t, $|S_E| = \Omega(|E|d)$ holds for a constant fraction of the epochs containing t. This would follow if we could show that for a constant fraction of steps t, $\sum_{E : t \in E} \frac{|E|}{|S_E|} = O(1)$.

The following proposition shows that something like this holds if we replace $|E|$ in the numerator by a related quantity. For epoch E at level at least 1, let $\pi(E)$ denote the parent of epoch E in the tree, which is the epoch containing E whose level is one less than that of E. Let:

$$\Delta_E = s_E - s_{\pi(E)},$$

which is the time from the start of $\pi(E)$ until the start of E.

Proposition 9.9.2. *For any time t,*

$$\sum_{E \neq \{1, \ldots, n\} : t \in E} \frac{\Delta_E}{|S_{\pi(E)}|} \leq \frac{1}{\lambda}(1 - \delta_0 + d\alpha).$$

Proof. For $i \in \{0, \ldots, d\}$ we make the following definitions:

- $E(i)$ denotes the epoch at level i containing t.
- $s(i)$ is the start time of $E(i)$. Observe that $s(0) = 1 \leq s(1) \leq \cdots \leq s(d+1) = t$.
- $S(i)$ is the segment associated $E(i)$.
- $\rho(i) = \rho_{s(i)-1}(S(i))$, which is the density of the segment $S(i)$ just prior to the start of epoch $E(i)$. Note that $\rho(0) = \rho_0\{1, \ldots, n\} = \delta_0$. This definition doesn't work for $\rho(d+1)$ (since there is no segment $S(d+1)$) so we define $\rho(d+1) = 1$ for convenience.
- $\Delta(i) = \Delta_{E(i)}$ which is equal to $s(i) - s(i-1)$.

For any $i \in \{1, \ldots, d\}$, the step interval $\{s(i-1), \ldots, s(i)-1\}$ has size $\Delta(i)$ and is a subset of $E(i-1)$. During this interval of steps all inserted items are placed in $S(i-1)$ and no item leaves $S(i-1)$. Hence the weight of $S(i-1)$ increases by $\lambda\Delta(i)$ and so:

$$\rho_{s(i)-1}(S(i-1)) - \rho(i-1) = \rho_{s(i)-1}(S(i-1)) - \rho_{s(i-1)-1}(S(i-1)) = \lambda\frac{\Delta(i)}{|S(i-1)|}.$$

We also have:

$$\rho(i) \geq \rho_{s(i)-1}(S(i-1))2^{-\alpha} \geq \rho_{s(i)-1}(S(i-1)) - \alpha.$$

For $i \leq d$ the first inequality holds by property (P6(α)) (which bounds the rate at which density degrades along a segment chain) and for $i = d+1$ it holds from the choice we made that $\rho(d+1) = 1$. The second inequality holds because $\rho(i)$ and α are in $\{0, \ldots, 1\}$.

Using the second inequality with the first and rearranging we get:

$$\frac{\Delta(i)}{|S(i-1)|} \leq \frac{1}{\lambda}\left(\rho(i) - \rho(i-1) + \alpha\right).$$

Summing this inequality for each $E(i)$, the sum on the right telescopes to give:

$$\sum_{E \neq \{1,\ldots,n\}: t \in E} \frac{\Delta_E}{|S_{\pi(E)}|} \leq \frac{1}{\lambda}(\rho(d+1) - \rho(0) + d\alpha) \leq \frac{1}{\lambda}(1 - \delta_0 + d\alpha),$$

as required. □

Intuitively, this proposition is useful because we expect that for a "typical" epoch E, $\Delta_E = \Omega(|\pi(E)|)$, i.e. the number of steps from the start of $\pi(E)$ to the start of E is typically a constant fraction of the length of $|\pi(E)|$. The technical work done in the proof makes this intuition precise.

9.9.2 The proof

Following the discussion of the previous subsection, we return to (9.15) and try to show that $\sum_{E\in\mathcal{N}} |S_E|$ is a $\Theta(d)$ factor larger than $W = \sum_{E\in\mathcal{H}} |E|$. To facilitate this comparison we define $\beta(E) = |E|/W$. Using the arithmetic-harmonic mean inequality (which says that if X is a positive valued random variable then $\mathbb{E}[X] \geq 1/\mathbb{E}[1/X]$) we get:

$$\sum_{E\in\mathcal{N}} |S_E| = W \sum_{E\in\mathcal{N}} \beta(E)\frac{|S_E|}{|E|} = \frac{W}{\sum_{E\in\mathcal{N}} \beta(E)\frac{|E|}{|S_E|}} = \frac{W^2}{\sum_{E\in\mathcal{N}} \frac{|E|^2}{|S_E|}} \geq \frac{n^2d^2}{4\sum_{E\in\mathcal{N}} \frac{|E|^2}{|S_E|}}.$$

If we can show that the denominator is $O(n)$ (where the big O does not depend on d) then we'll get the desired lower bound. This is indeed true, and the conclusion of the lemma follows immediately by combining the previous inequality with (9.15) and the following lemma:

Lemma 9.9.3.

$$\sum_{E\in\mathcal{N}} \frac{|E|^2}{|S_E|} \leq \frac{4n}{\lambda}\left(\frac{\lambda}{\sigma} + 1 - \delta_0 + d\alpha\right).$$

The reader should take note that we have switched from proving a lower bound on a sum to proving an upper bound on a related sum.

Proof. We prove the upper bound with the sum extended to the set $\mathcal{E}$ of all epochs at levels from 1 to d (not just non-terminal epochs).

We will prove the bound by rewriting the sum using some elementary accounting tricks. Here's the first one. Recall that for a positive integer k, k^2 is the sum of the first k odd numbers. Thus, letting s_E be the start time of epoch E we can write:

$$|E|^2 = \sum_{t\in E} 1 + 2(t - s_E).$$

We therefore have:

$$\sum_{E\in\mathcal{E}} \frac{|E|^2}{|S_E|} = \sum_{t=1}^{n} \sum_{E\in\mathcal{E}:t\in E} \frac{1 + 2(t - s_E)}{|S_E|}. \tag{9.16}$$

We hold t fixed, and bound the inner sum (and then multiply it by n). Let $\mathcal{E}(t)$ be the set of epochs at level 1 to d containing t. Let $\mathcal{H}(t)$ be the epochs at level 1 to $d+1$ containing t (i.e. including the leaf $\{t\}$).

Recalling the definition of Δ_F for an epoch F from the previous subsection, we have that $t - s_E = \sum_{F \in \mathcal{H}(t): F \subset E} \Delta_F$, where $F \subset E$ is strict containment. Therefore:

$$\begin{aligned}\sum_{E \in \mathcal{E}: t \in E} \frac{1 + 2(t - s_E)}{|S_E|} &= \sum_{E \in \mathcal{E}(t)} \frac{1}{|S_E|} + 2 \sum_{E \in \mathcal{E}(t)} \sum_{F \in \mathcal{H}(t): F \subset E} \frac{\Delta_F}{|S_E|} \\ &= \sum_{E \in \mathcal{E}(t)} \frac{1}{|S_E|} + 2 \sum_{F \in \mathcal{H}(t)} \Delta_F \sum_{E \in \mathcal{E}(t): F \subset E} \frac{1}{|S_E|} \quad (9.17)\end{aligned}$$

Now we note that property (P5) (that the segment sizes decrease by at least a factor of 2 down the epoch tree) implies that for any epoch $D \in \mathcal{E}$:

$$\sum_{E: D \subset E} \frac{1}{|S_E|} \leq \frac{2}{|S_{\pi(D)}|} \leq \frac{1}{|S_D|}. \quad (9.18)$$

We can use this to bound the first sum in (9.17) by taking D to be the epoch at level d. By property (P2(σ)), $|S_D| \geq \sigma$. For the second sum in (9.17) we take D to be F and use the second inequality in (9.18) to get:

$$\sum_{E \in \mathcal{E}: t \in E} \frac{1 + 2(t - s_E)}{|S_E|} \leq \frac{2}{\sigma} + 4 \left(\sum_{F \in \mathcal{H}(t)} \frac{\Delta_F}{|S_{\pi(F)}|} \right)$$

We can now apply Proposition 9.9.2 to obtain

$$\begin{aligned}\sum_{E \in \mathcal{E}: t \in E} \frac{1 + 2(t - s_E)}{|S_E|} &\leq 4 \left(\frac{1}{\sigma} + \frac{1 - \delta_0 + d\alpha)}{\lambda} \right) \\ &= \frac{4}{\lambda} \left(\frac{\lambda}{\sigma} + 1 - \delta_0 + d\alpha \right).\end{aligned}$$

Summing over all $t \in [n]$ and using (9.16) gives the desired bound. □

This completes the proof of Lemma 9.7.1.

9.10 Proof of Lemma 9.7.2

In this final section we show that the given strategy satisfies properties (P1)-(P7). We start with a detailed informal overview of the ideas of the adversary construction.

9.10.1 Motivating our adversary strategy

Our adversary has a reasonably short description, but the specific choices made in the definition may seem arbitrary:

- Why do we introduce the auxiliary sets $T_t(i)$?
- Why do we choose the particular quality function?
- If $i > j$ then it follows that $S_t(i) = \textbf{middle}(\textbf{densify}_{t-1}(\textbf{middle}(S_t(i-1))))$ from the above. Why do we need the two applications of **middle**?

These issues were introduced at a high level in Section 9.2. Here we consider them again in more technical detail, and show how we arrived at the specific choices. The proof in the later subsections can in principle be read without reading this section, but the technicalities involved in the proof hide the main ideas, which we discuss in this subsection. To simplify the discussion in this section we restrict to the case of large initial mingap. For this case the weight of a segment is the number of items, and the density is the fraction of occupied locations, and a segment is suitable (as defined earlier) if and only if there are at least two items stored in it.

As remarked following the statement of Lemma 9.7.1, when we apply the lemma to get a lower bound of $\Omega(n \log^2(n))$, we want the depth of the chain to be $d = \Theta(\log(n))$, and the density degradation parameter $\alpha = O(1/\log(n))$. The reader should keep these parameters in mind.

9.10.2 Satisfying (P1)-(P6)

Achieving all of the desired properties except (P7) with the desired parameter values is straightforward. In discussing these properties, we'll reason inductively. We start by proving the properties for the first segment chain. Then for step $t > 1$ we prove the properties for the tth segment chain assuming that they hold for the $t-1$st chain. When considering the tth chain we reason about the segments by increasing level. A natural attempt for selecting the first step is to take $S_1(1)$ to be the segment of size $\lfloor n/2 \rfloor$ having highest density, and for $i \geq 1$, define $S_1(i)$ to be the subsegment of $S_1(i-1)$ of size $\lfloor |S_1(i-1)|/2 \rfloor$ having highest density. This does not quite work because it might not satisfy (P6(α)); for a segment S, all subintervals of some fixed size k might have density less than, say $0.9\rho(S)$. This difficulty disappears if we relax the requirement that S have size exactly k: it is easy to show that for a segment S of size at least 8 and $k \leq |S|/2$, there must be a subsegment of size between $k/2$ and k having density at least that of S.

So for a given configuration and segment S consider the subsegment of size between $|S|/4$ and $|S|/2$ having maximum density (breaking ties arbitrarily). We'll refer to this as the *densest large subsegment* of S.

This suggests we choose $S_1(1)$ to be the densest large subsegment of $\{1, \ldots, n\}$ and for $i \geq 2$ choose $S_1(i)$ to be the densest large subsegment of $S_1(i-1)$. The resulting sequence has nondecreasing density (so certainly satisfies (P3(δ^*) and (P6(α))) and as the segment sizes decrease by at most a factor of 4, we can continue it for $\Theta(\log(n))$ levels and have conditions (P1)-(P6) hold for the first segment chain.

Next let's consider the case $t > 1$. Assume that we've already selected the segment chain for step $t-1$ so that conditions (P1)-(P6) hold. We want to build the chain for step t so that these continue to hold. Let j_{t-1} be the critical index for step $t-1$, as defined earlier, i.e., the first level for which B_{t-1} is not a subset of $S_{t-1}(j)$ (or $d+1$ if there is no such level). For $i < j_{t-1}$ the segment chain restriction requires $S_t(i) = S_{t-1}(i)$ and it is easy to check that properties (P1)-(P6) are preserved, because after the move by the algorithm the set of items stored in $S_{t-1}(i)$ changes only by the addition of the newly inserted item y_{t-1}. For levels $i \geq j_{t-1}$, it is not in general sufficient to take $S_t(i) = S_{t-1}(i)$ because the algorithm may have moved many items out of $S_{t-1}(i)$ and so the density restrictions (P3(δ^*)) and (P6(α)) need not hold. Instead, we rebuild the segments $S_t(i)$ for $i \geq j_{t-1}$, iteratively taking $S_t(i)$ to be the densest large subsegment of $S_t(i-1)$. This adversary satisfies (P1)-(P6). We will refer to this adversary as the *naive adversary*.

9.10.3 Requiring left and right buffers

We now turn our attention to condition (P7). Let E be an epoch at level i with associated segment S and start time s, and let Z be the set of items stored in S immediately prior to step s. We want to ensure that the number of moves charged to E is a constant fraction of $|Z|$. Consider the set Z' of items of Z that move at least once during E and for each $y \in Z'$, let t_y be the first step during the epoch that it moves. Under our convention for charging moves to epochs, we charge the move (y, t_y) to E provided that at step t_y, y is not stored within the successor interval $S_t(i+1)$. So q_E is at least the number of items $y \in Z$ that move during epoch E such that the first step it is moved it is not in the successor interval of S. To ensure that this is large, at the start time s we split S into three segments L, M, R where L is on the left and R is on the right. These are referred to, respectively, as the *left buffer* and *right buffer* of S. Let Y^L, Y^M and Y^R be the sets of items stored in each of these sets. We want our strategy to satisfy two things:

1. L, M, R are chosen so that Y^L and Y^R each contain a constant fraction of items from Z.

2. At every step $t \in E$, we restrict the choice of $S_t(i+1)$ so that it does not contain any items from $Y^L \cup Y^R$ that have not yet moved during epoch E. More precisely, let $B_{E,t-1}$ be the union of the busy segments from the beginning of E until step $t-1$; this is the region where the algorithm has shifted items during epoch E. Let $M_t = M \cup B_{E,t-1}$. We require that $S_t(i+1) \subseteq M_t$. In particular, when t is the start time of E we require $S_t(i+1) \subseteq M$.

If the selection strategy chooses $S_t(i+1)$ for each $t \in E$ to satisfy these two conditions, then (P7) holds for epoch E because the second condition implies that q_E is at least the number of items $y \in Y^L \cup Y^R$ that move at least once during the epoch and by the (assumed) laziness of the algorithm, during the epoch either all of the items in Y^L move, or all of the items in Y^R move.

9.10.4 Failure of the naive adversary

Unfortunately, the naive adversary does not meet these two conditions. We now discuss why, and modify the strategy so that it satisfies the above two conditions (and therefore (P7)) while preserving (P1)-(P6). To focus attention on the two conditions, we fix an epoch E at level i with start time s. The segment S associated to this epoch is equal to $S_t(i)$ for all $t \in E$. The epoch E is divided into one or more epochs at level $i+1$. Let H be the set of start times for level $i+1$ epochs during E. The two conditions restrict the choice of $S_t(i+1)$ for each $t \in H$.

For each step $t \in H$, the naive adversary selects $S_t(i+1)$ to be the densest large subsegment. Condition 2 above requires that $S_t(i+1) \subseteq M_t$ (and not just $S_t(i+1) \subseteq S$) so it is natural to modify the definition of $S_t(i+1)$ to be the subsegment of M_t (rather than the subsegment of S) of maximum density and size between $|M_t|/4$ and $|M_t|/2$.

The modified algorithm will still satisfy (P1), (P2(σ)) and (P5). However, where before we could be sure that the density of $S_t(i+1)$ is at least the density of S, now we only have that it is at least the density of M_t, but the density of M_t might be much lower than that of S. Thus conditions (P3(δ^*)), (P4) and (P6(α)) are no longer guaranteed.

9.10.5 Enforcing κ-lower balance

To solve this problem we'll insist that every segment chosen to start a new epoch satisfy a condition that will ensure that the density of M_t is either greater than

that of S or only slightly less. More precisely, we introduce a property of segments called κ-*lower balance*, where $\kappa > 0$ is a parameter that we will eventually take to be $\Theta(1/\log(n))$. We will say S is κ-lower balanced with respect to a given configuration if every subsegment of size at least $|S|/4$ has density at least the density of S times $\left(\frac{1}{4}\right)^{\kappa}$, which for small κ is $(1 - O(\kappa))$. If we knew that each selected segment satisfied this additional condition then for every step $t \in H$, M_t (which has size at least $|S|/4$) will have density at least $1 - O(\kappa)$ times that of S. This will be enough to get (P1)-(P7).

So we modify the construction of the segments $S_j(u)$ once again so that whenever a new epoch is started, the segment chosen for that epoch is κ-lower balanced. This is done as follows. Fix an epoch E at level i with start time s and associated segment S as before, and assume inductively that S is κ-lower balanced at the beginning of the epoch. We split S into three parts L, M, R where M is the middle third of S. By κ-lower balance we know that L,M,R all have density at least $\left(\frac{1}{4}\right)^{\kappa}$ times the density of S, and consequently L and R each contain about $1/3$ of the items stored in S.

We consider a step $t \in E$ that starts a new $i + 1$ epoch (this includes the case $t = s$) and describe how we'll select $S_t(i + 1)$. As previously defined, let $M_t = M \cup B_{E,t-1}$. By κ-lower balance, the density of M_t at the start of E is at least $1 - O(\kappa)$ times the density of S. This is still true at step t since both S and M_t contain the same items they had at the beginning of the epoch plus the items added during the epoch. We need to select $S_t(i + 1) \subseteq M_t$ that is κ-lower balanced. If M_t were κ-lower balanced we could just take $S_t(i + 1)$ to be M, but in general M_t need not be κ-lower balanced, so we'll need to search for a large κ-lower balanced subsegment.

It turns out to be easier to find a subsegment with a closely related property called κ-upper balance. S is κ-*upper balanced* if every subsegment of size at least $|S|/4$ has density *at most* the density of S times 4^{κ}. While κ-upper balance does not imply κ-lower balance, it is not hard to show that if U is $\kappa/24$-upper balanced then the middle third of U is κ-lower balanced. So we'll find a large subset of M_t that is $\kappa/24$-upper balanced and has density at least that of M_t and then take $S_t(i + 1)$ to be the middle third.

The following simple iterative process can be used to find a $\kappa/24$-upper balanced subset of M_t. Initialize the set A to be M_t and perform the following iteration: If A is $\kappa/24$-upper balanced then stop. Otherwise, since $\kappa/24$-upper balance is violated there is a subset A' of size at least $|A|/4$ that has density more than 4^{κ} times that of A. Now replace A by A' and repeat. Since during each iteration the density of A increases by a factor at least 4^{κ} at each step this must terminate.

This gives a well defined process for selecting the segment $S_t(i + 1)$ at the start

of a level i epoch: Start from M_t, apply the iterative process until it terminates and then take the middle third.

9.10.6 Controlling segment lengths

There is one remaining issue: the iterative process must terminate but might require many iterations, and $|S_t(i)|$ may be much smaller than $|S_t(i-1)|$. Therefore it is no longer immediate that we can continue this for $\Omega(\log(n))$ levels (while ensuring that the final segment has at least 2 items). Nevertheless, we can indeed continue this for $\Omega(\log(n))$ levels. In the troublesome case that $|S_t(i)|$ is much smaller than $|S_t(i-1)|$ (because the selection of $S_t(i)$ took many iterations), the density of $S_t(i)$ will be significantly higher than that of $S_t(i-1)$. Since the density can not rise above 1, this can be used to show that the segment sizes can't shrink too quickly. A convenient way to simultaneously account for size and density is to measure the quality of a segment by a function that combines both the density and size. A natural form for such a function is $\phi(U) = \rho(U)^\alpha |U|^\beta$. It's easy to check that if $\beta/\alpha \geq \kappa$ then this function increases during each of the above iterations. So we fix $\alpha = 1$ and $\beta = \kappa$, and for this choice of ϕ its not hard to show that in passing from $S_t(i)$ to $S_t(i+1)$, the function ϕ decreases by at most a factor $(1 - O(\kappa))$. This implies that we can take the depth to be $\Theta(1/\kappa)$. For $\kappa = \Theta(1/\log(n))$ this gives what we want.

The introduction of the quality function ϕ allows for a slight simplification of the selection of $S_t(i)$. The proof that ϕ is nondecreasing in each iteration actually shows that if U is a segment whose ϕ value is at least that of any subsegment of U then U is κ-upper balanced. So to obtain a κ-upper balanced subset of M_t we can (and do) replace the iterative process by the function **densify** of the previous section which selects the subset of maximum ϕ value.

To summarize, for the start time s of E, having chosen $S = S_s(i-1)$ the selection of $S_t(i)$ involves three steps: first we partition S into L, M, R and restrict to the middle third M of S (this corresponds to the set $T_s(i)$ in the definition of the adversary). The process of constructing $S_t(i+1)$ for $t \in E$ with $t > s$ is similar except we take $T_t(i)$ to be M_t instead of M.

9.11 The Proof that the Adversary Satisfies (P1)-(P7)

The remainder of this section gives the proof of Lemma 9.7.2 that our adversary with the parameters chosen according to (9.10)-(9.14) satisfies (P1)-(P7).

We start by noting that property (P1) and (P5) which require that the initial segment of every chain have size at most $n/2$ and the segments decrease in size by

at least a factor of 2 are obvious from the definition of the adversary. It remains to verify the remaining five properties.

9.11.1 Properties of **balance** and ϕ

Property (P6(α)) asserts that in each selected segment chain, the density of a selected segment can not be much smaller than its predecessor segment. The proof sketch of the previous subsection explained qualitatively how the function **balance** accomplishes this; here we provide the technical details. This will also be needed to establish (P3(δ^*)) and (P2(σ)) which give lower bounds on the density and weight of any segment occurring in any chain, and (P4) which establishes that there is always a suitable gap.

Let us fix a configuration (Y, f) and let ρ be the associated density function. Let $\kappa > 0$. We start by formally defining κ-upper balance and κ-lower balance with respect to the configuration (Y, f), which were mentioned in the previous subsection.

- S is *κ-upper balanced* (with respect to ρ) if every subsegment of size at least $|S|/4$ has density at most $\rho(S)4^\kappa$. item S is *κ-lower balanced* if every subsegment of size at least $|S|/4$ has density at least $\rho(S)(1/4)^\kappa$.

Recall the following definitions:

- The *quality of U with respect to* (Y, f) is the real number $\phi(U) = \rho(U)^{1/\kappa}|U|$.
- **densify**(U) is the subsegment of V of U that maximizes $\phi(V)$ (breaking ties arbitrarily).
- **balance**(U) is the subsegment V of U given by **middle**(**densify**(U)).

Proposition 9.11.1. *Let (Y, f) be an arbitrary configuration and let ρ be the associated density function. Let T be a segment and $D =$* **densify**(T).

1. *$\phi(T) \leq |T|$, i.e. the quality of a segment is at most its length. [This holds since $\rho(T) \leq 1$.]*
2. *$\rho(D) \geq \rho(T)$, i.e. applying* **densify** *to T produces a segment that is at least as dense.*
3. *D is κ-upper balanced. [Proof: If U is a subsegment of D of length at least $|D|/4$, the choice of D implies $\phi(U) \leq \phi(D)$, which implies $\phi(U)^\kappa \leq \phi(D)^\kappa$ which implies $\rho(U)(\frac{1}{4})^\kappa \leq \rho(D)$, which is the condition of κ-upper balance.]*

Lemma 9.11.2. *Fix a configuration (Y, f). Let T be an arbitrary segment, let $D = \textbf{densify}(T)$ and $S = \textbf{balance}(T) = \textbf{middle}(D)$. Assume that:*

$$|S| \geq 4 \text{ AND } \kappa \leq 1/24 \ln(4).$$

Then:

1. *For any subsegment U of S having size at least $|S|/4$, $\rho(U)/\rho(D) \geq (\frac{1}{4})^{24\kappa}$.*
2. *S is 25κ-lower balanced with respect to ρ.*
3. *$\phi(S)/\phi(T) \geq \frac{1}{3}(\frac{1}{4})^{24}$.*

Proof. We first show that parts 2 and 3 follow easily from part 1. For part 2, let U be a subsegment of S of size at least $|S|/4$. We have

$$\frac{\rho(U)}{\rho(S)} = \frac{\rho(U)}{\rho(D)}\frac{\rho(D)}{\rho(S)} \geq \left(\frac{1}{4}\right)^{24\kappa} \cdot \left(\frac{1}{4}\right)^{\kappa} = \left(\frac{1}{4}\right)^{25\kappa},$$

where the inequality uses part 1 and the fact that D is κ-upper balanced (by Proposition 9.11.1) and that $|S| \geq |D|/4$.

For part 3, note that

$$\frac{\phi(S)}{\phi(T)} \geq \frac{\phi(S)}{\phi(D)} = \frac{|S|}{|D|}\left(\frac{\rho(S)}{\rho(D)}\right)^{1/\kappa} \geq \frac{1}{3}\left(\frac{1}{4}\right)^{24},$$

by applying part 1 with $U = S$.

It remains to prove the first part. From Proposition 9.11.1, D is κ-upper balanced. $D - U$ consists of 2 segments L (on the left) and R (on the right). The κ-upper balance of D implies that $\rho(L)$ and $\rho(R)$ can't be much higher high than $\rho(D)$, and we'll show that this implies that the density of U can't be much lower lower $\rho(D)$. We have:

$$|D|\rho(D) = w(D) = w(L) + w(R) + w(U) = |L|\rho(L) + |R|\rho(R) + |U|\rho(U),$$

which implies:

$$\frac{\rho(U)}{\rho(D)} = \frac{1}{|U|}\left(|D| - |L|\frac{\rho(L)}{\rho(D)} - |R|\frac{\rho(R)}{\rho(D)}\right).$$

Since $|S| \geq 4$ and $S = \textbf{middle}(D)$ it follows that $|D| \geq 8$ and $|S| \leq |D|/2$. Since $U \subseteq S = \textbf{middle}(D)$, we have $|L|, |R| \geq |D|/4$ and since D is κ-upper balanced, it follows that $\rho(L)/\rho(D)$ and $\rho(R)/\rho(D)$ are each at most 4^{κ}. So:

$$\begin{aligned}
\frac{\rho(U)}{\rho(D)} &\geq \frac{1}{|U|}(|D| - (|L| + |R|)4^{\kappa}) \\
&\geq \frac{1}{|U|}(|D|4^{-\kappa} - |L| - |R|) \\
&= \frac{1}{|U|}(|D|4^{-\kappa} - |D| + |U|) \\
&= \frac{1}{|U|}(|U| - |D|(1 - e^{-\ln(4)\kappa})) \\
&\geq \left(1 - \frac{|D|}{|U|}\ln(4)\kappa)\right) \\
&\geq (1 - 12\ln(4)\kappa) \geq \left(\frac{1}{4}\right)^{24\kappa},
\end{aligned}$$

where the final inequality uses the hypothesis that $\kappa \leq 1/(24\ln(4))$ and the inequality $(1-x) \geq e^{-2x}$ for $x \leq 1/2$. □

9.11.2 Some technical inequalities involving the parameters

Our aim is to establish that the adversary satisfies (P1)-(P7) with the parameter choices of Equations (9.7)-(9.14).

We complete the proof in the next two subsections. The main part of the argument is an induction which provides a lower bound on the density and quality of all of the segments $S_t(i)$ and $T_t(i)$ produced by the adversary. Obviously the selected values of the parameters will play a role, and this role shows up as certain arithmetic inequalities that are required for the argument. Each of these inequalities can be verified by a routine calculation by plugging in the specific values of the parameters.

In preparation for this, we now collect all of the technical inequalities that are needed for the coming argument. We mention the role each inequality plays in the argument and highlight the most important ones. This section is optional for the reader; the reader can go directly to the next section and when encountering one of these needed inequalities in the next section, the reader can either do the (usually easy) verification himself or take our word for it.

Readers who choose to read this section may find the brief comments after each one hard to follow completely, because they refer forward to specific details in the subsequent proofs. The hope is that giving the reader an impression of the role that these technical properties play will facilitate reading the proof.

Here are the hypotheses carried over from Lemma 9.4.1:

(A1) $n \geq C_0$

(A2) $\delta_0 \in (\log(n))^{-2}, 1 - n^{-1/5})$.

The following properties of the chosen parameters can be deduced from (A1)-(A2) and the definitions of the parameters (9.7)-(9.14).

(R1) $\delta^*\sigma \geq 2$. This is a technical condition that ensures that $S_t(d)$ satisfies the hypotheses of Lemma 9.6.1 and thus has a suitable gap. It holds with a lot of room to spare.

(R2) $\sigma \geq 4$. For (P2), every segment $S_t(i)$ has size at least 4. This is a minor technical hypothesis of Lemma 9.11.3, which is needed so that we can apply Lemma 9.11.2. It holds trivially.

(R3) $(1/\delta_0)^{1/\kappa} \leq \sqrt{n}/2$. This is a significant inequality. In the basis of our induction, we want to show that (at the beginning of step t) the chosen segment $T_t(1)$ has quality value at least $\sqrt{n}$. This quality function value is easily bounded above by $\frac{n}{2}{\delta_0}^{1/\kappa}$. Since we want this to be at least $\sqrt{n}$ this restricts κ to be large enough. On the other hand we want κ to be $O(1/\log(n))$ so that the density degradation parameter α is that small. Fortunately these two restrictions can both be satisfied by the selected value of κ.

(R4) $\sigma \leq 2^{-2C_6 d}\sqrt{n}$. We want every segment to have length at least σ. The length of a segment is bounded below (at each step) by the quality function value, and the analysis will give us a lower bound on that. Our induction argument will lead to equations (9.29) and (9.27) which give lower bounds on the quality function value of all of the selected segments, and these lower bounds are all at least $2^{-2C_6 d}\sqrt{n}$. By adjusting the multiplicative constant for d we can make this as close to $\sqrt{n}$ as we want. We arbitrarily chose σ to be $n^{1/4}$ and adjusted d appropriately.

(R5) $2^{-2dC_6\kappa}\delta_0 \geq \delta^*$. For (P3($\delta^*$)) we need that every segment in every segment chain has density at least δ^*. This will follow immediately from the present inequality, and (9.28) which says that every segment in a segment chain has density at least $2^{-2dC_6\kappa}\delta_0$.

(R6) $\kappa \leq \frac{1}{24\ln(4)}$ and $\kappa \leq 1/50$. These are minor (and easily verifiable) technical upper bounds needed, respectively, for the hypothesis of Lemma 9.11.2, and the end of the proof of (P7)).

(R7) $\alpha = 2C_6\kappa$. This is a restatement of the definition (9.14) of α, which relates it to the adversary parameter κ. We prove (9.22) and (9.23) which give a lower bound on the ratio of densities of successive $S_t(i)$ in each chain as $2^{-2C_6\kappa}$,

so setting $\alpha = 2C_6\kappa$ gives us property Property (P6(α)) which requires that the ratio of densities of successive subsegments is at most $2^{-\alpha}$.

9.11.3 The behavior of ϕ and ρ along a segment chain

The next step in establishing the remaining properties is to prove a lemma that for each step t, gives a lower bound on ρ_{t-1} and ϕ_{t-1} of the first segment $S_t(1)$ and also shows that as we move from a segment to its successor, ρ_{t-1} and ϕ_{t-1} can't decrease by much.

Lemma 9.11.3. *Let C_6 and C_0 be as in (9.10) and (9.11). Suppose that the parameters d,κ,α,σ and δ^* satisfy (R1)-(R7) and that $n \geq C_0$. Let $S_t(i)$ and $T_t(i)$ be the segments chosen by the adversary and assume that all of them have size at least 4. For each $t \in \{1, \ldots, n\}$ we have:*

$$\rho_{t-1}(T_t(1)) \geq \delta_0, \tag{9.19}$$
$$\phi_{t-1}(T_t(1)) \geq \sqrt{n}, \tag{9.20}$$

and for $i \in \{1, \ldots, d\}$ we have:

if t starts an i-epoch then $S_t(i)$ is 25κ lower-balanced with respect to ρ_{t-1}. (9.21)

$$\frac{\rho_{t-1}(S_t(i))}{\rho_{t-1}(T_t(i))} \geq 2^{-C_6\kappa} = 2^{-\alpha/2}. \tag{9.22}$$
$$\frac{\rho_{t-1}(T_t(i+1))}{\rho_{t-1}(S_t(i))} \geq 2^{-C_6\kappa} = 2^{-\alpha/2} \qquad \text{if } i \leq d-1. \tag{9.23}$$
$$\frac{\phi_{t-1}(S_t(i))}{\phi_{t-1}(T_t(i))} \geq 2^{-C_6}. \tag{9.24}$$
$$\frac{\phi_{t-1}(T_t(i+1))}{\phi_{t-1}(S_t(i))} \geq 2^{-C_6} \qquad \text{if } i \leq d-1. \tag{9.25}$$

Proof. We fix an epoch E and prove the result for all $t \in E$. Let i be the level of the epoch. We divide into cases depending on whether or not t is the start time s_E. Case 1. $t = s_E$. For (9.19) and (9.20), recall that $T_t(1)$ is chosen to have highest density among segments of length between $n/2$ and n. Since the m locations of the array can be partitioned into segments of size between $n/2$ and n, and one of those must have density at least the density of the entire array, we conclude that $\rho_{t-1}(T_t(1)) \geq \delta_0$. From the definition of ϕ_{t-1}, $\phi_{t-1}(T_t(1)) \geq (n/2){\delta_0}^{1/\kappa}$, which is at least $\sqrt{n}$ (see (R3)).

For the proofs of the remaining parts in this case, we apply Lemma 9.11.2 with $T = T_t(i)$ and $S = S_t(i)$, in fact that lemma was designed The hypotheses of Lemma 9.11.2 require that κ be smaller than $24\ln(4)$ which is (R6) and that $|S_t(i)| \geq 4$, which is a hypothesis of the present lemma. Since $S_t(i) = \mathbf{balance}(T_t(i))$, part 2 of Lemma 9.11.2 implies that $S_t(i)$ is 25κ-lower balanced. Inequality (9.22) follows from the first part of Lemma 9.11.2 with $U = S$, using $C_6 = 60 \geq 48$. Inequality (9.25) follows from the third part of Lemma 9.11.2 with $U = S$. Since $S_t(i)$ is 25κ-lower balanced and $|T_t(i+1)| \geq |S_t(i)|/4$ we have (9.23). For (9.25) we use the definition of ϕ_{t-1}, together with (9.23) and the fact that $|T_t(i+1)| \geq |S_t(i)|/4$.

Case 2. $t > s_E$. We may assume by the first case that the four inequalities hold with t replaced by s_E. We will show that they continue to hold throughout the epoch E. We will repeatedly use the following easy fact:

Proposition 9.11.4. *Let $S \subseteq S'$ be segments and $s < t$ be steps. Suppose that for all steps $r \in \{s, \ldots, t-1\}$, the busy segment B_r is a subset of S. Then $\rho_r(S)$, $\phi_r(S)$, $\rho_r(S)/\rho_r(S')$ and $\phi_r(S)/\phi_r(S')$ are all nondecreasing as a function of $r \in \{s, \ldots, t-1\}$*

Proof. This follows from: At each step in $\{s, \ldots, t-1\}$, both $w_r(S)$ and $w_r(S')$ increase by λ and $w_r(S) \leq w_r(S')$. □

Let $s < t$ be the start time of the epoch. Note that during epoch E, the sets $S_t(i) = S_s(i)$ and $T_t(i) = T_s(i)$. Therefore (9.19) and (9.20) and (9.22) and (9.24) follow from the corresponding inequalities for step s together with Proposition 9.11.4.

For (9.23) and (9.25), we cannot apply Proposition 9.11.4 directly because, while $S_t(i) = S_s(i)$, it may not be true that $T_t(i+1) = T_s(i+1)$ because there may have been one or more new $i+1$-epochs started between s and t and so $T_u(i+1)$ may change during the epoch. Note that since the step interval $\{s, \ldots, t\}$ is a subinterval of the epoch E, at any time $u \in \{s, \ldots, t\}$ that $T_u(i+1)$ changes $i+1$ must be the critical index j_{u-1}. We will need to make careful use of the adversary construction (Adv1.b) of the T-set at the critical index.

Proposition 9.11.5. *Let $i \in \{1, \ldots, d\}$ be a level and t a step that belongs to the level i epoch E, whose start time is s. Then $T_t(i+1) = T_s(i+1) \cup \bigcup_u B_u$ where u ranges over all steps $u \in \{s, \ldots, t-1\}$.*

Proof. We prove this by induction on $t \geq s$; it holds trivially for $t = s$. Assume $t > s$; by induction it suffices to show that $T_t(i+1) = T_{t-1}(i+1) \cup B_{t-1}$. If the critical index j_{t-1} is greater than $i+1$ then this is trivial since then $B_{t-1} \subseteq S_{t-1}(i+1) \subseteq T_{t-1}(i+1)$. Otherwise $j_{t-1} = i+1$ and the conclusion follows directly from the definition of the adversary. □

We are trying to show that the density of $T_t(i+1)$ can not be much smaller than the density of $S_t(i+1)$. According to Proposition 9.11.5, the interval $T_t(i+1)$ is equal to $T_s(i+1)$ combined with some busy sets. These busy sets are determined by the algorithms behavior, so it is conceivable that the algorithm might be able to choose the busy sets cleverly so as to drive the density of $T_t(i+1)$ down significantly. We'll show this can't happen using the first part of Lemma 9.11.2. What we'll do is consider the density of the segment $T_t(i+1)$ with respect to the density function ρ_{s-1} used during step s. Since $T_t(i+1)$ is a subsegment of $S_t(i) = S_s(i)$ that contains $T_s(i+1)$ and is therefore of size at least $|S_t(i)|/4$, we can apply the second part of Lemma 9.11.2 to get that $\rho_{s-1}(T_t(i+1))/\rho_{s-1}(S_s(i)) \geq \left(\frac{1}{4}\right)^{24}$ which is at least $2^{-C_6\kappa}$. Proposition 9.11.4 implies that the same inequality holds for ρ_{t-1} (keeping in mind that $S_t(i) = S_s(i)$). Since $|T_t(i+1)|/|S_t(i)| \geq 1/3$ we also get (9.25). □

9.11.4 Completing the proof of Lemma 9.7.2

Using Lemma 9.11.3 repeatedly we have by induction on $i = 1, \ldots, d$ for fixed $t \in \{1, \ldots, n\}$, that:

$$\begin{aligned}
\rho_{t-1}(T_t(i)) &\geq \delta_0 2^{(2-2i)C_6\kappa} && (9.26)\\
\phi_{t-1}(T_t(i)) &\geq \frac{\sqrt{n}}{2} 2^{(2-2i)C_6} \geq \sigma && (9.27)\\
\rho_{t-1}(S_t(i)) &\geq \delta_0 2^{(1-2i)C_6\kappa} \geq \delta^* && (9.28)\\
\phi_{t-1}(S_t(i)) &\geq \frac{\sqrt{n}}{2} 2^{(1-2i)C_6} \geq \sigma. && (9.29)
\end{aligned}$$

The final inequality of (9.28) follows from (R5). The final inequalities of (9.27) and (9.29) follow from (R4). Note that the final inequality of (9.29) and (R2) imply that as we proceed to level i in the induction, the hypothesis $|S_t(i)| \geq 4$ of Lemma 9.11.3 holds at each step.

This gives us what we need to establish the remaining properties.

Proof of Property (P2(σ)). This says that all segments selected for segment chains have length at least σ. Since $|S_t(d)| \geq \phi_{t-1}(S_t(d))$ this follows from (9.29). □

Proof of Property (P3(δ^*)). This says that all segments selected for segment chains have density at least δ^* and is included in (9.28). □

Proof of Property (P4). We need to establish that the hypotheses of Lemma 9.6.1 (the suitable gap lemma) are satisfied. For the lower bound on the mingap of Y_0 we have from the hypothesis of Lemma 9.4.1 and the choice $\lambda = \delta_0/6$ that $\text{mingap}(Y_0) \geq 1 + 12/\delta_0 \geq 1 + 2/\lambda$ as required. The hypothesis $\rho_{t-1}(S_t(i)) \geq 2\lambda$ follows from (P3(δ^*)) and the fact that $\delta^* \geq \delta/3 \geq 2\lambda$. The hypothesis

$w_{t-1}(S_t(i)) \geq 2$ follows from $w_{t-1}(S_t(i)) = |S_t(i)|\rho(S_t(i)) \geq \sigma\delta^*$ by properties (P2(σ)) and (P3(δ^*)) which is at least 2 (e.g.see (R1)). □

Proof of Property (P6(α)). This requires a lower bound on $\rho_{t-1}(S_{t-1}(1))$ and on the ratios $\rho_{t-1}(S_t(i))/\rho_{t-1}(S_t(i-1))$. For the first bound we use (9.22) with $i = 1$ and (9.19). For the second we combine (9.22) and (9.23). In both cases we need the fact that $\alpha = 2C_6\kappa$. □

Proof of Property (P7). Let E be an epoch at level i. We want to bound q_E from below, where q_E is the number of moves done during epoch E that were assigned to epoch E, which means moves that occured during steps $t \in E$ of items that were stored in $S_t(i) - T_t(i+1)$ prior to the move. Let s denote the start time, and c denote the closing time, of epoch E. The busy segment B_c includes a location outside of $S_E(i)$ (this is the reason that the epoch closed at step c.) Without loss of generality let us say that B_c includes a location that is to the left of $S_E(i)$. Let L be the left segment of $S_s(i) - T_s(i+1)$.

The number of moves charged to epoch E is q_E. We'll show that (1) every item stored in L at the start time s of E moves during E and (2) the first such move is charged to epoch E, and (3) the number of such items is at least $|S_s(i)|\rho_{s-1}(S_s(i))/8$. This immediately immediately gives (P7).

For (1), note that B_c must include a location in $S_c(i+1) \subseteq T_c(i+1)$ and so $B_c \cup T_c(i+1)$ is a segment that must contain all of L. By Proposition 9.11.5, L must be a subset of the union of the busy segments $B_s \cup \cdots \cup B_c$ which means that every item stored in L under $\mathbf{f}_{s-1}$ must move during E. For (2) consider an item y that was stored in location $\ell \in L$ under $\mathbf{f}_{s-1}$. Let t be the earliest step in E that $\ell \in B_t$. Thus $\ell \notin T_t(i+1)$ so $\ell \notin S_t(i+1)$ and so the move of y at step t is charged to epoch E. For (3) L is a subsegment of $S_t(i) = S_s(i)$ of size at least $|S_s(i)|/4$. As $S_s(i)$ is 25κ-lower balanced with respect to ρ_{s-1} by (9.21), $\rho(L) \geq \rho(S_s(i))(1/4)^{25\kappa} \geq \rho(S_s(i))/2$, since $\kappa \leq 1/50$ by (R6). □

This completes the proof of Lemma 9.7.2, and thereby also the proof of Lemma 9.4.1.

Bibliography

[1] Yehuda Afek, Baruch Awerbuch, Serge A. Plotkin, and Michael E. Saks. *Local management of a global resource in a communication network.* Journal of the ACM, 43(1):1–19, 1996.

[2] Martin Babka, Jan Bulánek, Vladimír Čunát, Michal Koucký, and Michael E. Saks. *On online labeling with polynomially many labels.* In Proceedings of the 20th Annual European Symposium on Algorithms, ESA '12, pages 121–132, 2012.

[3] Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. *Two simplified algorithms for maintaining order in a list.* In Proceedings of the 10th Annual European Symposium on Algorithms, ESA '02, pages 152–164, 2002.

[4] Michael A. Bender, Erik D. Demaine, and Martin Farach-Colton. *Cache-oblivious B-trees.* SIAM Journal on Computing, 35:341–358, 2005.

[5] Michael A. Bender, Ziyang Duan, John Iacono, and Jing Wu. *A locality-preserving cache-oblivious dynamic dictionary.* Journal of Algorithms, 53(2):115 – 136, 2004.

[6] Michael A. Bender, and Haodong Hu. *An adaptive packed-memory array.* ACM Transactions on Database Systems 32(4), 2007.

[7] Richard S. Bird and Stefan Sadnicki. *Minimal on-line labelling.* Information Processing Letters, 101:41–45, 2007.

[8] Alan Borodin, and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.

[9] Gerth Stølting Brodal, Rolf Fagerberg, and Riko Jacob. *Cache oblivious search trees via binary trees of small height.* In Proceedings of the 13th annual ACM-SIAM Symposium on Discrete Algorithms, SODA '02, pages 39–48, 2002.

[10] Jan Bulánek, Michal Koucký, and Michael E. Saks. *Tight lower bounds for the online labeling problem.* In Proceedings of the 44th Symposium of Theory of Computation, STOC '12, pages 1185–1198, 2012.

[11] Jan Bulánek, Michal Koucký, and Michael Saks. *On Randomized Online Labeling with Polynomially Many Labels.* In Proceedings of the 40th International Colloquium on Automata, Languages and Programming, ICALP '13, pages 291-302, 2013.

[12] Paul F. Dietz, Joel I. Seiferas, and Ju Zhang. *Lower bounds for smooth list labeling.* Manuscript, 2005. (Listed in the references of [13]).

[13] Paul F. Dietz, Joel I. Seiferas, and Ju Zhang. *A tight lower bound for online monotonic list labeling.* SIAM Journal on Discrete Mathematics, 18:626–637, 2005.

[14] Paul F. Dietz and Ju Zhang. *Lower bounds for monotonic list labeling.* In Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory, SWAT '90, pages 173–180, 1990.

[15] Yuval Emek and Amos Korman. *New bounds for the controller problem.* Distributed Computing, 24(3-4):177–186, 2011.

[16] Alon Itai, and Irit Katriel. *Canonical density control.* Information Processing Letters, 104:200–204, 2007.

[17] Alon Itai, Alan G. Konheim, and Michael Rodeh. *A sparse table implementation of priority queues.* In Proceedings of the 8th Colloquium on Automata, Languages and Programming, ICALP '81, pages 417–431, 1981.

[18] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*, Addison Wesley Longman Publishing Co., Inc., 1998.

[19] Amos Korman and Shay Kutten. *Controller and estimator for dynamic networks.* In Proceedings of the 26th annual ACM Symposium on Principles of Distributed Computing, PODC '26, pages 175–184, 2007.

[20] Dan E. Willard. *A density control algorithm for doing insertions and deletions in a sequentially ordered file in a good worst-case time.* Information and Computation, 97(2):150–204, 1992.

[21] Ju Zhang. *Density Control and On-Line Labeling Problems.* PhD thesis, University of Rochester, Rochester, NY, USA, 1993.