Tutorial 2

Data Structures 1, 28. 2. 2025 `https://iuuk.mff.cuni.cz/~chmel/2425/ds1en/`

Exercise 1 (Stretching arrays)
During the lecture, you have seen that a flexible-sized array can be implemented in amortized constant time by letting the array fill up, and when it is full and we want to insert another element, we copy everything into an array that is double the size.
What would happen, if we changed the capacity C in a different way rather than doubling it every time? (Consider only inserts here.)

a) $C \rightsquigarrow C + k$, where $k \geq 1$ is a constant,

b) $C \rightsquigarrow C^2$,

c) $C \rightsquigarrow k \cdot C$ for a constant $k > 1$.

Exercise 2 (Flexible arrays have potential!)
In the lecture, you have seen the analysis of the doubling flexible arrays. There, you used the aggregate method to argue that doubling the array when full and halving the array when at the quarter of its capacity gives amortized constant time.
Prove the same result using a potential defined as $\phi(\text{array}) = |\frac{\text{capacity of array}}{2} - \text{number of elements in array}|$.

Exercise 3 (Bounded-balance-trees and aggregated analysis)
During the lecture, you have shown using a potential that the amortized complexity of all operations on BB[α]-trees is $\mathcal{O}(\log n)$. Prove the same result using aggregate analysis.

Bonus exercises

Exercise 4 (Binary counter)
Recall the proof that the binary counter (staring at zero) has amortized constant complexity of adding one. How does the amortized cost change if we allow not only adding one, but also subtracting one?
Is it possible to modify the counter so that the amortized cost remains constant?

Exercise 5 (Binary counter II)
We are in the same situation as in the previous exercise (Binary counter) and only want to add ones.
Show that when we pay 2^k for swapping the k-th bit (and the 0th bit is the least significant bit), then adding ℓ ones costs $\mathcal{O}(\ell \cdot n)$.

Exercise 6 (A sublinear set)
We have the following implementation of a set (of integers) that supports operations Insert, Lookup.
The set `M` is implemented as an array of arrays, where the array `M[i]` has length 2^i. Every array `M[i]` is either empty or completely full, and every array is sorted in increasing order (though there is no relationship between distinct arrays).
As an example, if we have 9 elements in our set, the arrays `M[0], M[3]` are full and all others are empty (this corresponds to the binary representation of the number 9). See, for example the following:

```
M[0] = {5}
M[1] = {}
M[2] = {}
M[3] = {3, 4, 8, 10, 11, 15, 19, 22}
```

a) What is the complexity of the Lookup operation, that is implemented by binary searching on every full array?

b) The Insert operation will be implemented as follows: we start by creating an array with the single inserted element. Next, we look at the array `M[0]`. If it is empty, we add our new array there. If it is full, then we merge our array with the array `M[0]` and we continue with the array `M[1]`, until we find an empty array `M[i]`. What is the (amortized) cost of the Insert operation?

Hint: the previous exercise might be helpful.