Takto jsou zobrazeny poznámky cvičících k tomu, co a jak je tu napsáno, skutečný text je mimo oranžové rámečky.

Počítač, na kterém byly provedeny testy: Procesor Intel(R) Core(TM) i7-5557U CPU @ 3.10GHz, RAM 16GB.

Operační systém: Debian 8

Překlad: clang++ verze 3.8, optimalizace -O2.

Konfigurace počítače je důležitá zejména pro měření času. Důležitými parametry jsou: verze procesoru a velikost RAM. Model procesoru můžete na linuxu zjistit příkazem `cat /proc/cpuinfo`. Tyto parametry uvádíme proto, aby všechny testy byly opakovatelné.

1 Test 1 – naivní vs standardní

1.1 Standardní implementace

Je povoleno kouknout se do zdrojového kódu nebo výstupu generátoru a zjistit, co měříme.

První graf (Obrázek 1) zobrazuje průměrný počet kroků na jednu operaci pro různé typy testů a standardní implementaci vektoru. V sekvenčním testu jsme postupně vložili n prvků a následně jsme je zase všechny odebrali. Ve zbývajících dvou testech jsme nejprve vložili n (respektive nejbližší nižší mocninu dvojky) prvků a poté jsme udělali $10n$ náhodných operací v případě náhodného testu nebo jsme $10n$-krát provedli posloupnost dvou vložení a dvou odebrání prvků v případě extrémního testu. Z přednášky víme, že amortizovaná cena jedné operace push nebo pop je konstantní.

Odhadneme asymptotickou složitost operace.

Z grafu naměřených hodnot můžeme vidět, že u všech typů a všech provedených měření lze průměrný počet kroků na jednu operaci shora odhadnout konstantou, což souhlasí s tvrzením z přednášky.

Pozor, že děláme odhady toho, co jsme naměřili. V žádném případě z naměřených dat nedostáváme důkaz, že se datová struktura bude chovat dle našich představ i pro vyšší, než námi naměřené hodnoty. Dokonce ani pro měřené velikosti nedostáváme horní odhad chovaní datové struktury (mohou existovat horší posloupnosti operací). Takové tvrzení by bylo třeba zdůvodnit (důkazem z přednášky, intuicí. . .).

Odhadneme multiplikativní konstantu.

Z grafu pro sekvenční test můžeme shora odhadnout multiplikativní konstantu okolo 2.2 kroku na operaci. Povšimněme si, že vektor dosahoval mnohem lepších výsledků pro náhodný a extrémní test, kde bylo průměrně na operaci potřeba udělat 1.1 kroku.

Kdybychom do kroků nezapočítávali vložení prvků, ale pouze každé překopírování prvku, při změně velikosti vektoru, pak by nám konstanta vyšla zhruba 0.2, což je menší než 1. V některých domácích úkolech se tohle může stát.

Všimneme si, co je v grafech za zvláštnosti a pokusíme se je zdůvodnit.

U všech tří křivek si můžeme všimnout, že přibližně na mocninách dvojky dojde k prudkému nárůstu počtu kroků, který se potom postupně snižuje k další mocnině dvojky. Nejvíce se toto chování projevuje u sekvenčního testu. Domnívám se, že je to způsobeno tím, že pro tuto velikost dojde k alokaci většího pole a překopírování prvků mezi novým a starým polem. Při vkládání více prvků do již alokované volné paměti není potřeba alokovat více místa a proto se průměrný počet kroků na operaci snižuje. U náhodného a extrémního testu nepozoruji v grafech tak výrazné skoky. Myslím si, že jsou způsobeny úvodní inicializační fází. Poté se udělá spousta operací, které nevyvolají realokaci, a tudíž dojde k zjemnění skoků z inicializační fáze.

Dále si můžeme všimnout, že pro velmi malé velikosti množiny standardní implementace v sekvenčním i extrémním testu o trošku roste, ale na druhou stranu v náhodném testu trošku klesá.

Myslím, že je to dané jednak tím, že jsme úvodní velikost vektoru zvolili osm, to se projeví zejména v testech, při kterých se velikost vektoru jen zvětšuje a pak jen zmenšuje. Dalším vlivem může být to, že pro malé vektory je větší pravděpodobnost několika operací push a několika operací pop tak, že se velikost vektoru změní (pokud stojíme v nule a podle hodu mincí jdeme o krok vpravo nebo vlevo, pak v 100 krocích častěji dojdeme do 10 než do 80).

V tomto případě je zdůvodnění nad rámec znalostí z přednášky a základní pochopení chování datové struktury, proto jej není nutné uvádět.

1.2 Naivní implementace

Přidáváme i graf s logaritmickou škálou, protože nárůst počtu kroků v extrémním případě způsobí, že ostatní křivky jsou příliš blízko nule a nejsou vidět.

Na dalších třech grafech (Obrázek 2, 3 a 4) můžeme pozorovat chování naivní implementace. V grafu naivní implementace (Obrázek 2) můžeme vidět, že na rozdíl od správné implementace průměrný počet kroků na jednu operaci v našem měření roste v extrémním testu lineárně s velikostí držené množiny. Multiplikativní konstanta se pohybuje mezi jednou polovinou a jednou čtvrtinou (viz Obrázek 4).

Myslím si, že k lineárnímu nárůstu dochází díky tomu, že struktura je nucena často přealokovávat celé pole. Domnívám se, že k tomu dochází díky vhodně zvolené velikosti držené množiny (vždy držíme množinu velikosti nejbližší menší mocniny dvojky) a posloupnosti příkazů (střídání operace push a pop).

Rozdílné je také chování u náhodného testu, ve kterém se v případě naivní implementace objevují mnohem větší peaky, pokud velikost držené množiny je přibližně mocnina dvojky. Výšku peaků bychom z grafu mohli odhadnout pomocí odmocniny z n s multiplikativní konstantou $\frac{3}{10}$. Tato funkce je znázorněna v grafu (Obrázek 4) šedou křivkou. Toto celkem sedí s teorií náhodných procházek na celých číslech.

Všimnout si jevu je třeba a je třeba to napsat! Zdůvodnění mimo selského rozumu, znalostí z Bc. studia a z přednášek třeba není, proto zmínka o náhodných procházkách není nutná. V tomto konkrétním případě není nutný ani odhad rychlosti růstu jako odmocnina. Za chybné zdůvodnění nebo odhad asymptotiky nejspíš nestrhneme víc bodů než za chybějící zdůvodnění.

Ideální by nejspíš bylo změřit tyto hodnoty i okolo větších mocnin dvojky, ale bylo by velmi časově náročné to udělat pro více než dvě další.

Návrh dalších experimentů testujících naše hypotézy také není špatný.

Také si můžeme všimnout drobného nárůstu naivní implementace viditelného na Obrázku 2. Nevím, čím je toto způsobené.

Není třeba zdůvodnit naprosto všechno, jen to, co se přímo týká teorie nebo je jednoduše vymyslitelné.

1.3 Shrnutí

V sekvečních testech vykazovaly obě struktury podobné vlastnosti, ale v ostatních testech se chovala standardní struktura lépe. Nejvýraznější byl rozdíl v extrémním testu, ve kterém standardní implementace dosahovala konstantní složitosti okolo 1.06, zatímco naivní implementace roste lineárně v závislosti na velikosti držené množiny. V náhodném testu dosahovaly obě struktury shodných výsledků, vyjma případů, kdy velikost držené mocniny byla okolo mocniny 2. V těchto případech si v naměřených výsledcích správná struktura udržovala konstantní průměr počtu kroků na jednu operaci, zatímco v případě naivní struktury průměr rostl s odmocninou velikosti množiny.

2 Test 2 – push s různými c

Z grafů znovu pozorujeme kolísání na mocninách nafukovacího parametru c (předtím bylo pozorováno na mocninách $c = 2$) a pak pozvolné klesání. Vysvětlení je obdobné, na mocnině c znovu alokujeme a pak přidáváme další prvky do prázdného místa, ale nic za ně nealokujeme, pouze jejich počtem dělíme.

Všechny křivky bych odhadl zdola zhruba jako $1 + \frac{1}{c-1}$ a shora přibližně $2 + \frac{1}{c-1}$, což odpovídá právě alokaci na mocninách c a větě z přednášky, pro několik hodnot vyobrazeno na Obrázku 8.

Jak jsme si řekli na přednášce, i pro malá $c > 1$ dynamické pole může fungovat, ale se zmenšujícím odstupem c od jedničky rychle ztrácíme efektivitu. Dle naměřených výsledků datovou strukturu výrazně nezrychlíme, pokud přenastavíme hodnotu c ze dvou na vyšší, avšak v takovém případě alokujeme velké množství paměti.

Pozorování byla provedena na základě Obrázku 7.

Logaritmické měřítko uvádíme jen pro ukázku.

3 Měření času

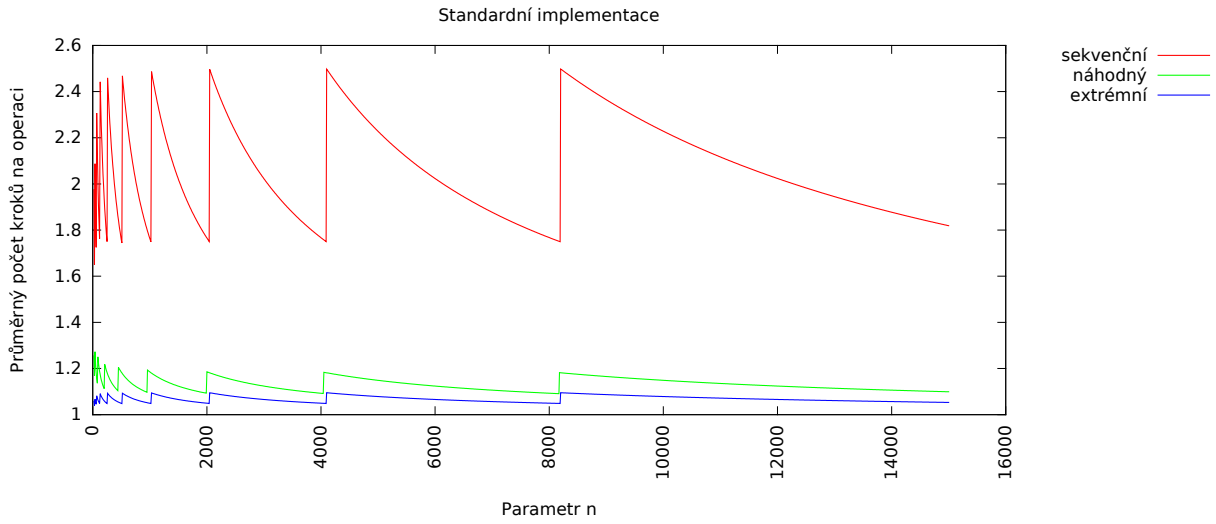
Pro obě měření bereme stejnou sekvenci a pětkrát měříme průměrnou dobu jedné operace.

V grafu měření času standardní implementace (Obrázek 12) vidíme, že se hodnoty pohybují více méně v malém konstantním rozmezí. Průměrný čas na operaci se pohybuje mezi $2 \cdot 10^{-9}$ a $3 \cdot 10^{-9}$, což odpovídá teorii dokázané na přednášce, že cena jedné operace je amortizovaně konstantní. Díky opakování měření vidíme, že jednotlivé měření není přesné a projevuje se náhodnost (daná měřením na reálném počítači).

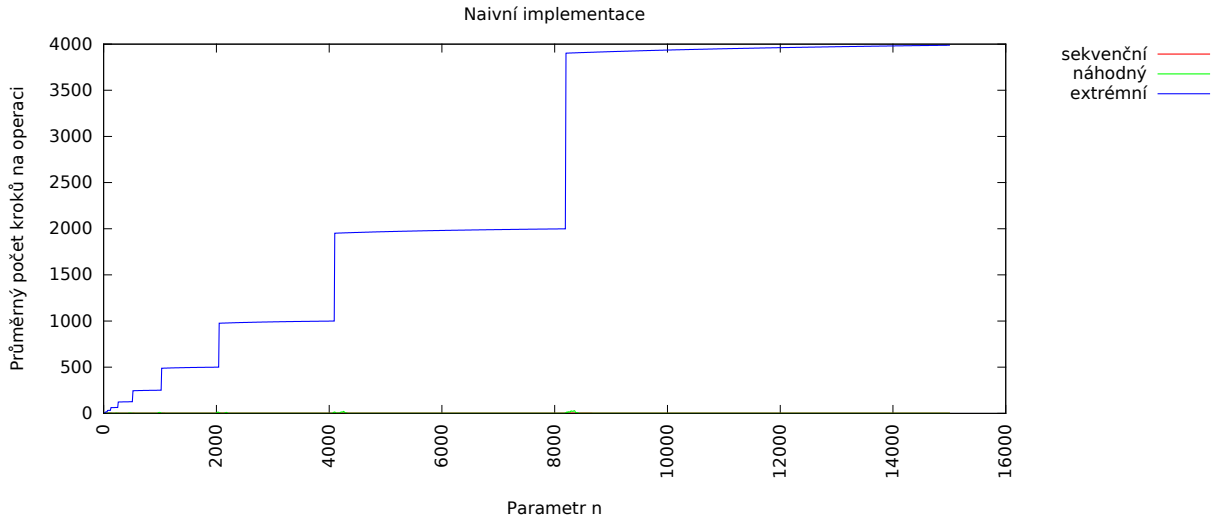
V grafu měření naivní implementace (Obrázek 13) jsou náhodné odchylky zanedbatelné vzhledem k průměrné délce trvání jedné operace. Největší naměřená hodnota je zhruba 10000-krát větší než hodnota pro stejně velké pole u standardní implementace. Celkově tento graf tvarem odpovídá grafu na Obrázku 5, odůvodnění je stejné jako pro simulaci.

4 Grafy

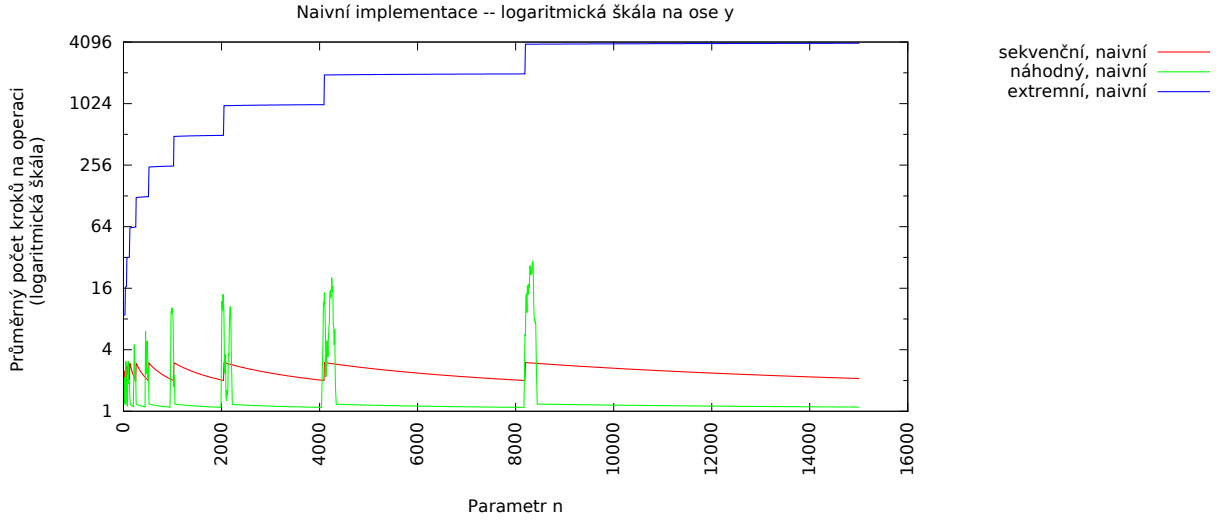
4.1 Grafy Test 1



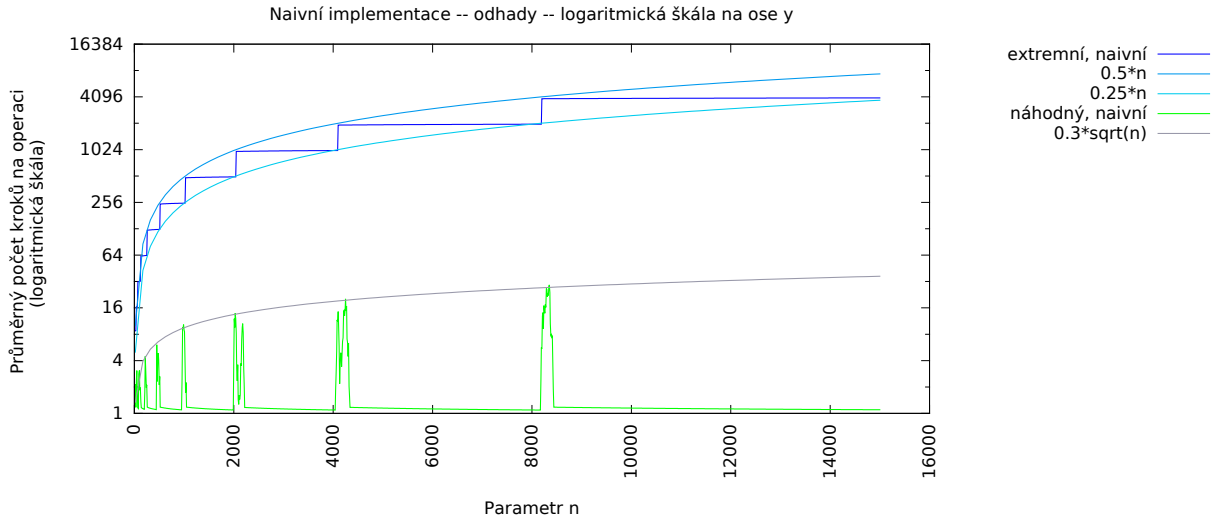
Obrázek 1: Správná implementace.



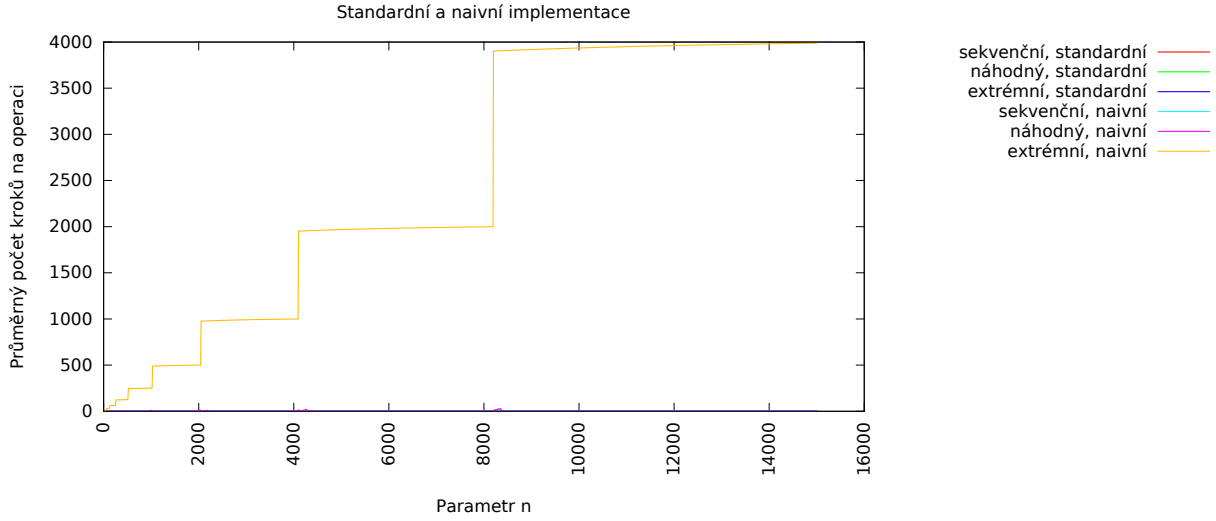
Obrázek 2: Naivní implementace.



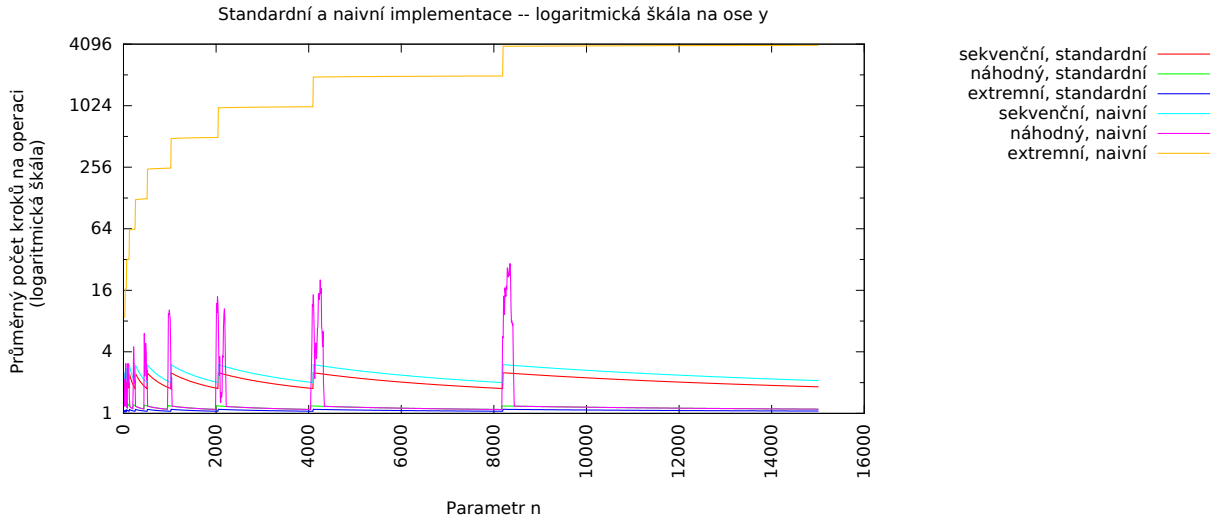
Obrázek 3: Naivní implementace v logaritmické škále.



Obrázek 4: Naivní implementace spolu s křivkami odhadů v logaritmické škále.

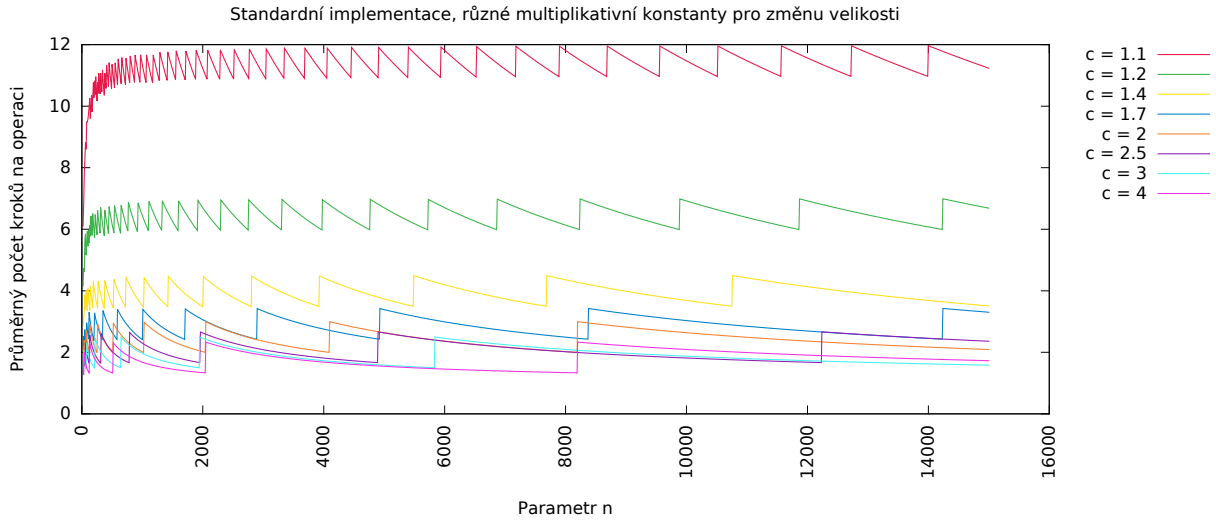


Obrázek 5: Naivní vs správná implementace.

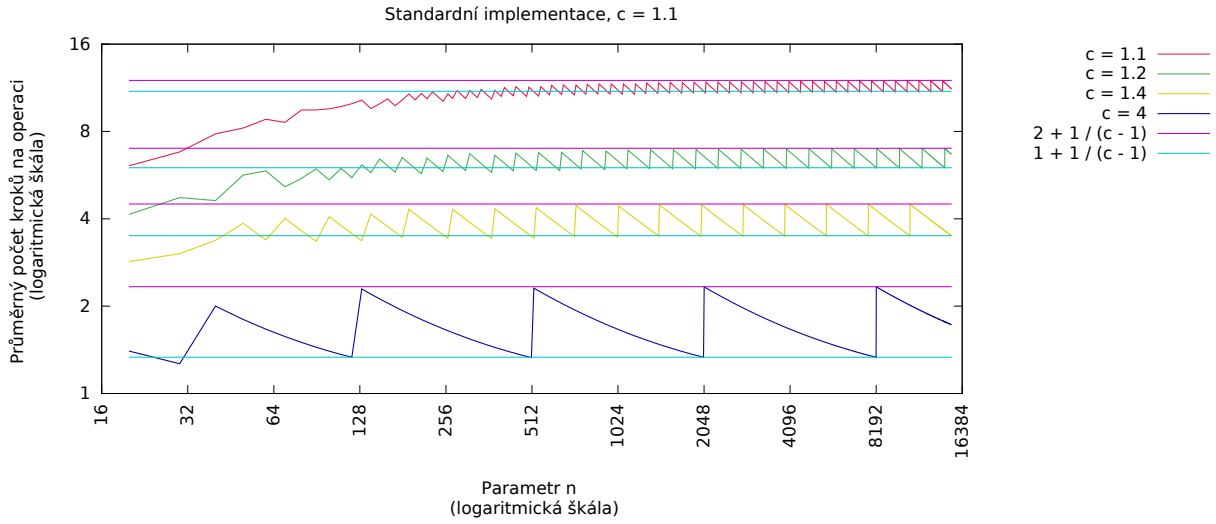


Obrázek 6: Naivní vs správná implementace v logaritmické škále.

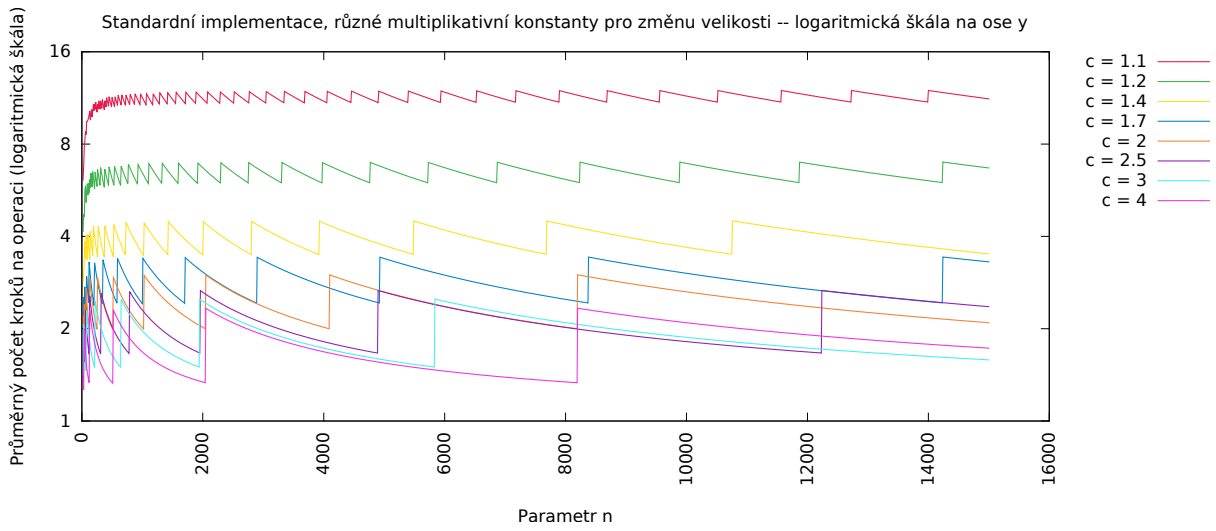
4.2 Grafy 2



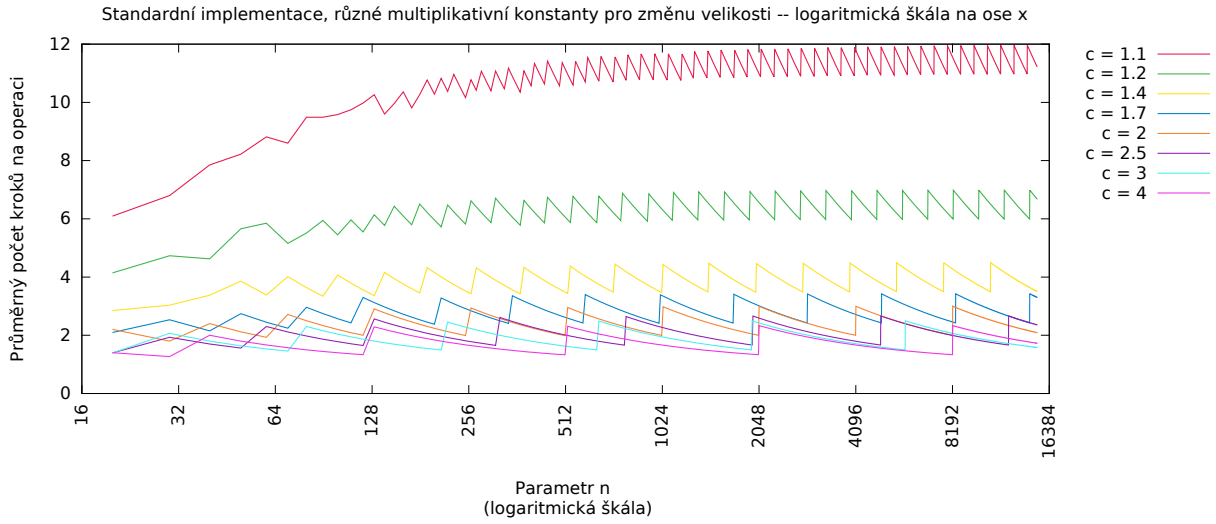
Obrázek 7: Různé zvětšování alokovaného prostoru, standardní implementace.



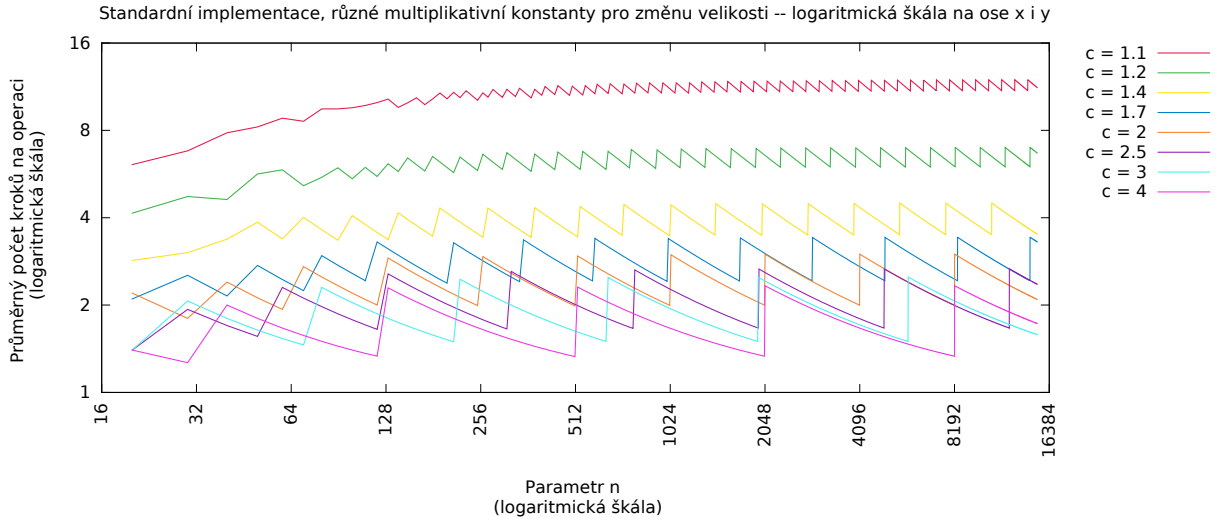
Obrázek 8: Různé zvětšování alokovaného prostoru, standardní implementace, spolu s odhady konstant.



Obrázek 9: Různé zvětšování alokovaného prostoru, standardní implementace a logaritmická škála na ose y.

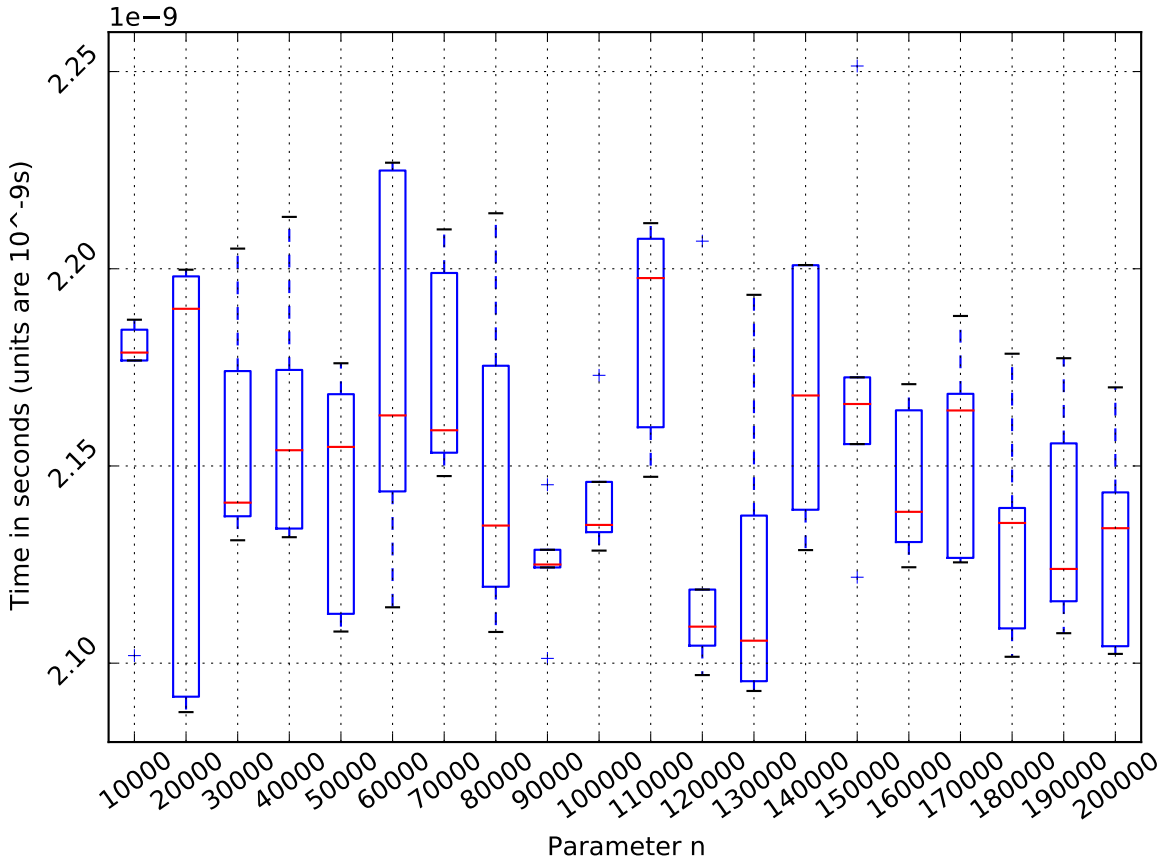


Obrázek 10: Různé zvětšování alokovaného prostoru.

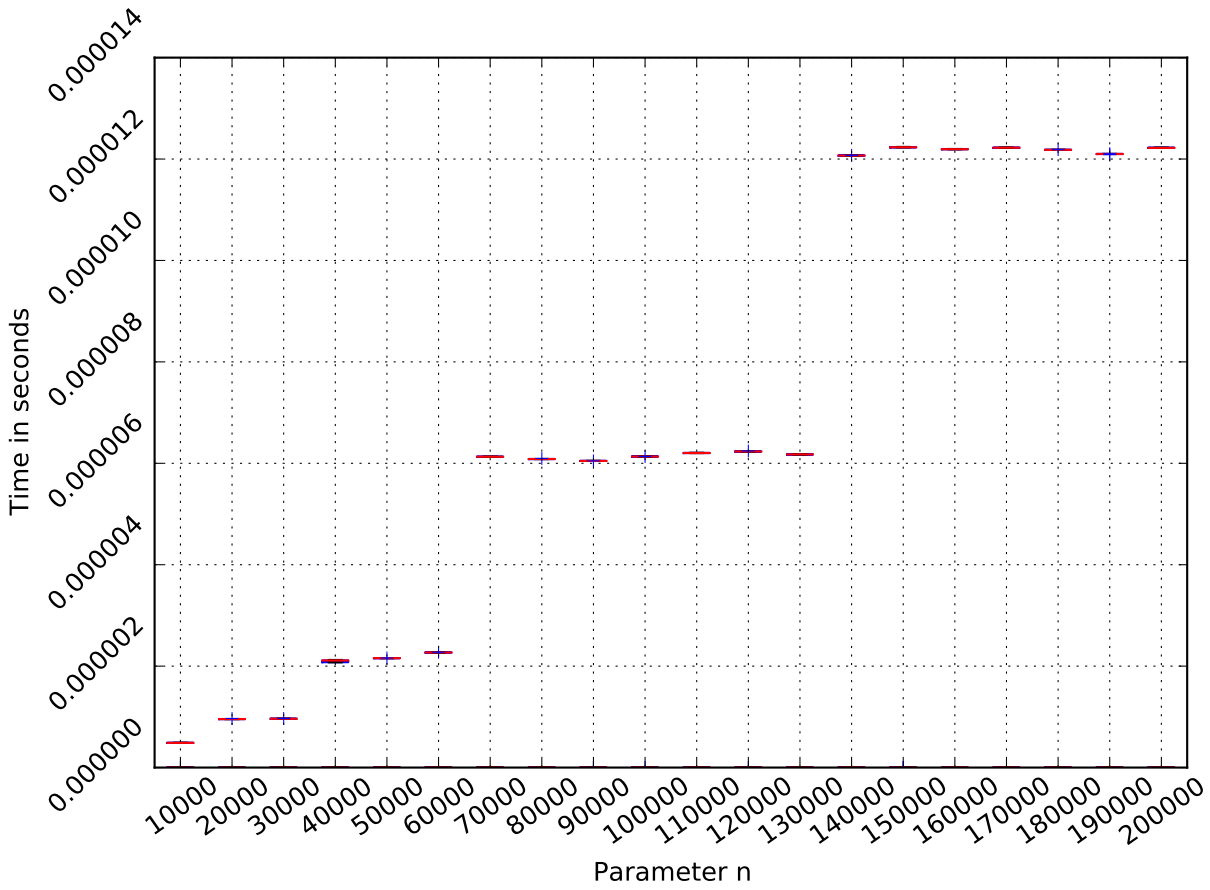


Obrázek 11: Různé zvětšování alokovaného prostoru.

4.3 Grafy čas



Obrázek 12: Standardní implementace.



Obrázek 13: Naivní implementace.