Fakulta riadenia a informatiky
Informatika

Domáce zadanie 3
Analýza výkonu ADT prioritný front

doc. Ing. **Miroslav** **Kvaššay** PhD.
STREDA 12, 13 Maroš Gorný, 5ZYI21
2021/2022

Obsah

Úloha 2

Zadanie – Overenie výkonu v scenári

Realizácie ADT prioritný front, ktoré ste implementovali v úlohe 1, otestujte v scenároch definovaných v Tab. 1. V každom scenári vykonajte spolu 100 000 operácií. Jednotlivé operácie sú v jednotlivých scenároch volané náhodne tak, aby na konci súhlasil podiel jednotlivých operácií (neplatí, že najskôr sa zavolajú operácie na vkladanie prvkov, potom operácie na mazanie prvkov a nakoniec operácie na sprístupnenie prvkov). Parametre do operácií sú taktiež náhodné; priorita je náhodné číslo z intervalu <0; 100 000>.

Tab. 1 Testovacie scenáre pre ADT prioritný front

Scenár	Podiel operácií		
	push	pop	peek
A	35	35	30
B	50	30	20
C	70	25	5

V rámci analýzy výkonu v scenári je nutné merať len dĺžku trvania vybranej operácie. To znamená, že do merania **sa nesmie započítavať čas potrebný pre generovanie pomocných údajov**

Testovanie – Implementácia

Použité inštancie a funkcie

PriorityQueue<T>* - Pointer na prioritný front s ktorým budem pracovať.
SimpleTest - Inštancia s ktorou môžem merať čas
FileLogConsumer - Pointer na Logger s ktorým budem schopný ukladať dáta.
srand(time(NULL)); - Funkcia s ktorou môžem nastaviť náhodný seed pre funkciu rand().
rand() - Funkcia ktorá vráti náhodné číslo medzi 0 a RAND_MAX.

Logika testovania

Výber vhodnej implementácie

Konečný podiel operácií má byť rovný podielu zvoleného v tabuľke. Implementácia bude teda veľmi podobná ako v predošlom zadaní. To znamená, že si náhodne vygenerujem **100 čísel** a určím si hranice, v ktorých sa má daná metóda volať.

Implementácia hraníc pre výber metódy

Hranice sú určené typom scenáru, ale pre príklad môžeme povedať, že máme **4** metódy a každá má pravdepodobnosť 25 %, preto prvá hranica bude v bode **25**, druhá v bode **50**, tretia v bode **75** a štvrtá v bode **100**.

Výber metódy

Keď sú **určené hranice** a vygenerované náhodné číslo **od 0 po 99**, môžeme určiť ktorá metóda sa zavolá.

V prípade že číslo bude pod prvou hranicou, teda bude menšie ako 25, tak sa zavolá prvá metóda. Ak číslo bude väčšie, ale bude pod hranicou 50, zavolá sa druhá metóda atď..

Taktiež musím splniť **podiel operácií**, takže budem počítať koľko krát sa mi operácia zavolala a ak dosiahla svoje maximum, potom budem vyberať prvú metódu ktorá toto maximum ešte nedosiahla **v poradí: push, peek, pop.**

Výber metódy – peek, pop (komplikácie)

V prípade metód peek a pop môže nastať komplikácia v prípade,

- Že prioritný front je **prázdny** a teda nebudeme môcť zvoliť platný index. V takomto prípade skontrolujem či môžem zavolať metódu push, ktorá bude fungovať aj pri prázdnom prioritnom fronte a ak spĺňam všetky podmienky na jej zavolanie, tak ju zavolám.
- Že prioritný front je prázdny a metóda push dosiahla svoj **maximálny počet volaní.** V takomto prípade si pri metóde
    - pop, pridám jeden nulový prvok do prioritného frontu s nulovou prioritou, pretože pri vložení jedného prvku pri prázdnom prioritnom fronte tieto parametre nehrajú žiadnu rolu a následne zavolám metódu pop pri ktorej odmeriam čas metódy pop a inkrementujem počítadlo volaní metódy pop.
    - peek, pridám jeden nulový prvok do prioritného frontu s nulovou prioritou, pretože pri vložení jedného prvku pri prázdnom prioritnom fronte tieto parametre nehrajú žiadnu rolu a následne zavolám metódu peek ktorej zmeriam čas. Po zmeraní času prvok popnem a inkrementujem počítadlo volaní metódy peek.

Tieto komplikácie mi ale zapríčinia **tvorenie zhlukov** daných metód, avšak zväčša len pri konci pretože metóda push bude mať malý percentuálny podiel a už bude plne využitá.

Volanie metódy a meranie času

Pri volaní metódy musím metódu zavolať s platným indexom. V sekcii Výber metódy – peek, pop (komplikácie) som vyriešil problém pri metóde peek a pop s veľkosťou prioritného frontu 0.

Najprv si vytvorím **náhodné dáta**, ktoré budem vkladať do metódy **push**. Tieto dáta budú obsahovať čísla od 0 po RAND_MAX.

Následne si vytvorím náhodnú prioritu, ktorá bude v intervale **od 0 po 100 000** vrátane.

Pri metóde push vkladám za parametre náhodnú prioritu a náhodné dáta a následne meriam čas danej metódy. Pri ostatných metódach meriam len čas.

Ukladanie údajov

Dáta som ukladal do CSV súboru, kde

1. Stĺpec – **počet** prvkov v prioritnom fronte po zavolaní metódy
2. Stĺpec – **čas** metódy v nanosekundách
3. Stĺpec – volaná **metóda**

Size	Time[ns]	Method
1	2200	push
2	2000	push
3	2600	push
2	3000	pop
3	2300	push
4	2300	push
5	5300	push

Testovanie – Výsledky

Na vyhodnotenie daných výsledkov som použil kontingenčnú tabuľku, kde na **X osi je veľkosť prioritného frontu** a na **osi Y je priemerný čas metód** v nanosekundách [ns].

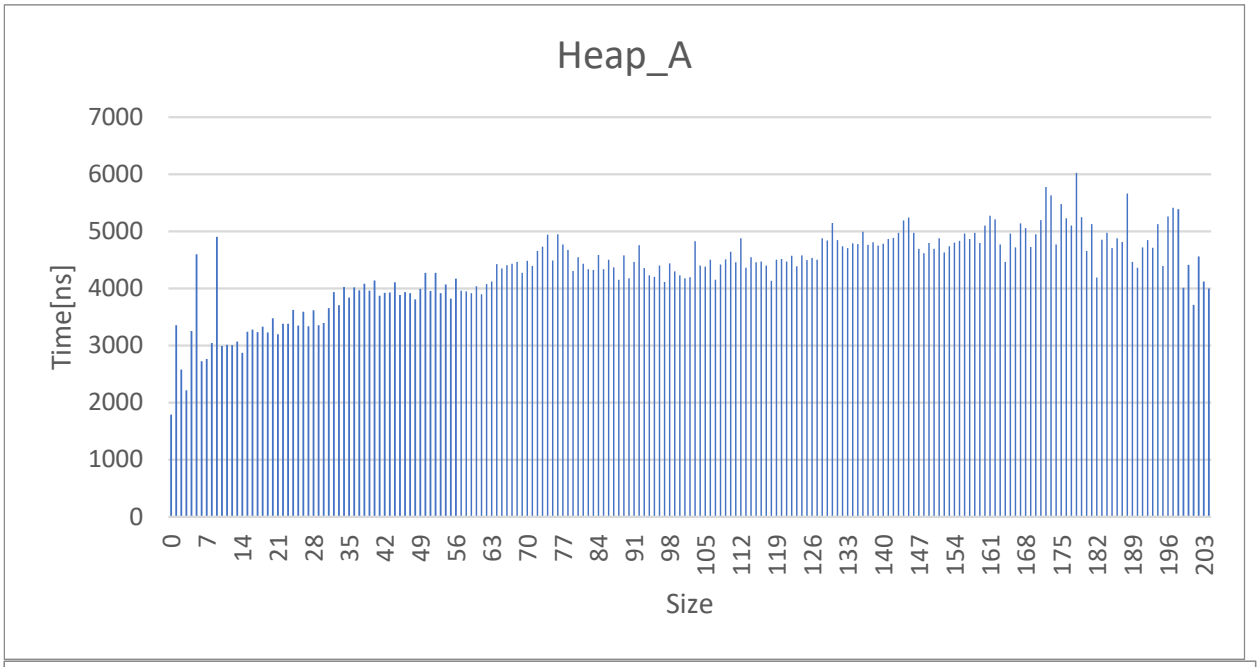
Tým pádom môžem porovnať scenáre ako celok a nemusím sa pozerať na jednotlivé metódy. Pozerám sa na ich spoločný priemer, takže porovnávam celkovú rýchlosť prioritného frontu a nie jeho najlepšiu alebo najhoršiu zložitosť.

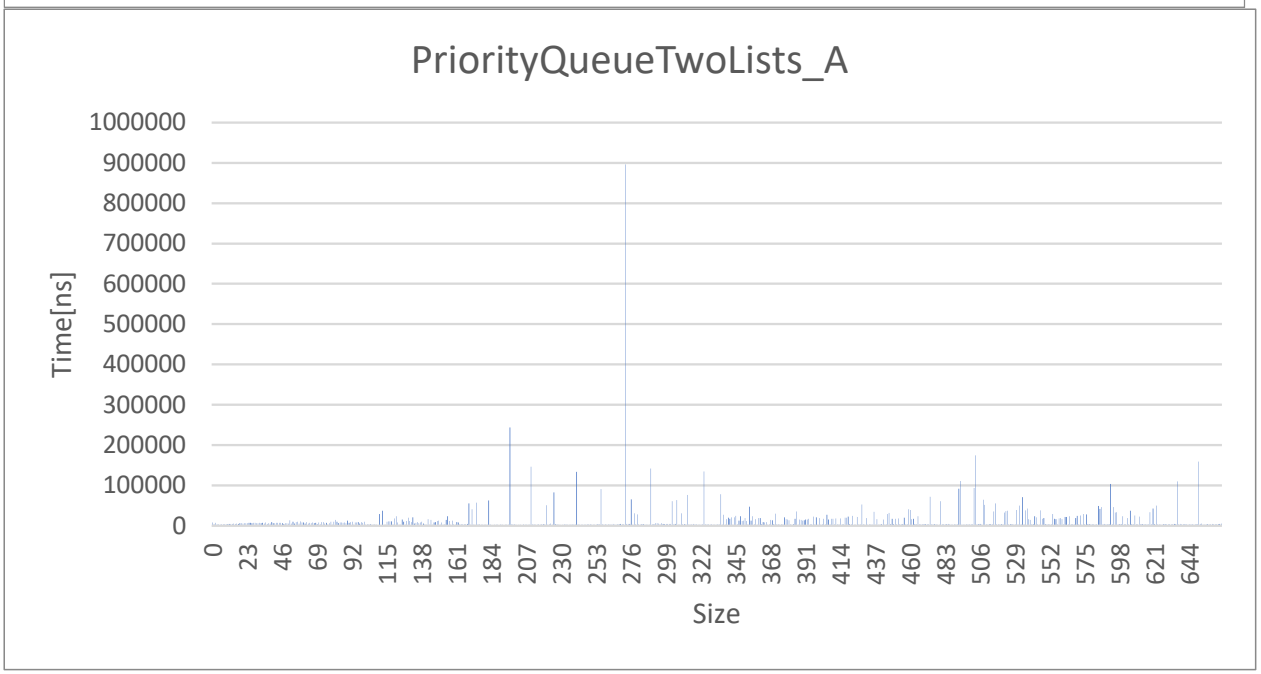
Scenár A

Zadanie

Scenár	Podiel operácií		
	push	pop	peek
A	35	35	30

Prezentácia výsledkov





Vyhodnotenie výsledkov

Na základe grafov môžeme vidieť, že

Heap

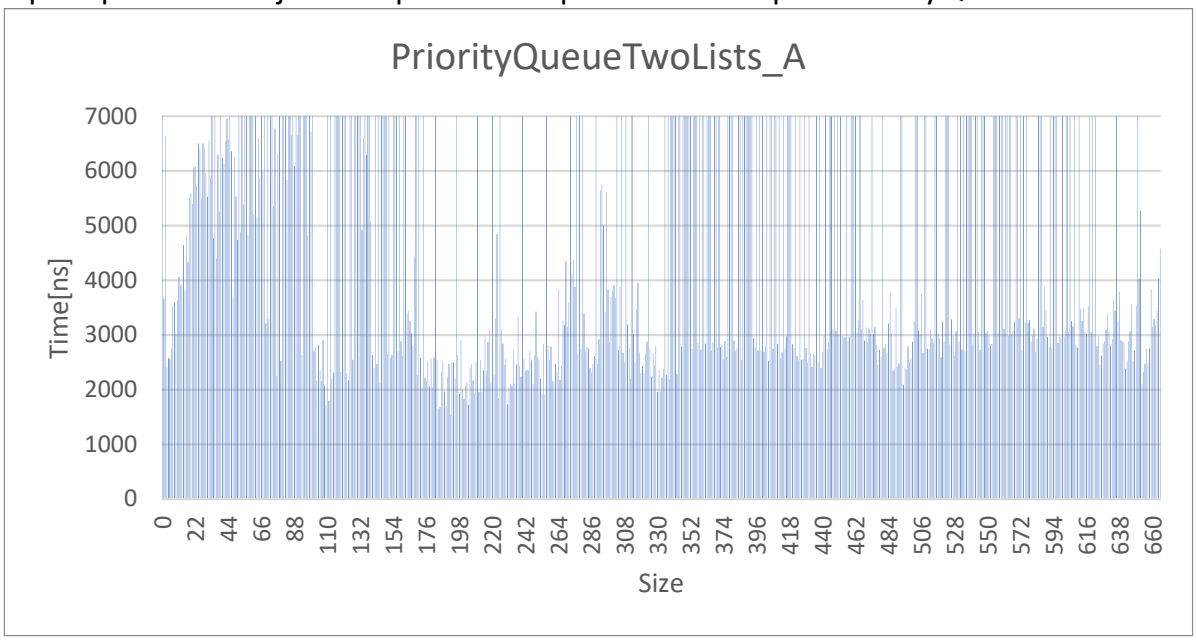
- Ako celok mu **priemerná rýchlosť klesá**, teda čím väčší tento prioritný front je, tým je rýchlosť pomalšia.
- Môžeme však vidieť, že zo začiatku rýchlosť klesá trošku rýchlejšie a následne sa klesanie rýchlosti spomaľuje.

PriorityQueueTwoLists

- Ako celku sa mu **priemerná rýchlosť raz za čas veľmi spomalí** , teda veľkú časť spracuje relatívne rýchlo, avšak občas potrebuje X násobný čas pre reštrukturalizáciu.
- Pri danom grafe vidíme, že pri veľkosti približne 276 prioritný front potreboval viac času. Po tomto spracovaní sa však rýchlosť znova zlepšila a len raz za čas sa znova minimálne zhoršila.

Heap vs PriorityQueueTwoLists

- V danom scenári je **rýchlejší Heap** kvôli tomu, že vôbec nejde do extrémov. Pre lepšie porovnanie je však potrebné upraviť mierku pre PriorityQueueTwoLists



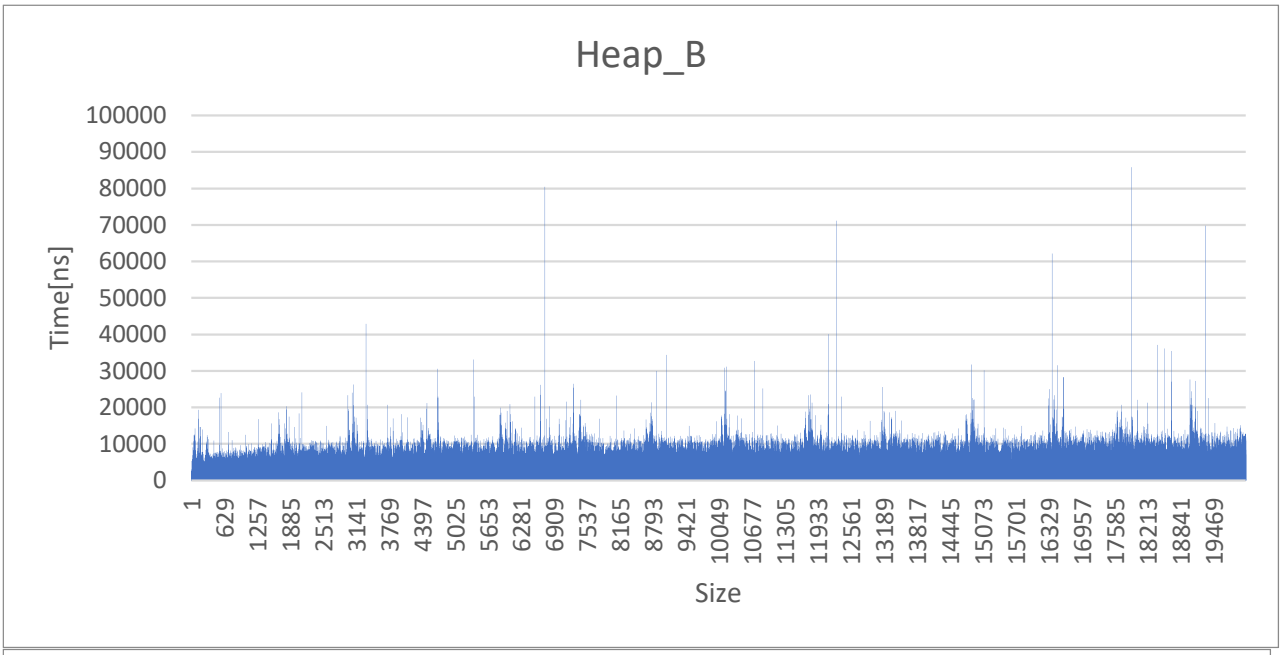
- Tu môžeme vidieť, že raz za čas potrebuje PriorityQueueTwoLists čas na **reštrukturalizáciu** ktorá je ale nesmierne **pomalá** oproti rýchlosti Heap-u. Aj napriek tomu, že Heap dosiahol len približne 200 prvkov, je vidno, že rýchlosť sa pohybuje okolo trendu a neosciluje. Za to pri PriorityQueueTwoLists osciláciu vidíme. Keby extrémy odstránime, môžeme si všimnúť, že PriorityQueueTwoLists sa správa rýchlejšie ako Heap.

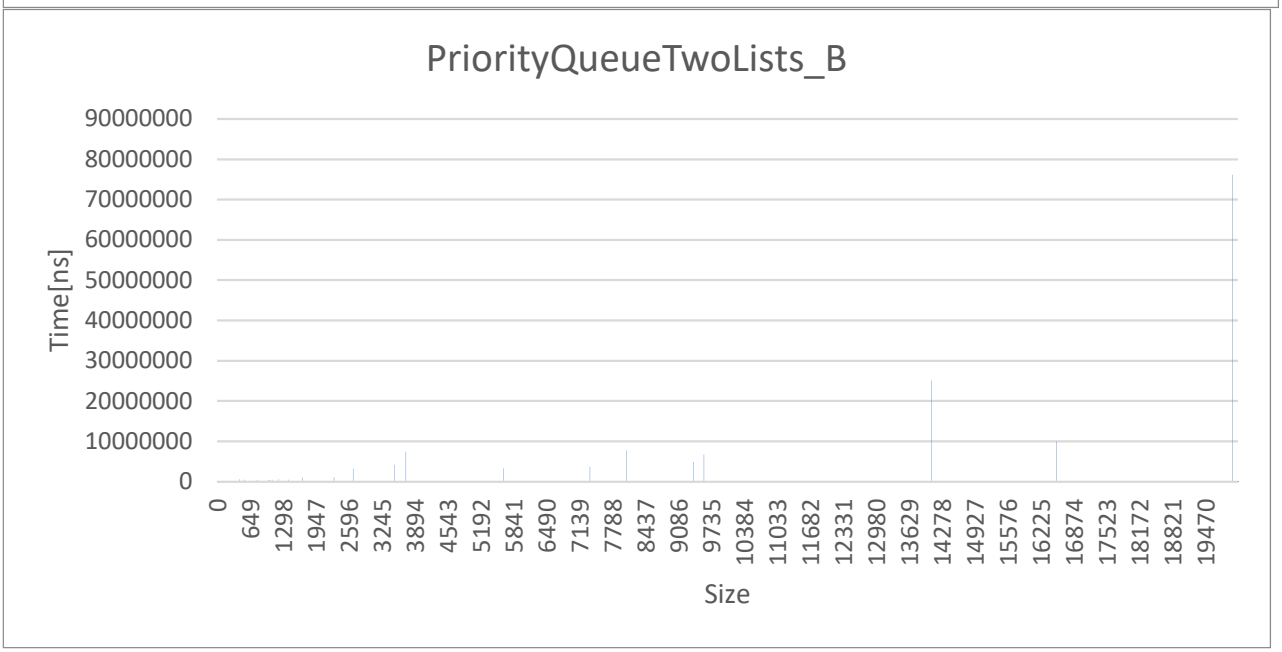
Scenár B

Zadanie

Scenár	Podiel operácií		
	push	pop	peek
B	50	30	20

Prezentácia výsledkov





Vyhodnotenie výsledkov

Na základe grafov môžeme vidieť, že

Heap

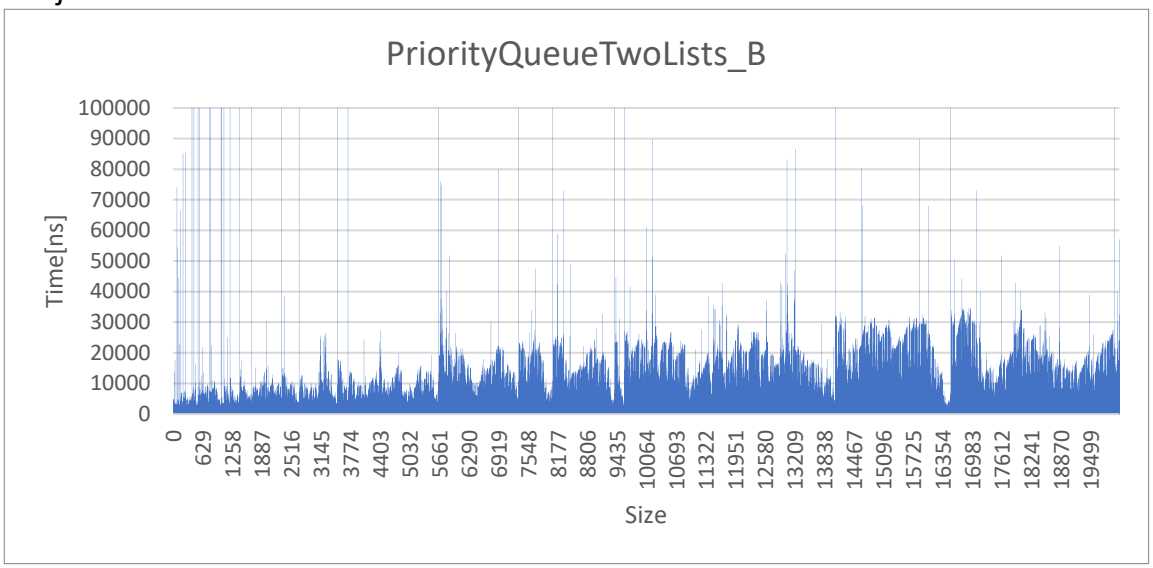
- Vyzerá to tak, že ako celok má **priemernú rýchlosť rýchlu a po určitom počte prvkov takmer konštantnú,** avšak niekedy tam je vidno osciláciu.

PriorityQueueTwoLists

- **Priemerná rýchlosť vyzerá veľmi rýchla**, avšak občas tam je výrazné spomalenie.

Heap vs PriorityQueueTwoLists

- Na presnejšie porovnanie potrebujeme nastaviť podobnú mierku zobrazenia údajov.



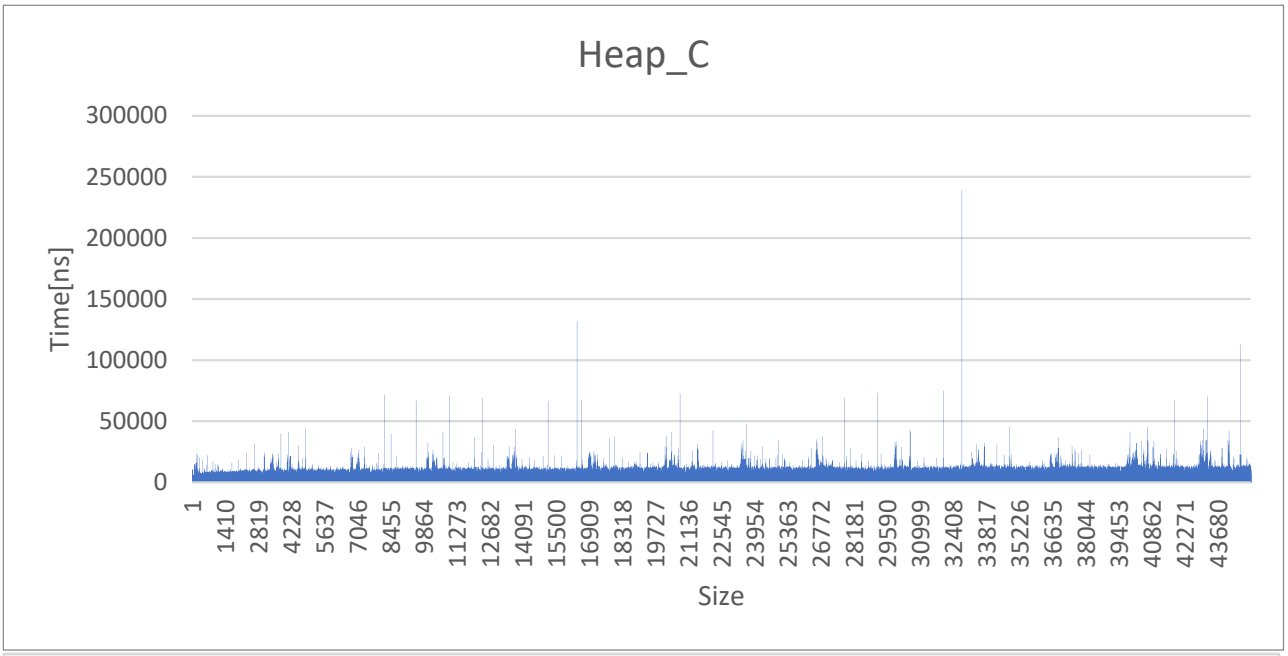
- Tu už môžeme vidieť, že rýchlosť niekedy klesne pod 10 000 ns, avšak tiež nad túto hranicu stúpne. Čo ale môžeme vidieť z prvého grafu je to, že občas sa tá rýchlosť spomalí extrémne. Preto je v danom scenári určite rýchlejší Heap ktorému sa tiež rýchlosť raz za čas spomalí, avšak spomalenie nie je až také extrémne ako pri PriorityQueueTwoLists. Pri danom scenári však existujú situácie pri ktorých by bol rýchlejší PriorityQueueTwoLists.

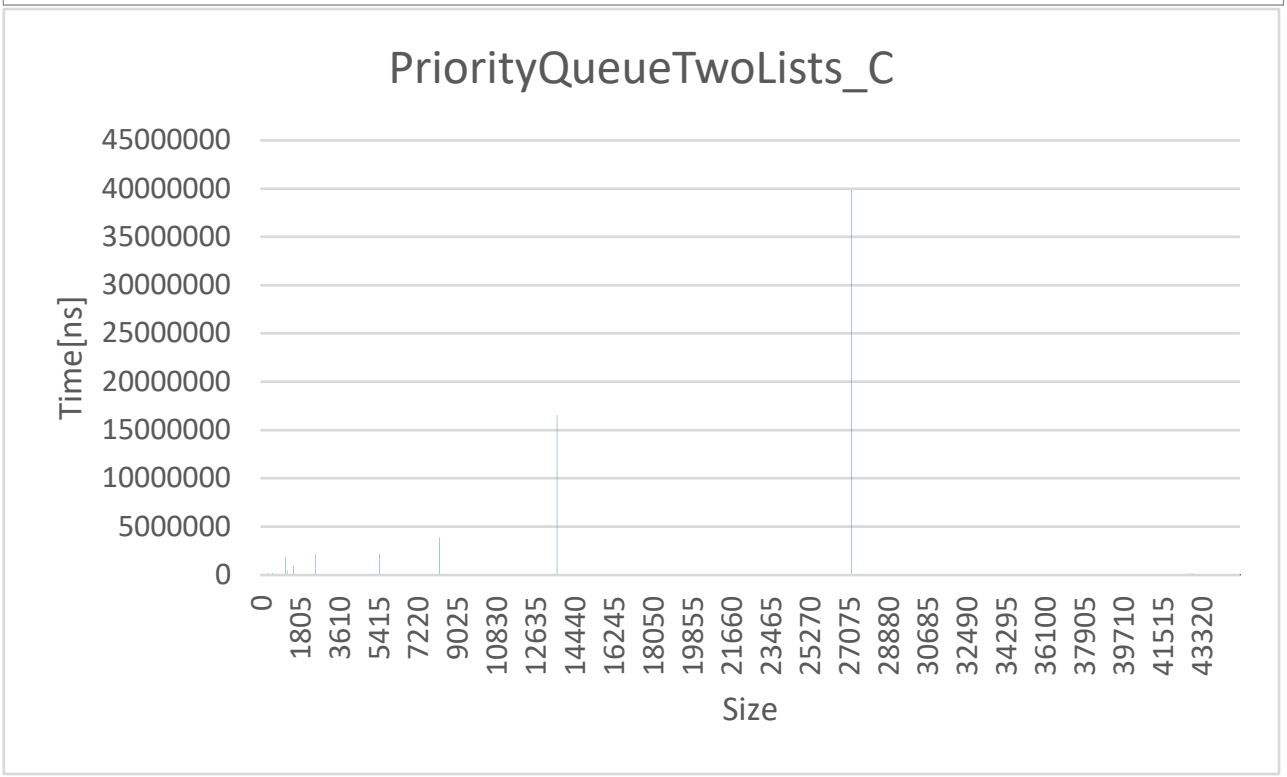
Scenár C

Zadanie

Scenár	Podiel operácií		
	push	pop	peek
C	70	25	5

Prezentácia výsledkov





Vyhodnotenie výsledkov

Na základe grafov môžeme vidieť, že

Heap

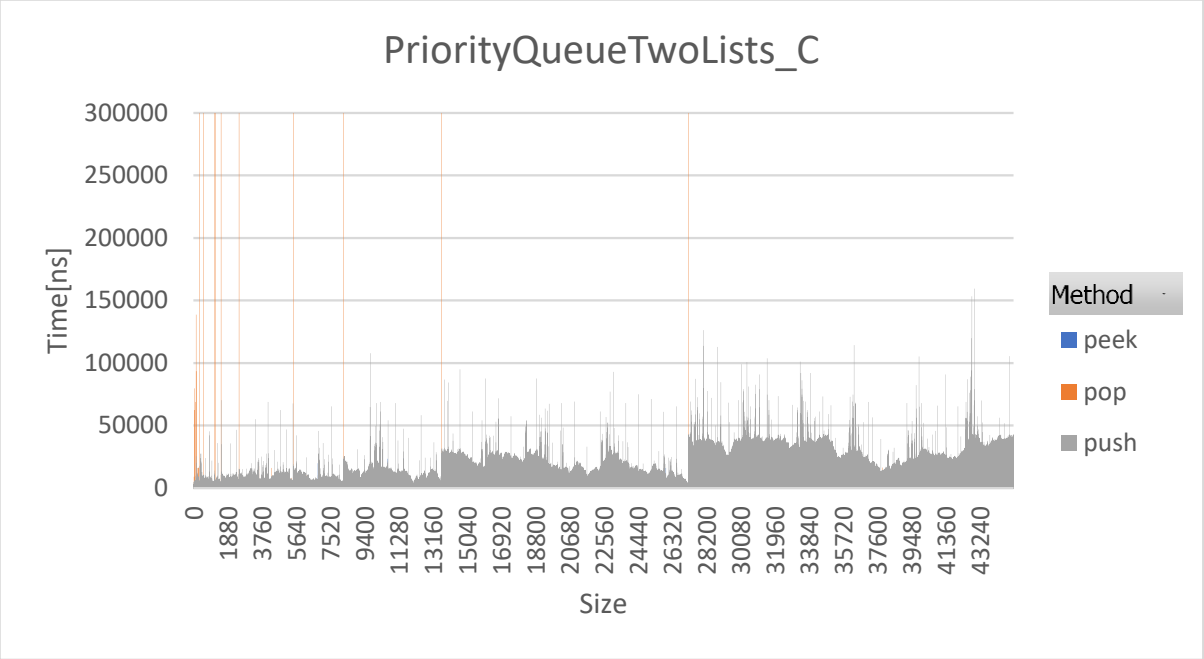
- Vyzerá to tak, že ako celok má **priemernú rýchlosť takmer konštantnú,** avšak niekedy tam je vidno osciláciu.

PriorityQueueTwoLists

- **Priemerná rýchlosť vyzerá veľmi rýchla**, avšak občas tam je výrazné spomalenie.

Heap vs PriorityQueueTwoLists

- Na presnejšie porovnanie potrebujeme nastaviť podobnú mierku zobrazenia údajov. Pre lepšie pochopenie je pridané aj rozdelenie metód.



- Ako môžeme vidieť, aj tu je priemerná rýchlosť oproti Heap-u pomalšia. Heap raz za čas potrebuje viac času na vykonanie metódy, ale potrebuje ho menej krát a aj ten čas na vykonanie metódy je menší. Pri PriorityQueueTwoLists metóda pop zaberie veľmi veľa času, avšak následne po nej je priemerná rýchlosť prijateľná.

Záver

Vďaka testom môžeme vidieť, že ak je v prioritnom fronte viac prvkov, tak bez ohľadu na typ metód ktoré budeme používať častejšie, tak je **vhodnejšie používať Heap.**

Môžeme si ale všimnúť, že pri niektorých situáciách bol PriorityQueueTwoLists rýchlejší, hlavne po tom ako uskutočnil reštrukturalizáciu, tak nejaký čas pracoval veľmi rýchlo.

V konečnom dôsledku by sme teda mohli vyvodiť záver taký, že ak by som bez ohľadu na situáciu potreboval rýchly prioritný front, vyhral by Heap. Ak by mi ale občasné spomalenie rýchlosti nevadili, vhodnejší vyzerá byt PriorityQueueTwoLists.

Úloha 3

Zadanie – Analýza časových zložitostí

V rámci analýzy časových zložitostí je nutné otestovať rýchlosť operácií push, pop a peek v závislosti od počtu prvkov a implementácie prioritného frontu a na základe nameraných a spracovaných údajov **odhadnúť hornú asymptotickú zložitosť jednotlivých operácií**.

V rámci analýzy časových zložitostí vybraných operácií je nutné merať len dĺžku trvania vybranej operácie. To znamená, že do merania **sa nesmie započítavať čas potrebný pre generovanie pomocných údajov**.

Testovanie – Implementácia

Použité inštancie a funkcie

PriorityQueue<T>* - Pointer na prioritný front s ktorým budem pracovať.
SimpleTest - Inštancia s ktorou môžem merať čas
FileLogConsumer - Pointer na Logger s ktorým budem schopný ukladať dáta.
srand(time(NULL)); - Funkcia s ktorou môžem nastaviť náhodný seed pre funkciu rand().
rand() - Funkcia ktorá vráti náhodné číslo medzi 0 a RAND_MAX.

Logika testovania

Testovanie budem robiť tak, že zavolám metódu **100 000 krát**.

- Pri metóde **push** pôjde veľkosť od 0 po 99 999.
- Pri metóde **peek** pôjde veľkosť od 1 po 100 000.
    - Teda pred každým volaním sa veľkosť zväčší o 1.
- Pri metóde **pop** pôjde veľkosť od 100 000 po 1.

Ukladanie údajov

Také isté ukladanie ako v úlohe 2, teda dáta som ukladal do CSV súboru, kde

1. Stĺpec – **počet** prvkov v prioritnom fronte
2. Stĺpec – **čas** metódy v nanosekundách
3. Stĺpec – **volaná metóda**

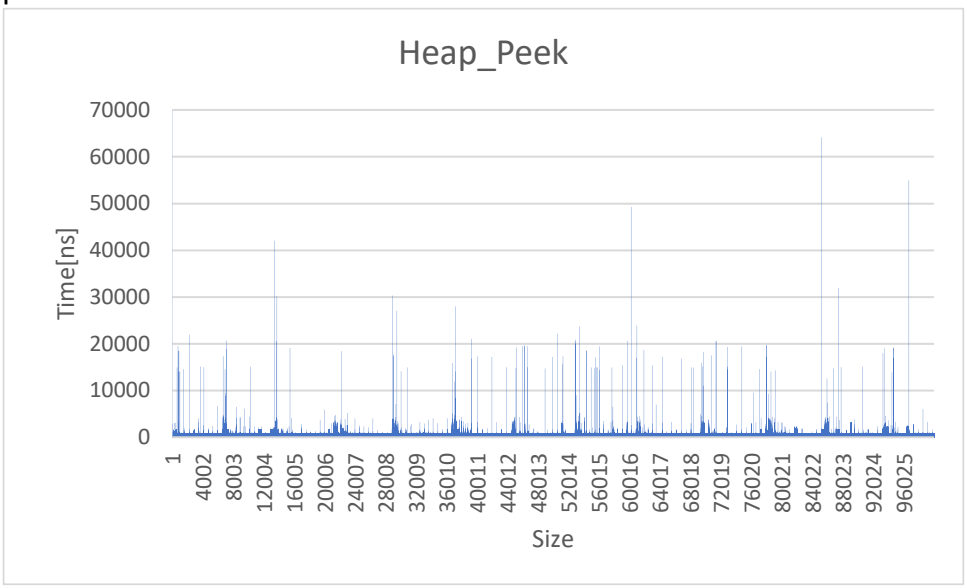
Size	Time[ns]	Method
1	491600	peek
2	2900	peek
3	900	peek
4	900	peek
5	900	peek
6	900	peek

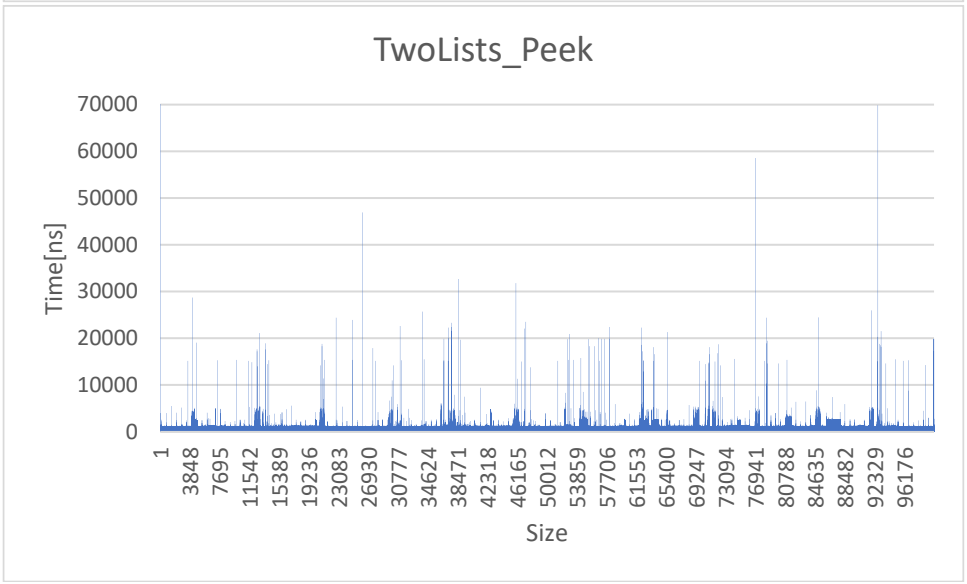
Testovanie – Výsledky

Na vyhodnotenie daných výsledkov som použil **column graph** , kde na **X** osi je **veľkosť prioritného frontu** a na **Y** osi je **čas** v nanosekundách.

Peek

Môžeme vidieť, že obidva prioritné fronty sú rýchle a rýchlosť je pri väčšine času **konštantná**. Niekedy mierne vyskočí, avšak implementácia je spravená tak, aby rýchlosť bola konštantná a teda odhadujem, že tieto spomalenia sú spôsobené procesmi na počítači.



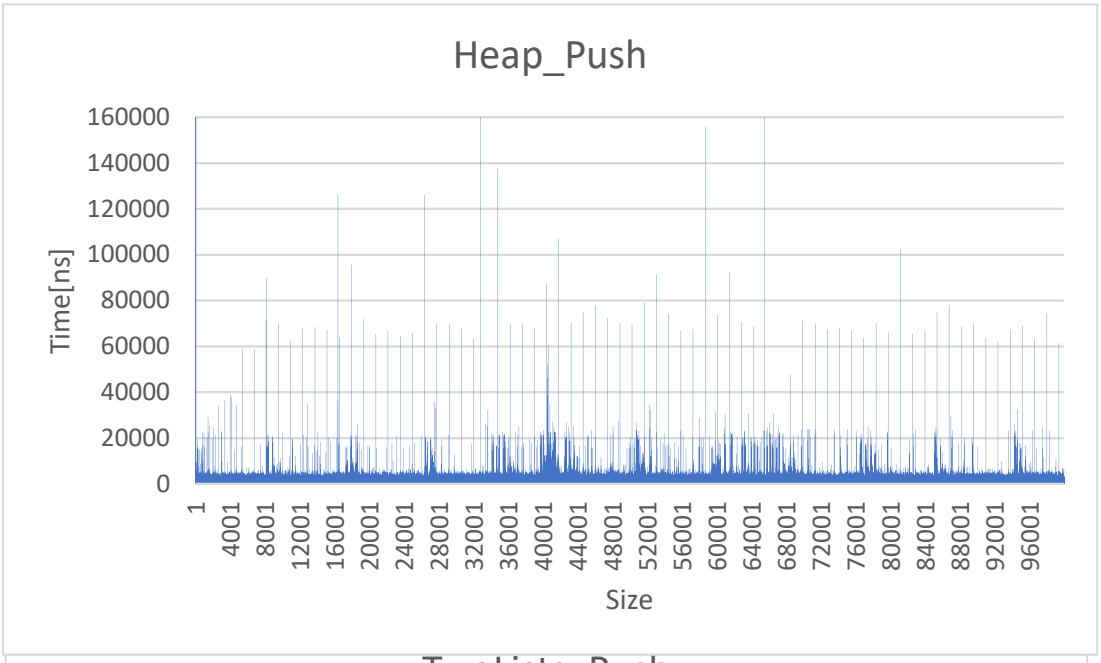


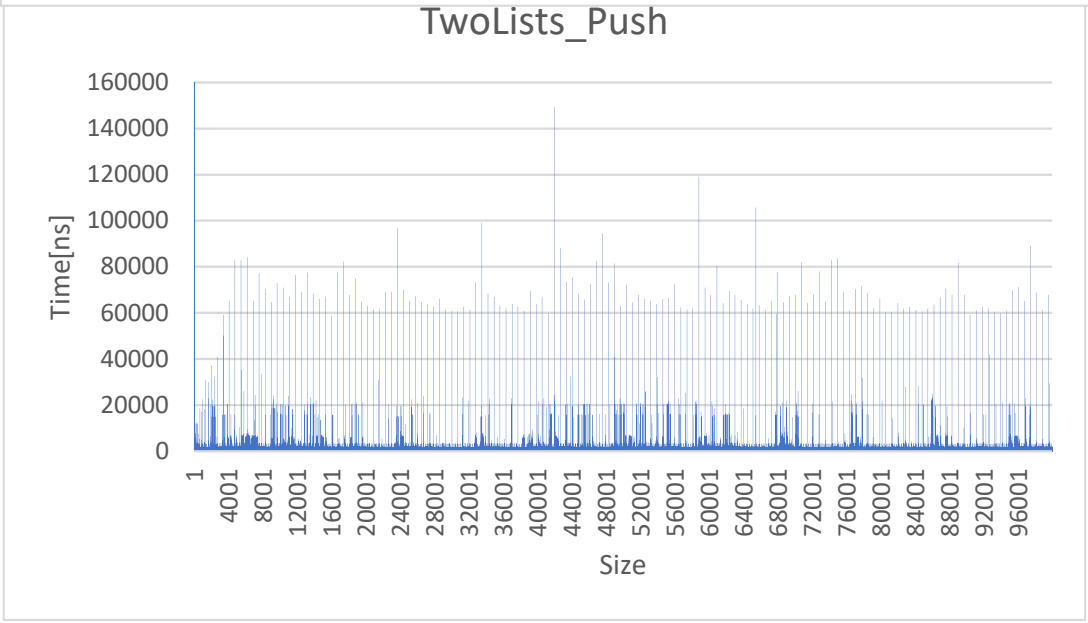
Push

Môžeme vidieť, že pri obidvoch metódach je rýchlosť rýchla, ale raz za čas sa spomalí.

- Pri Heap sa rýchlosť spomalí keď sa vloží prvok s najväčšou prioritou a teda sa prvok musí **prebublať** na správne miesto.
- Pri TwoLists je problém pri prvku ktorý ma prioritu o trošku menšiu ako najhorší prvok v krátkom zozname a preto sa musí **posúvať** celý list.

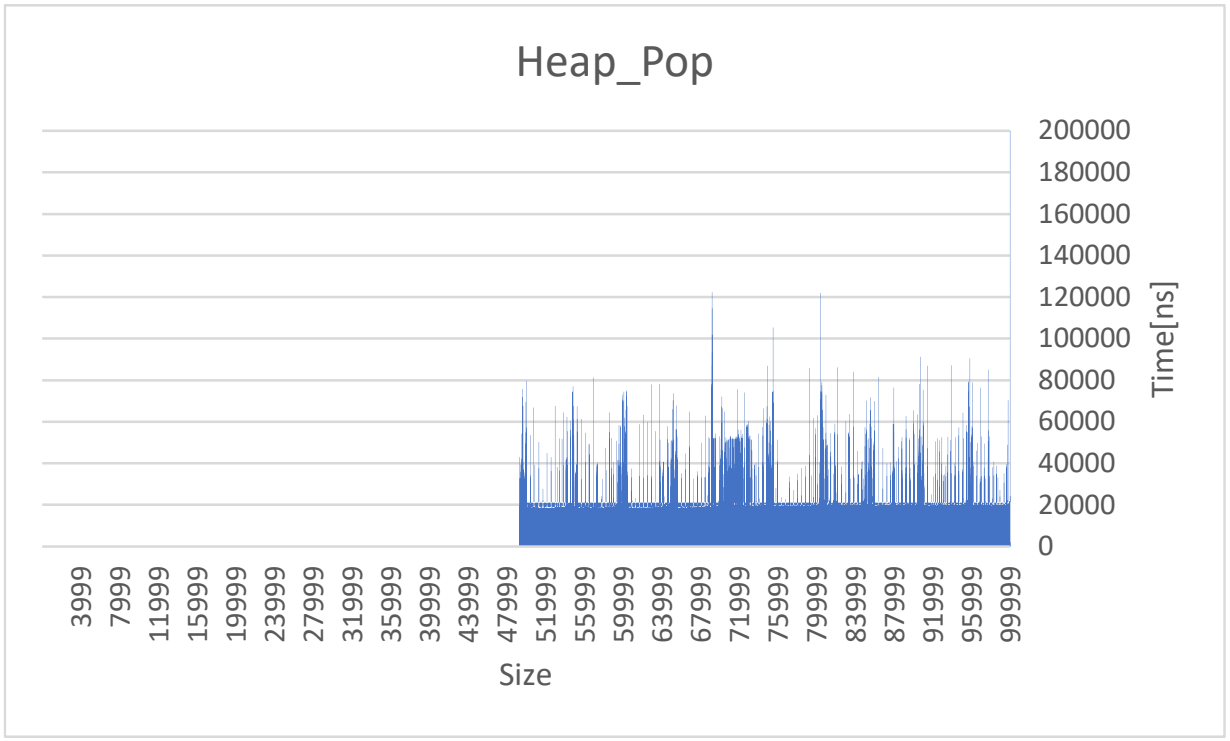
Zložitosť sa na seba veľmi podobá, avšak musíme si uvedomiť, že zatiaľ čo pri Heap sa vložený prvok posúva na celej **jednej vetve ktorá ide od koreňa**, tak pri TwoLists sa **posúva** len **po veľkosti krátkeho zoznamu**, alebo sa pridá do dlhého zoznamu. Ak sa ale posunie na 0 miesto, pri najhoršom sa posunie krátky zoznam **pri inserte a pri vrátení prvku** ktorý **presiahol kapacitu**. Preto zložitosť push pri **TwoLists je 2*n**, kde **n** je **veľkosť krátkeho zoznamu**. Pri danej implementácii je však veľkosť krátkeho zoznamu vždy maximálne 2, pretože **nerobím reštrukturalizáciu**. Pri **Heap** je zložitosť **logaritmická**.

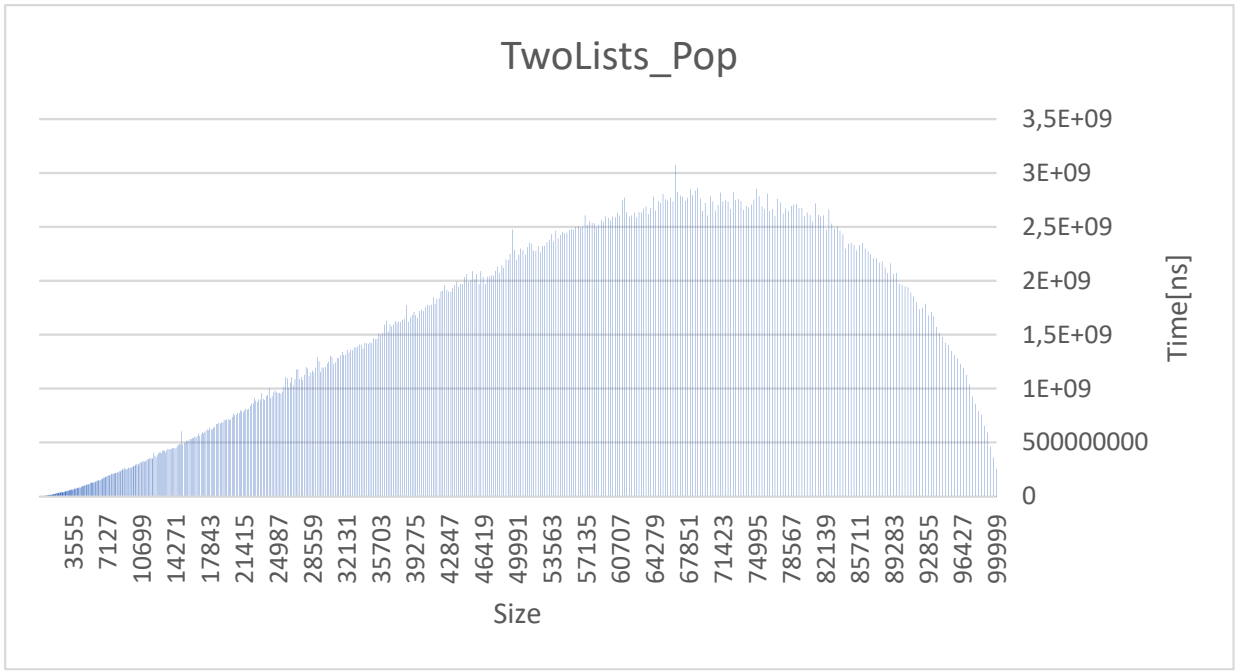




Pop

Tu je značne **rýchlejší Heap**.
Pri Heap vidíme jasne logaritmickú zložitosť ktorá je oveľa rýchlejšia ako rýchlosť v TwoListe. Pri TwoListe vidíme, že často krát sa rýchlosť pohybuje pri minime, avšak častejšie je veľmi pomalá kvôli reštrukturalizácii. Z grafu neviem úplne presne určiť o akú zložitosť ide, ale pri najhoršom prípade to bude sqrt(n) * n.





Záver – Odhad hornej asymptotickej zložitosti

Záver

Na základe grafov by som sa pri **vkladaní** prvkov pri **väčšom množstve** rozhodol pre **TwoList**, pri **sprístupnení** prvkov sú obidva prioritné fronty rovnaké a pre **vybranie prvku** by som sa jednoznačne rozhodol pre **Heap.**

Peek

- Heap
  - **O(1)**
- TwoLists
  - **O(1)**

Push

- Heap
  - **O(log(n))**
- TwoLists
  - **O (n) - kde n je veľkosť krátkeho zoznamu**

Pop

- Heap
  - **O(log(n))**
- TwoLists
  - Predpoklad: **O(sqrt(n) * n) - kde n je celkový počet prvkov**

Úloha 4 (bonusová úloha)

Cieľom bonusovej úlohy je otestovať vplyv dĺžky kratšieho zoznamu na výkon prioritného frontu implementovaného ako dvojzoznam (využite implementáciu z úlohy 1). Pri testovaní je nutné porovnať nasledujúce stratégie pre definovanie dĺžky kratšieho zoznamu (čiuž pri vzniku štruktúry alebo pri jej reštrukturalizácii):

- konštantná dĺžka definovaná ako 1/1000 počtu všetkých vložení do prioritnéhofrontu;
- variabilná dĺžka definovaná ako √n, kde n je počet prvkov v prioritnom fronte;
- variabilná dĺžka definovaná ako n/2, kde n je počet prvkov v prioritnom fronte.

Uvedené stratégie porovnajte s využitím scenárov definovaných v úlohe 2. Pre umožnenie menenia dĺžky kratšieho zoznamu na základe rôznej stratégie môžete využiť podobné prístupy, ako boli uvedené v bonusovej úlohe v domácom zadaní 2.

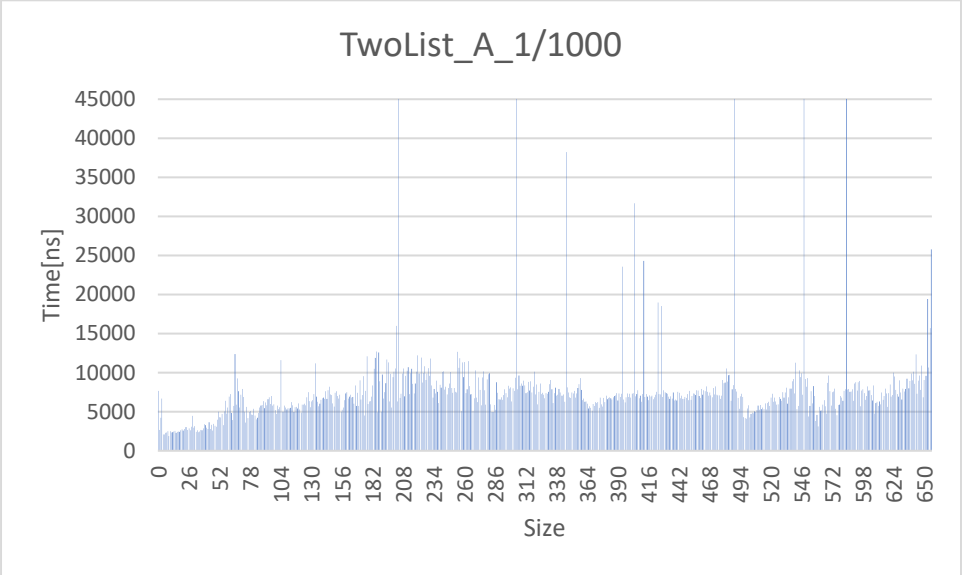
Testovanie a reprezentácia výsledkov

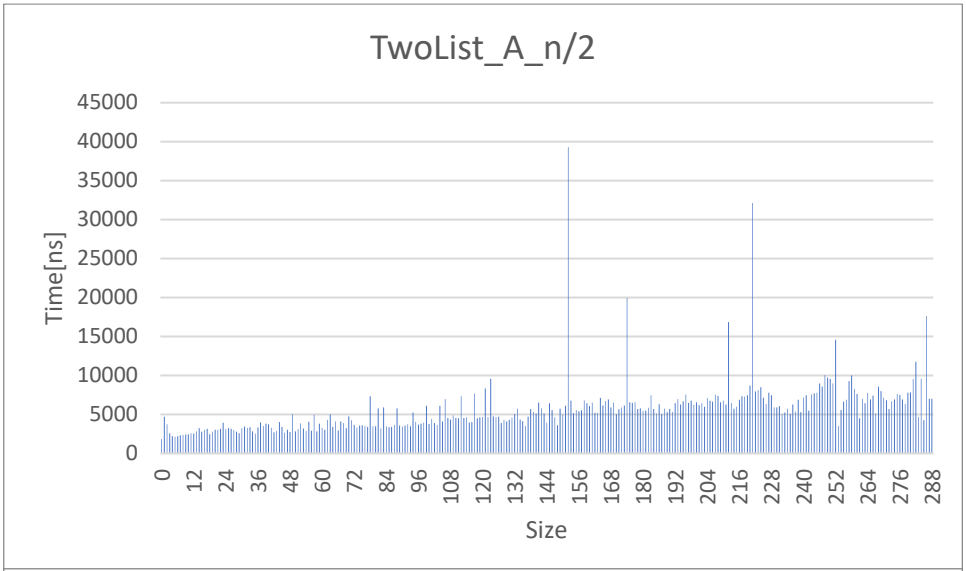
Priebeh testovania bude prebiehať podľa zadania a teda cez použitie scenárov z úlohy 2. Bude sa však meniť dĺžka kratšieho zoznamu v TwoList.

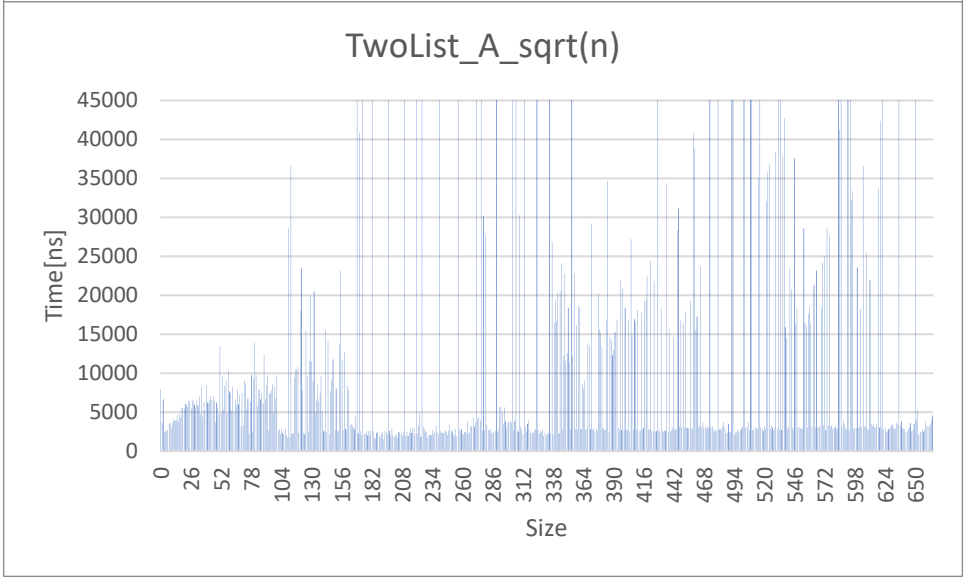
Reprezentácia dát bude zoskupená podľa scenárov. Na lepšie porovnanie budem používať tú istú mierku, ktorá mi však niekedy môže odseknúť dlhšie trvanie metód. Mierku ale spravím optimálnu pre všetky tri scenáre, takže ak by som mal dáta 3,6 a 9, tak mierku zvolím 6.

Reprezentácia výsledkov

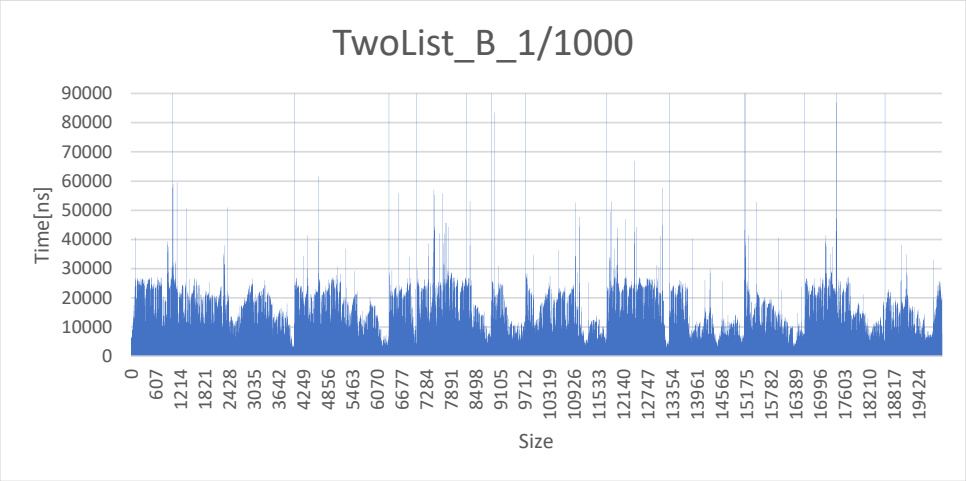
Scenár A – Reprezentácia výsledkov

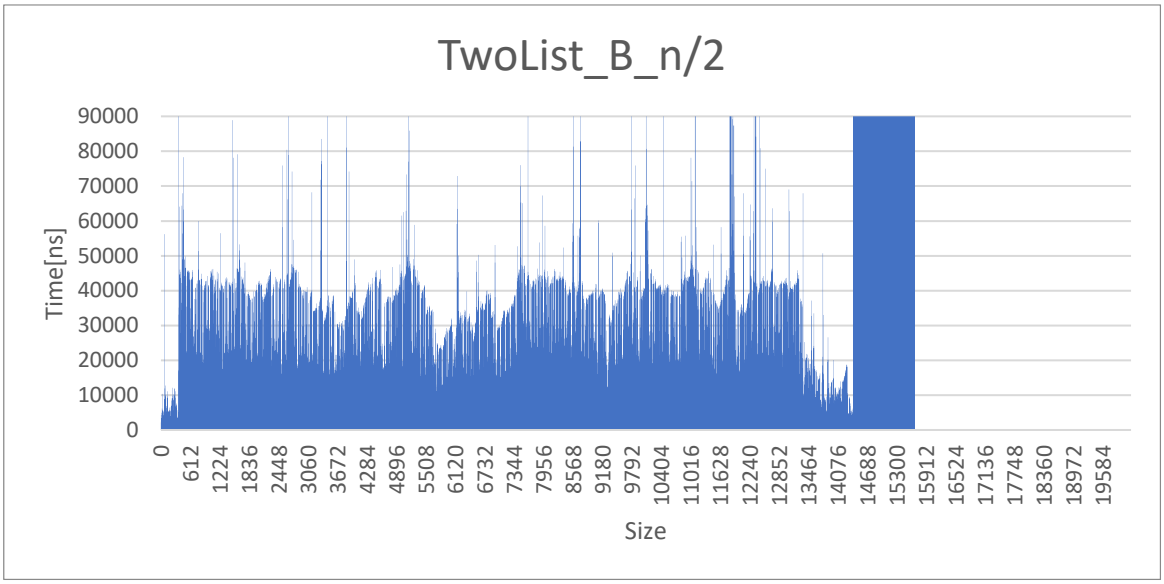


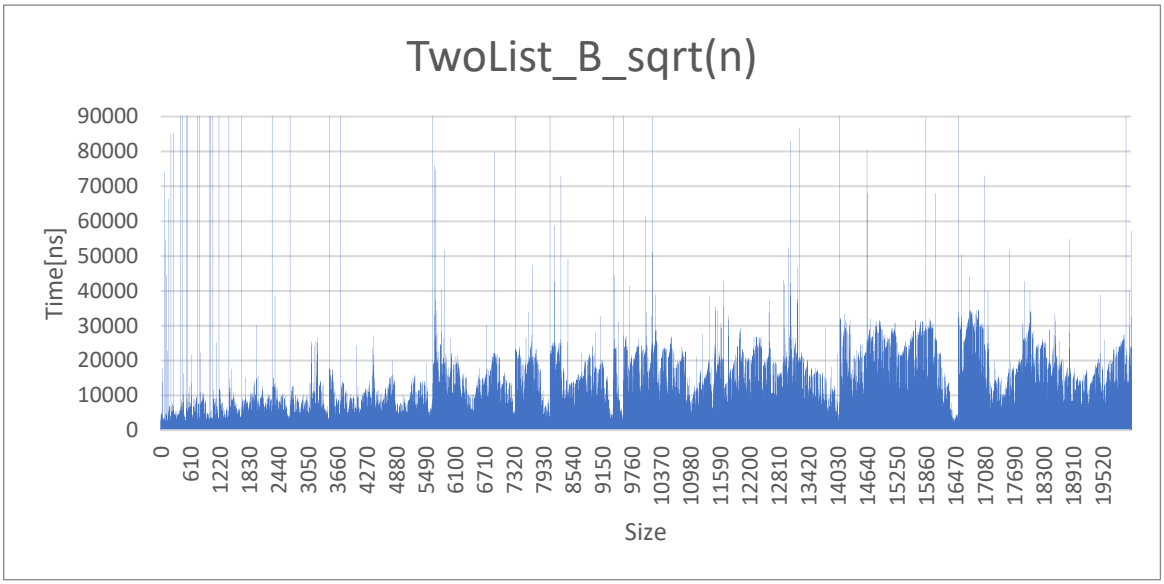




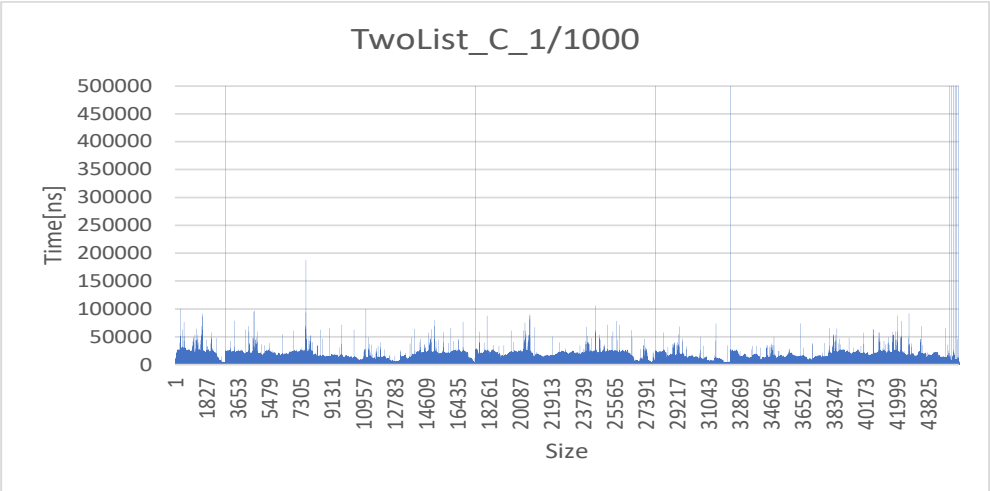
Scenár B – Reprezentácia výsledkov

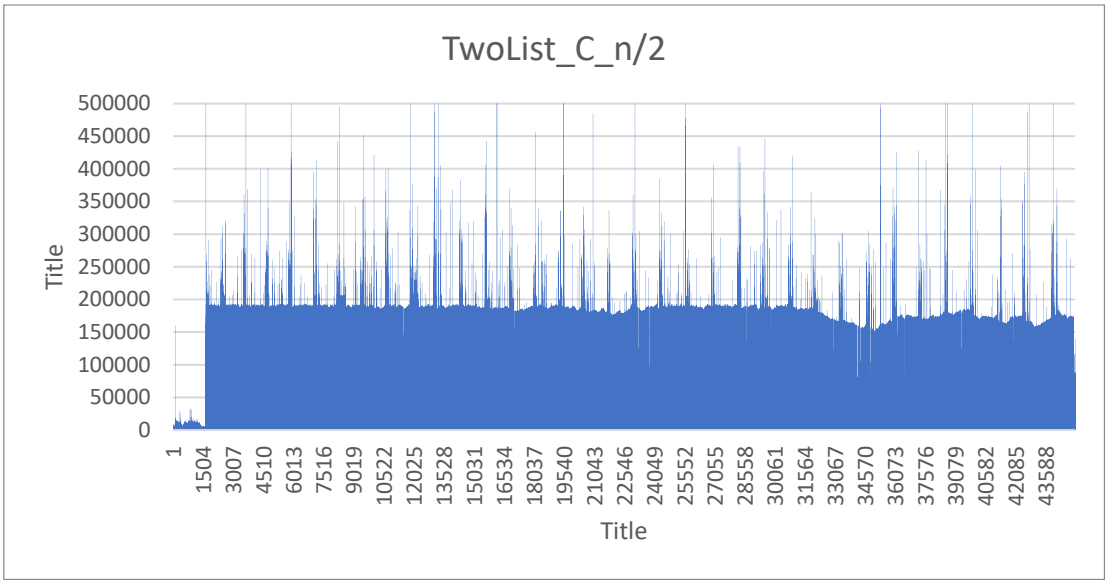


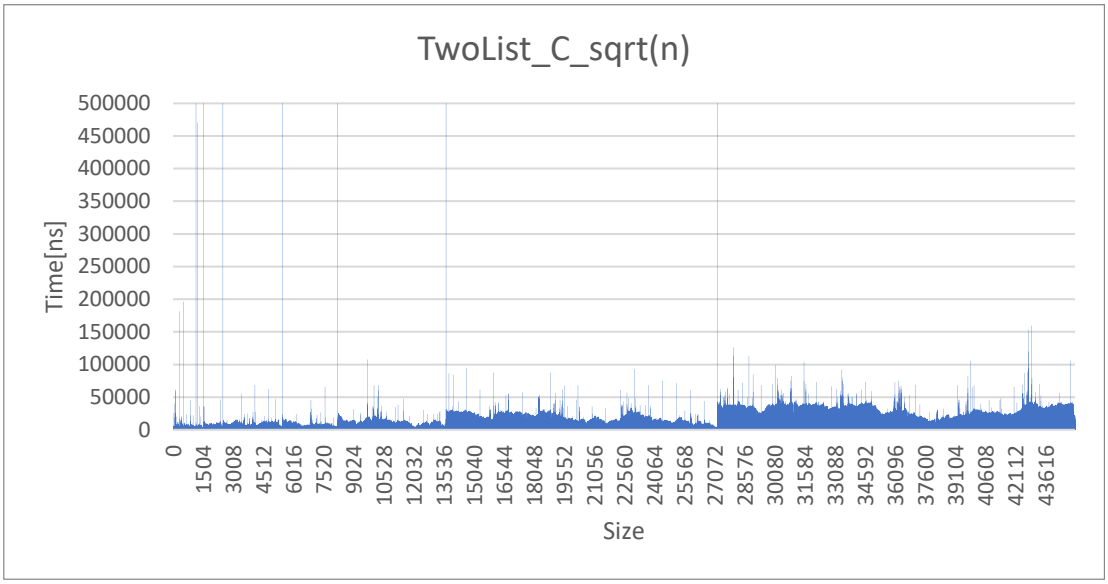




Scenár C – Reprezentácia výsledkov







Vyhodnotenie výsledkov

Scenár A – Vyhodnotenie

Pri **menšom počte dát** pôsobí **rýchlejšie** variabilná dĺžka definovaná ako **n/2** (n je počet prvkov v prioritnom fronte) a konštantná dĺžka definovaná ako **1/1000** počtu všetkých vložení.

Pri **väčšom počte dát** vidíme **amortizovanú zložitosť** pri dĺžke definovanej ako **√n**, kde n je počet prvkov v prioritnom fronte. Rýchlosť je teda dobrá, avšak niekedy vystrelí.

Scenár B – Vyhodnotenie

Môžeme si všimnúť, že **rýchlosť je relatívne dobrá** pri konštantnej dĺžke definovanej ako **1/1000** počtu a pri dĺžke definovanej ako **√n**, kde n je počet prvkov v prioritnom fronte.

Dĺžka 1/1000 sa tvári tak, že tá rýchlosť je v podstate celkom podobná bez ohľadu na veľkosť, aj keď ku koncu častejšie padá k minimum.

Dĺžka √n zase pôsobí tak, že s počtom prvkom sa rýchlosť spomaľuje.

Pri **menšom počte** je podľa grafu **vhodnejšia dĺžka √n** a pri **väčšom počte** zase **dĺžka 1/1000**.

Scenár C – Vyhodnotenie

Znova môžeme vidieť, že **rýchlosť je relatívne dobrá** pri konštantnej dĺžke definovanej ako **1/1000** počtu **a pri dĺžke** definovanej ako **√n**, kde n je počet prvkov v prioritnom fronte.

Znova podľa grafu platí to čo pri predošlom scenári, že ak je **dĺžka kratšia**, **lepšiu rýchlosť** dosahuje **√n** a ak je **dĺžka dlhšia**, tak **lepšiu rýchlosť** dosahuje dĺžka **1/1000.**

Záver

Ak je **veľkosť prioritného frontu naozaj malá** (do 200 prvkov), najlepšie pôsobila variabilná dĺžka definovaná ako **n/2.**

Ak je **veľkosť stredná** (približne do 12 000 prvkov), najlepšie pôsobila variabilná dĺžka definovaná ako **√n.**

Ak je **veľkosť veľká** (približne nad 12 000 prvkov), potom vyzerala najlepšie konštantná dĺžka definovaná ako **1/1000** počtu všetkých vložení.

Avšak vždy **musíme počítať s amortizovanou zložitosťou**, pretože raz za čas je potrebná reštrukturalizácia.