WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
POLITECHNIKA WROCŁAWSKA

KIERUNEK: INFORMATYKA TECHNICZNA

Struktury danych i złożoność obliczeniowa

PROJEKT 1

Autor: Konrad Pempera

Wrocław, kwiecień 2023

Spis treści

1 Cel, założenia i zakres projektu

Celem projektu było za zaprojektowanie i zaimplenentowanie wybranych struktur danych oraz podstawowych operacji związanych z tymi strukturami. W szczególności obiektami prac projektowych były:

- tablica dynamiczna,
- lista dwukierunkowa,
- kopiec binarny (typu maksimum),
- drzewo przeszukiwań binarnych BST,
- drzewo czerwono-czarne.

W ramach projektu należało dodatkowo zbadać efektywność czasową operacji dodawania, usuwania oraz wyszukiwania elementów.

2 Wstęp teoretyczny

Złożoność obliczeniowa określa wydajność algorytmu, czyli określa ile operacji musi zostać wykonanych w zależności od ilości danych, co więcej określa ile głównych operacji musi zostać wykonane aby rozwiązać problem dla elementów wejściowych. Złożoność obliczeniowa dzieli się na: złożoność czasową i złożoność pamięciową. Złożoność czasowa określa, ile czasu algorytm potrzebuje do wykonania określonej operacji w zależności od rozmiaru danych wejściowych. Z kolei złożoność pamięciowa określa, ile pamięci komputerowej potrzebuje algorytm, aby przetworzyć dane wejściowe.

Złożoności czasowe różnych operacji na różnych strukturach zostaną omówione podczas prezentacji wyników eksperymentu komputerowego.

Na podstawie złożonosci obliczeniowe algorytmów (operacji na strukturze danych), możemy oszacować czas wykonywania poszczególnych operacji w szczególności dla notacji O() mamy:

- O(1) - najbardziej pożądana złożoność obliczeniowa, zwana stałą złożonością obliczeniową. W operacjach o takiej złożoności obliczeniowej, liczba operacji potrzebnych do wykonania zadanej czynności nie zależy od rozmiaru struktury oraz danych w strukturze. Można ją zaobserwować podczas dodawania elementu na koniec tablicy o stałym rozmiarze lub dodawaniu oraz usuwaniu na początek oraz koniec listy.
- $O(logn)$ - logarytmiczna złożoność obliczeniowa. Liczba operacji potrzebnych do wykonania zadanej czynności rośnie (w najgorszym przypadku) logarytmicznie w zależności do liczby danych wejściowych. Złożoność ta jest możliwa do zaobserwowania podczas dodawania, usuwania lub wyszukiwania elementu w drzewie czerwono-czarnym.
- $O(n)$ - liniowa złożoność obliczeniowa. Liczba operacji, które program musi wykonać jest (w najgorszym przypadku) wprost proporcjonalna do liczby elementów w strukturze. Złożoność ta jest zauważalna między innymi podczas operacji dodawania, usuwania, wyszukiwania w tablicy alokowanej dynamicznie.
- $nlogn$ - logarytmiczno - liniowa złożoność obliczeniowa. Czas wykonywania operacji w takim algorytmie jest (w najgorszym przypadku) nieznacznie dłuższy od operacji wykonywanych w czasie liniowym w szczególności dla dużej liczby danych w strukturze różnica jest w praktyce stała. Występuje ona w wielu algorytmach sortowania np. Quicksort czy Mergesort.
- n^2 - kwadratowa złożoność obliczeniowa. Liczba operacji jest proporcjonalna do kwadratu rozmiaru danych na wejściu. Złożoność tego typu można zauważyć w przypadku sortowania bąbelkowego.

Ze względu na swoją specyfikę i zastosowania, struktury przechowujące kolekcje danych można podzielić na dwie kategorie:

- struktury w których elementy identyfikowane są przez pozycję w strukturze. Zależy nam wówczas o szybkiej informacji o elemencie pierwszym, ostatnim, $x - tym$.
- struktury w których elementy identyfikowane są przez wartość (wartość klucza).Zależy nam wówczas o szybkiej informacji o elemencie najmniejszym, największym, czy element o zadanej wartości jest już w strukturze.

‘ Pierwsza grupa struktur danych powinna oferować efektywne czasowo operacje:

- addFirst(int value) - dodanie elementu o zadanej wartości na pierwszą pozycje struktury,
- addLast(int value) - dodanie elementu o zadanej wartości na ostatnią pozycje struktury,
- iμsert(int value, int pos) - dodanie elementu o zadanej wartości na zadaną pozycje,
- deleteFirst() - usunięcie elementu z pierwszej pozycji struktury,
- deleteLast() - usunięcie elementu z ostatniej pozycji struktury ,
- deleteAt(int pos) - usunięcie elementu z zadanej pozycji,
- find(int value) - wyszukwanie elementu o zadanej wartości w strukturze.

Natomiast druga grupa struktur danych powinna oferować efektywne czasowo operacje:

- add(int value) - dodanie elementu o zadanej wartości do struktury,
- deleteRoot(int value) - usunięcie korzenia z drzewa,
- findvalue(int value) - wyszukwanie elementu o zadanej wartości w strukturze.

3 Przebieg procesu projektowanie i implementacji

3.1 Przyjęte założenia

Podczas projektowania i implementacji przyjęto następujące założenia:

- podstawowym elementem struktur jest 4 bajtowa liczba całkowita,
- wszystkie struktury danych są alokowane dynamicznie, z wyjątkiem tablicy, która przechowuje strukturę kopca,
- w przypadku wyszukiwania, dodawania, odejmowania większej ilości liczb niż jedna, wartości liczb generowane są losowo,
- zadanie jest zrealizowane w postaci jednego programu,
- program jest zrealizowana w formie koμsolowej.

3.2 Język programowania i wykorzystane narzędzia

Program, który jest częścią projektu został napisany w języku C++. Dla każdej struktury zaprojektowano i zaimplementowano dedykowaną klasę. W celu zautomatyzowania przeprowadzenia testów efektywności struktur przygotowano dwie klasy bazowe (dla pierwszej i drugiej grupy struktur) z wirtualnymi metodami, które zostały przeciążone w klasach właściwych (więcej szczegółów znajduje się w rozdziale poprzednim). Implementacje oraz testy struktur przeprowadzono w środowisku Visual Studio.

4 Metodologia przeprowadzenie eksperymentu

Wyznaczenie czasu działania operacji wykonywanej na strukturze danych w rzeczywistym systemie komputerowym zależy od wielu czynników w ogólnym przypadku są to:

- aktualna częstotliwość taktowania procesora,
- aktualnie wykonywane zadania obliczeniowe przez komputer,
- liczba danych znajdujących się w strukturze,
- wartość danej na na której wykonywana jest operacja.

Pierwsze dwa czynniki niezależne są o nas i w praktyce nie mamy na nie wpływu. Co najwyżej możemy ustawić tryb pracy komputera na najwyższą wydajność aby zapobiec istotnemu zmniejszeniu częstotliwości taktowania procesora spowodowanego działanie systemu zarządzania poborem energii. W celu wyeliminowania tych czynników, każdy test wykonywany był wykonywany 5-cio krotnie na identycznych danych. Ostatecznie za czas wykonywania testowanej czynności przyjęto najmniejszy z otrzymanych czasów.

Ze względu na bardzo krótki czas działania większości operacji na strukturach, każda testowana operacja wykonywana była wielokrotnie przy czym liczba tych czynności była znacząco mniejsza od liczby elementów znajdujących się w strukturze tj. ostatecznie przyjęto 1%.

Podczas wyznaczania czasu operacji dla określonej struktury i zadanej liczby n elementów w niej się znajdujących generowano $a = 1\% \cdot n$ elementów oraz

- zmierzono łączny czas dodawania a elementów do struktury - w przypadku dodawania,
- dodano a elementów do struktury i zmierzono łączny czas usunięcia a elementów znajdujących się w strukturze (a - losowo wybranych elementów w przypadku listy i tablicy oraz a razy korzeń)- w przypadku usuwania,
- zmierzono łączny czas wyszukania a elementów znajdujących się w strukturze - w przypadku wyszukiwania.

Ostatecznie czas wykonania operacji na strukturze otrzymano dzieląc łączny czas wykonania operacji przez liczbę a.

W tabeli 1 przedstawiono dla każdej struktury zakres liczby elementów oraz krok (zmian tej wartości) podczas testów operacji na strukturach. Testy struktur przeprowadzono na identycznych danych.

Tabela 1: Zakres liczby elementów w strukturze

Struktura	Liczba początkowa	Krok	Liczba końcowa
Tablica dynamiczna	500	2500	4750
Lista dwukierunkowa	5000	5000	95000
Kopiec	5000	5000	95000
Drzewo BST	5000	5000	95000
Drzewo RB	5000	5000	95000

W celu uzależnienia się od wpływu wartości danych wejściowych (mają one istotny wpływ na operacje wykonywane na drzewach) badania efektywności czasowej przeprowadzono dziesięciokrotnie na różnych danych wejściowych. Na podstawie dziesięciu otrzymanych czasów wyliczono wartość średnią, minimalną i maksymalną. Dodatkowo, dla każdej operacji na strukturze danych wyznaczono metodą najmniejszych kwadratów współczynniki krzywej odpowiadającej oczekiwanej złożoności czasowej. Wszystkie wymienione wartości oraz wykres wspomnianej funkcji zostały przedstawione na wykresach zamieszczonych w kolejnych rozdziałach.

Porównanie czasu działania różnych operacji na strukturze lub tych samych operacji na różnych strukturach zostało przeprowadzone na podstawie postaci i/lub współczynników aproksymowanych funkcji opisujących zależność czasu obliczeń od liczby elementów znajdujących się w strukturze.

5 Tablica dynamiczna

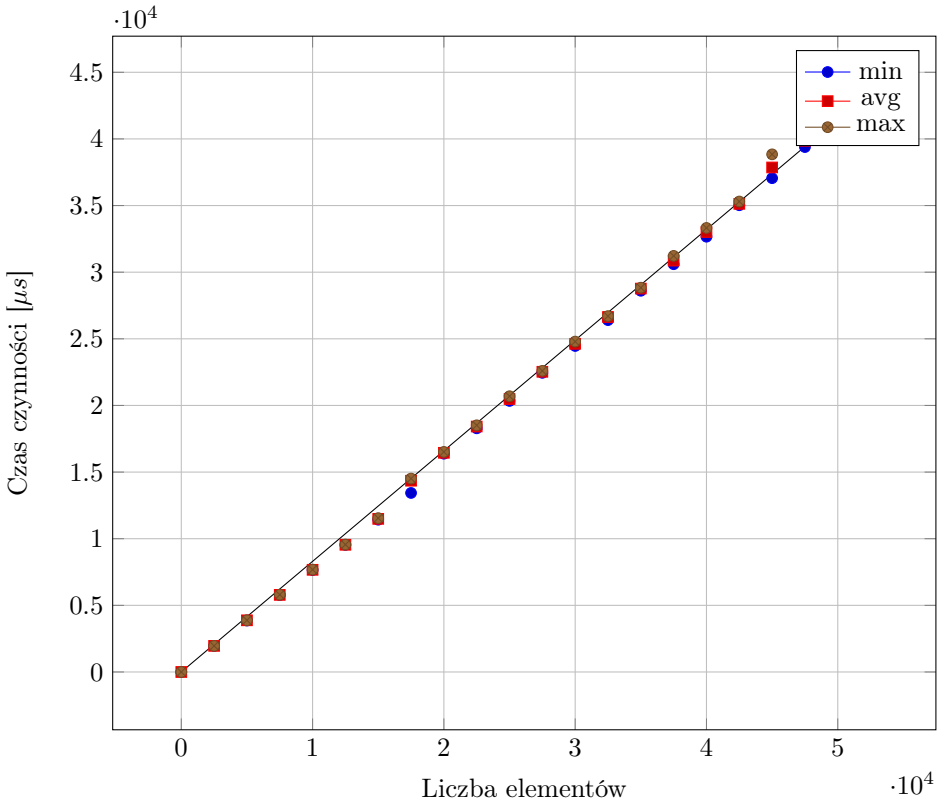
Tablica dynamiczna jest strukturą, której rozmiar zmienia się dynamicznie przy każdej operacji dodania lub usunięcia elementu. Zatem każde dodanie lub usunięcie elementu wymaga alokacji nowego obszaru pamięci oraz skopiowania zawartości tablicy. Tablicę dynamiczną można zaimplementować w ten sposób, że alokowana będzie większa ilość pamięci niż faktycznie jest potrzebna (z nadmiarem), wówczas operacje dodawania i usuwania przebiegają szybciej. Celem zadania projektowego było zbadanie czasu obliczeń w zależności od liczby elementów w strukturze dla pierwszego typu opisanej tablicy dynamicznej. Taki rodzaj zarządzania pamięcią jest szczególnie przydatny, gdy rozmiar tablicy jest duży lub zmienny. Na strukturze przebadano następujące operacje, których teoretyczna złożoność czasowa wynosi:

- dodawanie na początek - $O(n)$,
- dodawanie na koniec - $O(n)$,
- dodawanie na losową pozycje - $O(n)$,
- usuwanie z początku - $O(n)$,
- usuwanie z końca - $O(n)$,
- usuwanie z losowej pozycji - $O(n)$,
- wyszukiwanie elementu - $O(n)$.

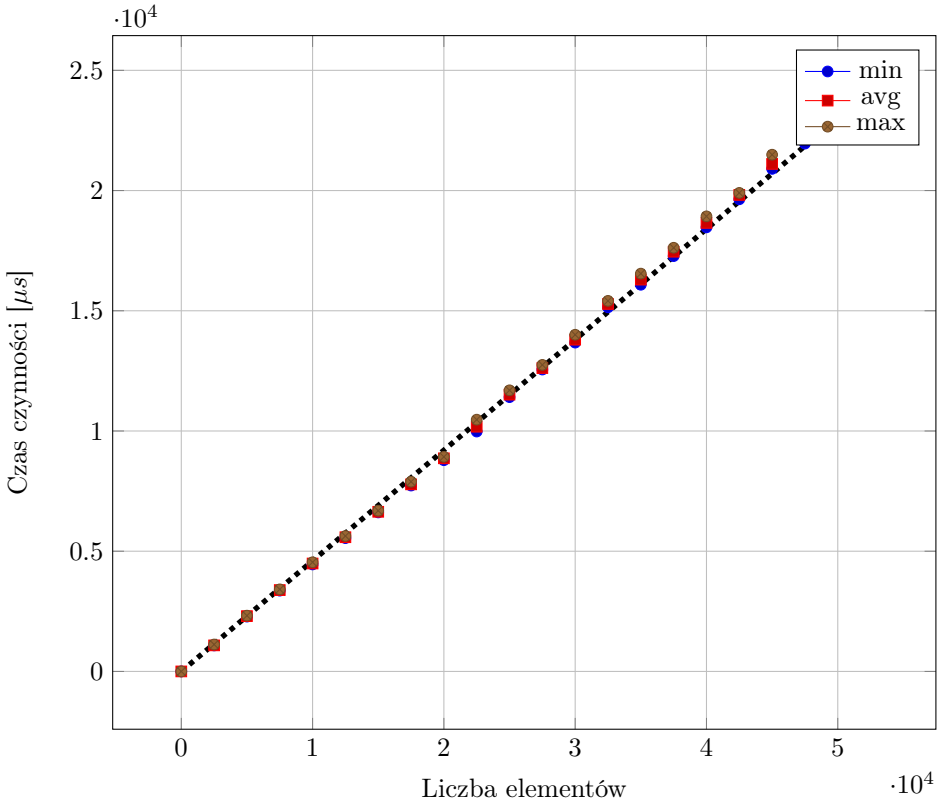
Badania przeprowadzono dla początkowej liczby elementów 0 ze skokiem 2500 do 47 5000. W Tabeli 2 przedstawiono średnie czasy działania funkcji AddFirst, AddLast, Random Iμstert, DeleteFirst, DeleteLast, Random Remove oraz Find. Natomiast na Rysunkach 1, 2, 3, 4, 5, 6, 7 przedstawiono wartość minimalną, średnią oraz maksymalną czasu operacji w zależności od liczby elementów oraz zależność czasu obliczeń od rozmiaru problemu wyestymowaną przy pomocy metody najmniejszych kwadratów.

Tabela 2: Pomiary czasow wykonywania operacji na tablicy

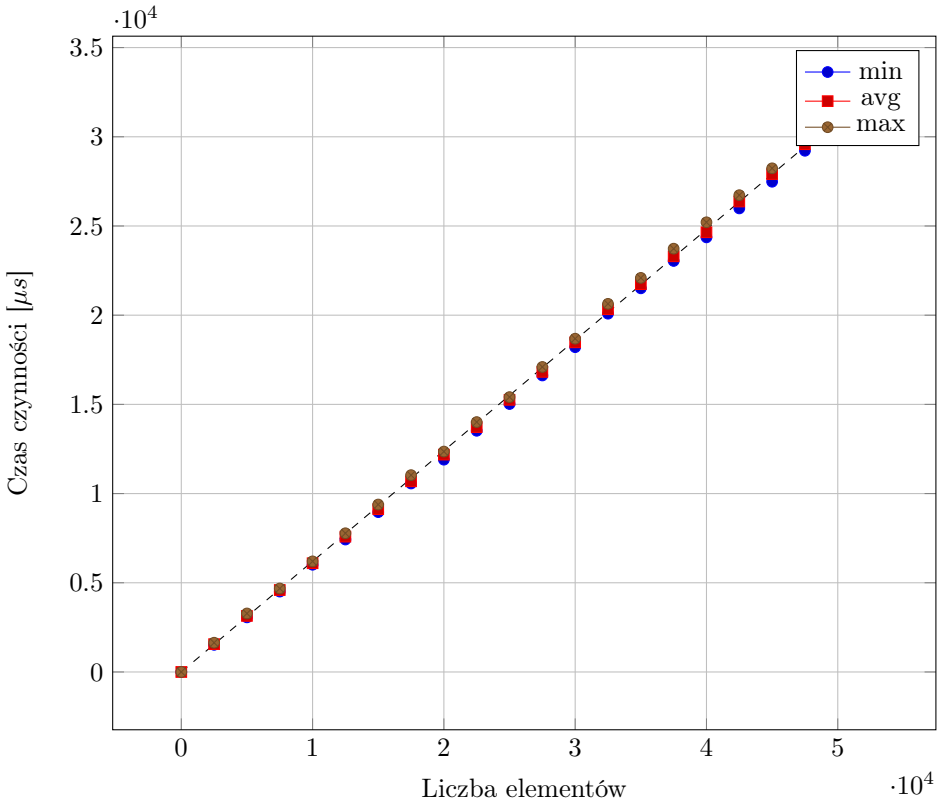
n	AddFirst [μs]	AddLast [μs]	Random Iμsert [μs]	DelFirst [μs]	DelLast [μs]	Random Remove [μs]	Find [μs]
0	0.0	0.0	0.0	null	null	null	null
2500	1960.0	1084.0	1560.0	2120.0	1080.0	1600.0	472.0
5000	3878.0	2300.0	3142.0	4418.0	2280.0	3340.0	956.0
7500	5786.3	3383.5	4594.3	6539.6	3353.2	4919.7	1546.3
10000	7667.0	4489.0	6096.0	8761.0	4466.0	6593.0	1956.0
12500	9545.6	5581.6	7587.2	10852.8	5523.2	8135.2	2387.2
15000	11490.2	6639.6	9124.4	12987.6	6590.3	9651.0	2965.0
17500	14352.7	7780.6	10687.7	15213.9	7542.4	11354.3	3439.6
20000	16446.5	8864.5	12184.5	17355.0	8459.0	12745.0	3860.5
22500	18418.2	10178.0	13726.2	19485.9	9572.5	14472.4	4449.7
25000	20485.6	11512.8	15254.8	21701.2	10640.0	16079.6	4772.0
27500	22525.9	12618.9	16821.8	23861.5	11711.5	17714.8	5305.3
30000	24625.1	13798.6	18478.7	26061.9	12771.6	19347.6	5870.3
32500	26638.3	15283.2	20334.2	28275.7	13953.7	21202.2	6169.2
35000	28769.1	16295.9	21739.3	30706.4	15575.0	22810.5	6878.7
37500	30870.7	17456.4	23301.4	32906.7	16739.8	24335.0	7487.7
40000	33017.3	18654.4	24648.4	35144.8	17852.4	25706.8	7941.8
42500	35128.7	19818.1	26371.4	37246.4	18911.0	27446.8	8531.0
45000	37858.5	21108.1	27910.9	39351.1	20007.8	28997.0	9271.5
47500	39861.3	22251.4	29583.1	41584.6	21104.3	30741.6	9524.3



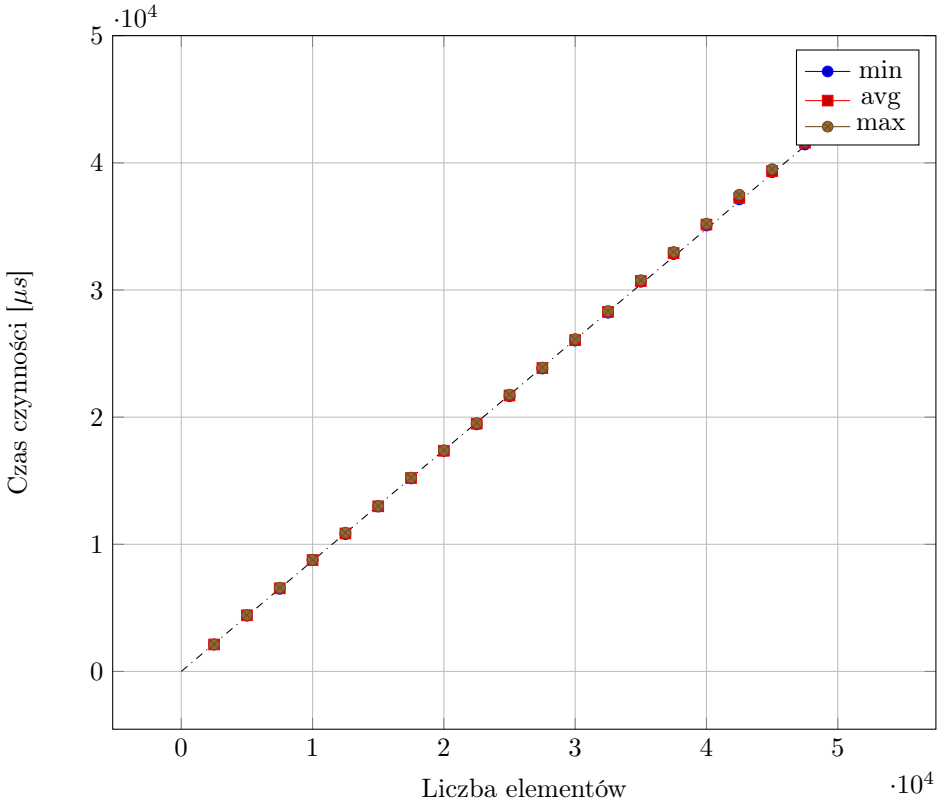
Rysunek 1: Wykres zależności czasu dodawania elementów na początek tablicy od liczby elementów w strukturze



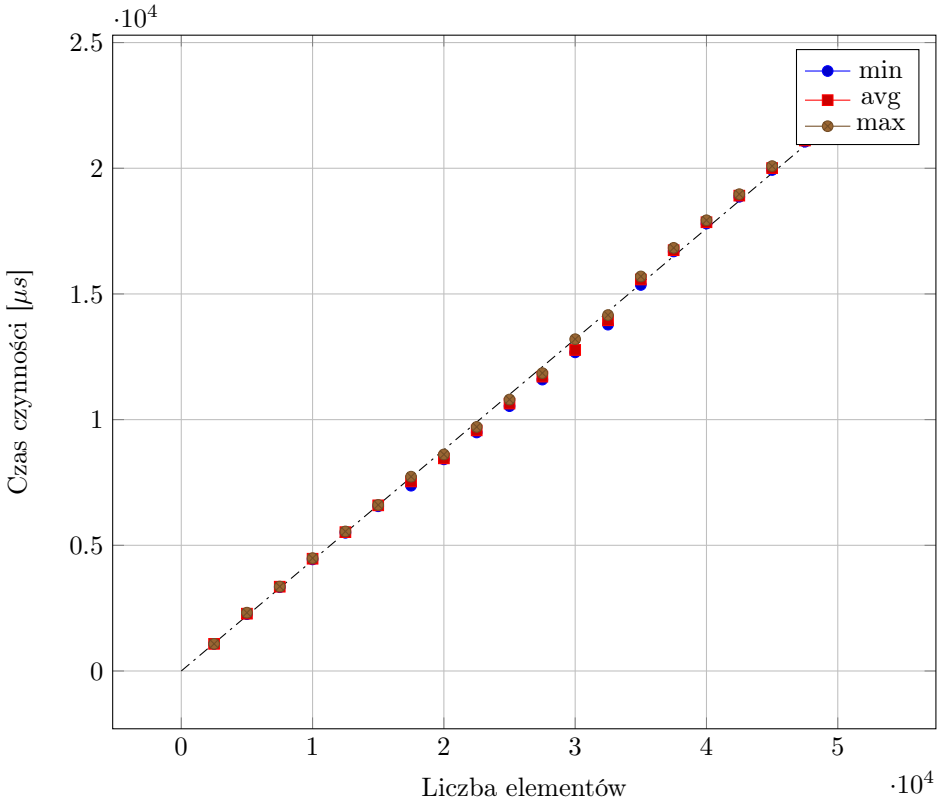
Rysunek 2: Wykres zależności czasu dodawania elementów na koniec tablicy od liczby elementów w strukturze



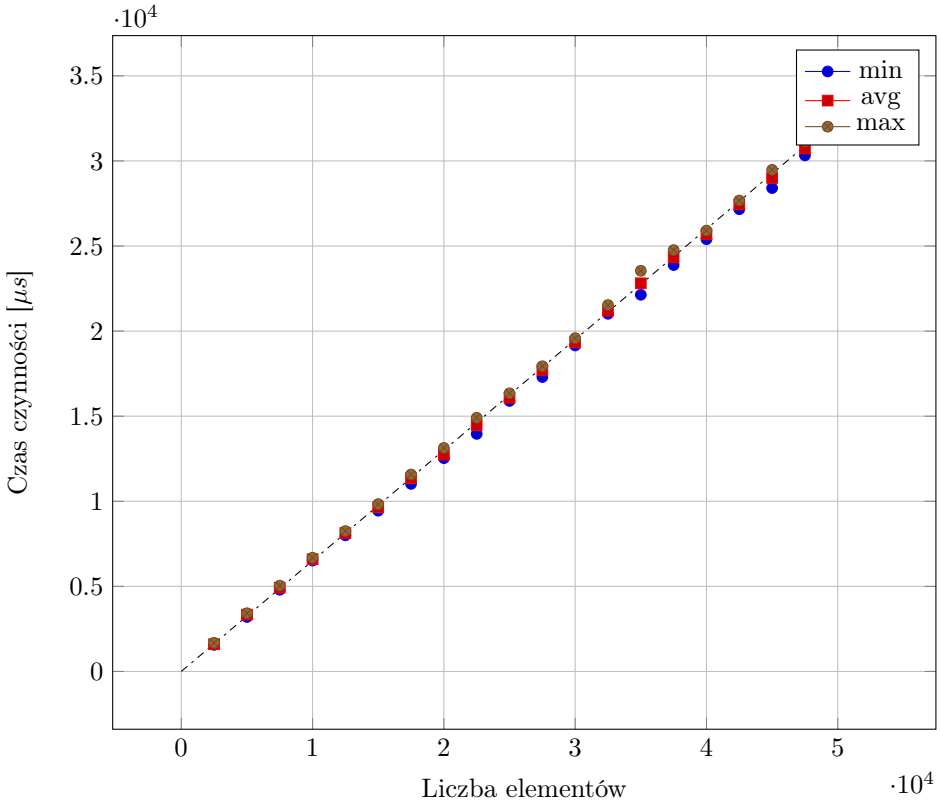
Rysunek 3: Wykres zależności czasu dodawania elementów w losowe miejsca w tablicy od liczby elementów w strukturze



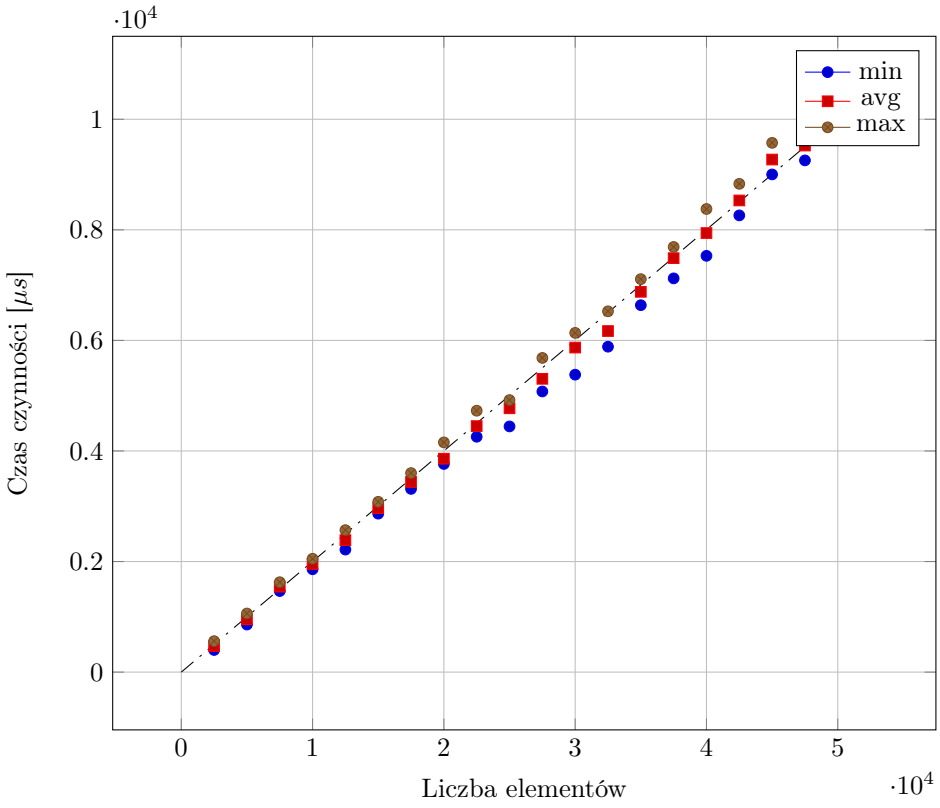
Rysunek 4: Wykres zależności czasu usuwania elementów z początku tablicy od liczby elementów w strukturze



Rysunek 5: Wykres zależności czasu usuwania elementów z końca tablicy od liczby elementów w strukturze



Rysunek 6: Wykres zależności czasu usuwania elementów z losowych pozycji tablicy od liczby elementów w strukturze



Rysunek 7: Wykres zależności czasu wyszukiwania elementów w tablicy od liczby elementów w strukturze

5.1 Dodawanie elementu do tablicy

Jak można zauważyć dodawanie na początek tablicy trwa nieco dłużej niż dodawanie na koniec tablicy. Zjawisko to jest spowodowane koniecznością przesuwania wszystkich elementów o pozycje dalej w przypadku dodawania na początek. Podczas dodawania na koniec nie ma potrzeby "rozsuwania"elementów tablicy, ponieważ za wstawianym elementem nic już się nie znajduje. Średnie czasy dodawania w losowe miejsca mieszczą się w przedziale pomiędzy średnimi czasami dodawania na początek, a średnimi czasami dodawania na koniec.

5.2 Usuwanie elementu z tablicy

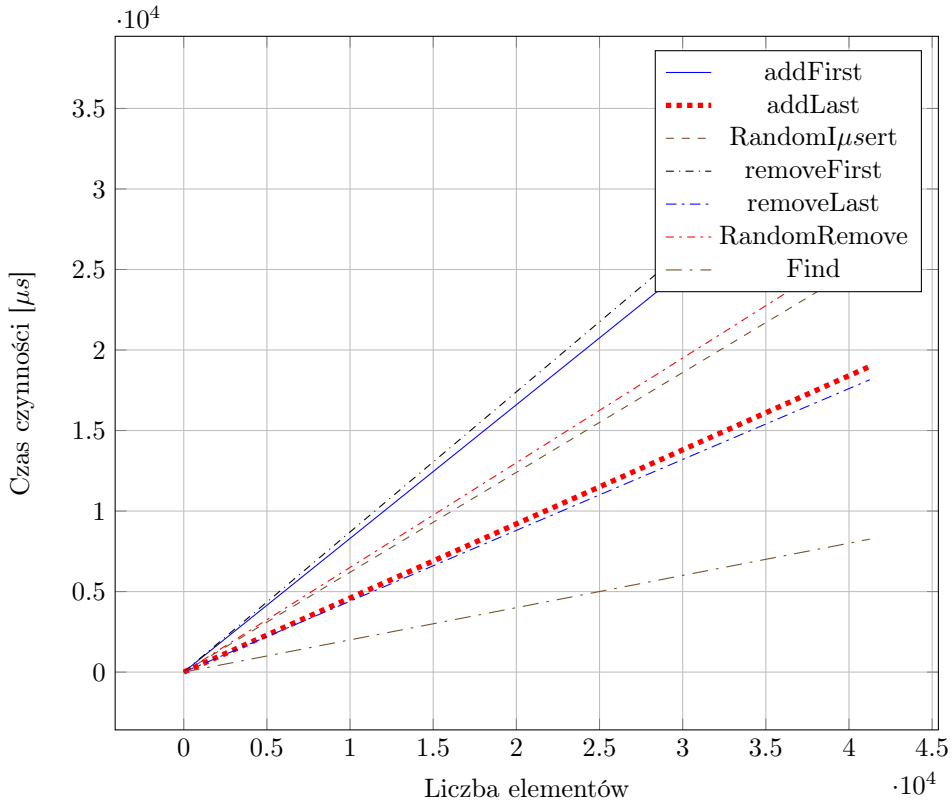
Jak można zauważyć również i w tym przypadku usuwanie z końca tablicy trwa krócej niż usuwanie z początku. Sytuacja ta jest spowodowana tym samym czynnikiem co występuje podczas dodawania elementu do tablicy tj. konieczność zmiany pozycji elementów znajdujących się w na dalszych pozycjach tablicy niż ta, w której znajduje się usuwany element.

5.3 Wyszukiwanie elementu

Ostatnią testowaną operacją na tablicy alokowanej dynamicznie jest wyszukiwanie w niej zadanego elementu. Czasy operacji na tablicy w zależności od ilości elementów w niej zostały przedstawione na wykresie na Rysunku 7.

5.4 Porównanie

Na Rysunku 8 zestawiono czasy wszystkich rozpatrywanych operacji na tablicy. Wykresy usuwania elementów najbardziej pokrywają się z teorią. Usuwanie z początku trwa najdłużej, a z końca najkrócej, średnie czasy usuwania z losowych pozycji mieszczą się pomiędzy wyżej wymienionymi operacjami.



Rysunek 8: Zestawienie zależności czasu wykonywania operacji na tablicy

Wyszukiwanie jest najszybsze i trwa cztery razy krócej niż dodawanie na początek tabeli, które jest najbardziej czasochłonne.

5.5 Wnioski

Analizując tablice i wykresy zależności czasu od wielkości struktury można zaobserwować, że złożoność obliczeniowa operacji na tablicy jest zgodna z teoretycznymi przewidywaniami tzn. jest ona liniowa. Porównując ze sobą średnie czasy dodawania, odejmowania oraz wyszukiwania w strukturze, można zauważyć, że czasy różnych operacji na strukturze są do siebie bardzo zbliżone. Operacje dodawania oraz usuwania z losowej pozycji są do siebie podobne. Na początku należy zaalokować odpowiednią ilość pamięci, następnie przepisać tablice do nowo-zaalokowanej tablicy a następnie rozsunąć lub zsunąć elementy w tablicy wkoło rozpatrywanej pozycji. Czas wyszukiwania w tym przypadku również jest zbliżony do czasu pozostałych operacji, jednak prawdopodobnie jest to spowodowane stosunkowo niewielką ilością elementów w tablicy co wiąże się z koniecznością przeszukania i porównania wszystkich elementów w strukturze. Uniemożliwia to w jednoznaczne stwierdzenie czy czas wyszukiwania elementu jest porównywalny z czasem dodawania oraz usuwania ze struktury.

6 Lista dwukierunkowa

Lista dwukierunkowa jest strukturą, która przechowuje elementy w kolejności dodawania i każdy jej element zawiera wskaźnik na następny oraz poprzedni element. Pierwszy element nazywany jest ogonem i jego poprzednik nie istnieje, natomiast element ostatni nazywany jest ogonem i nie posiada on wskaźnika na następny element.

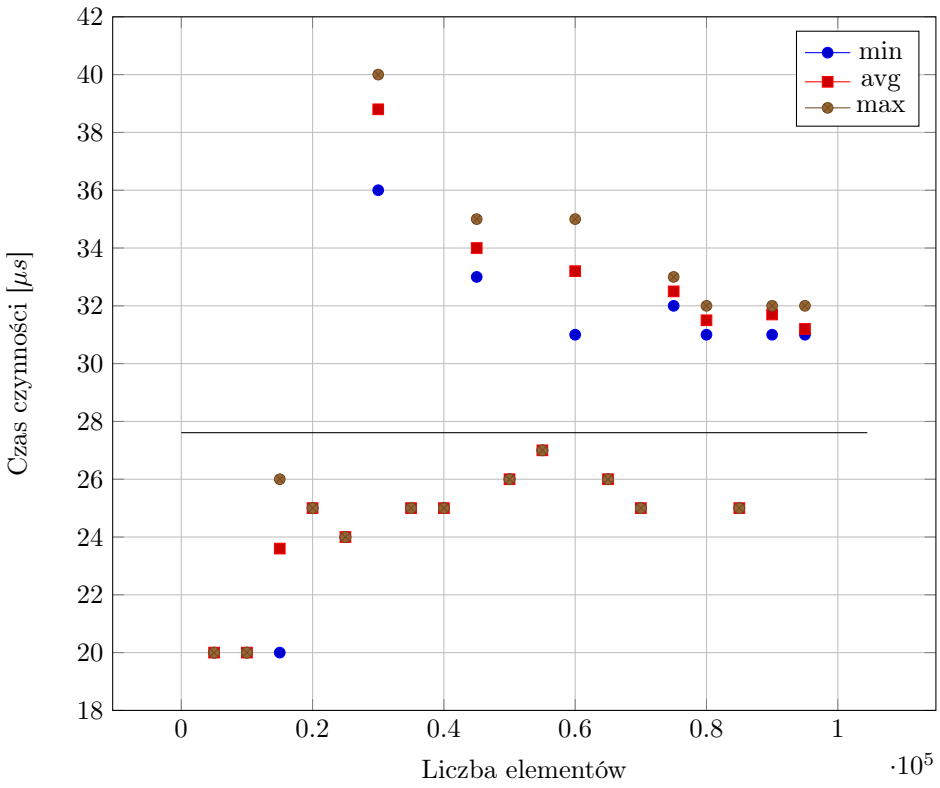
Celem zadania projektowego było zbadanie czasu obliczeń w zależności od liczby elementów w strukturze dla ww. struktury. Na strukturze przebadano następujące operacje, których teoretyczna złożoność czasowa wynosi:

- dodawanie na początek - $O(1)$,
- dodawanie na koniec - $O(1)$,
- dodawanie na losową pozycje - $O(n)$ (optymistyczna $O(1)$, w najgorszym przypadku $O(n)$, średnio $O(n)$),
- usuwanie z początku - $O(1)$,
- usuwanie z końca - $O(1)$,
- usuwanie z losowej pozycji - $O(n)$ (optymistyczna $O(1)$, w najgorszym przypadku $O(n)$, średnio $O(n)$),
- wyszukiwanie elementu - $O(n)$ (optymistyczna $O(1)$, w najgorszym przypadku $O(n)$, średnio $O(n)$).

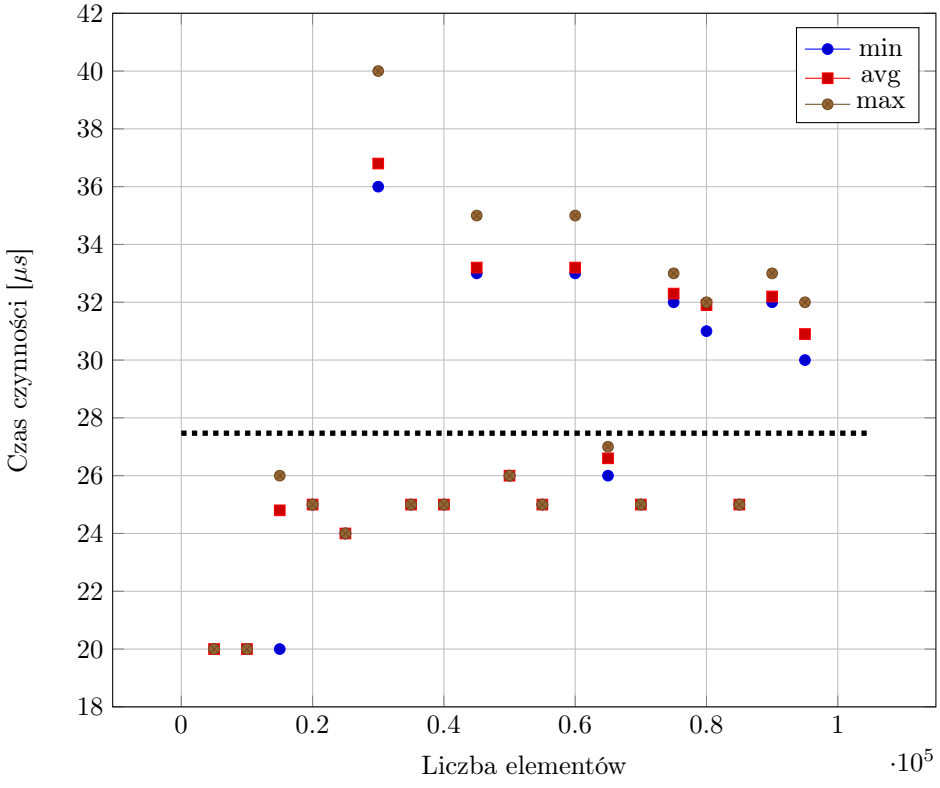
Badania przeprowadzono dla początkowej liczby elementów wynoszącej od 5000 do 95000 ze skokiem 5000. W Tabeli 3 przedstawiono średnie czasy działania funkcji AddFirst, AddLast, Random Iμstert, DeleteFirst, DeleteLast, Random Remove oraz Find. Natomiast na Rysunkach 9, 10, 11, 12, 13, 14, 15 przedstawiono wartość minimalną, średnią oraz maksymalną czasu operacji w zależności od liczby elementów oraz zależność czasu obliczeń od rozmiaru problemu wyestymowaną przy pomocy metody najmniejszych kwadratów.

Tabela 3: Pomiary czasow wykonywania operacji na liscie

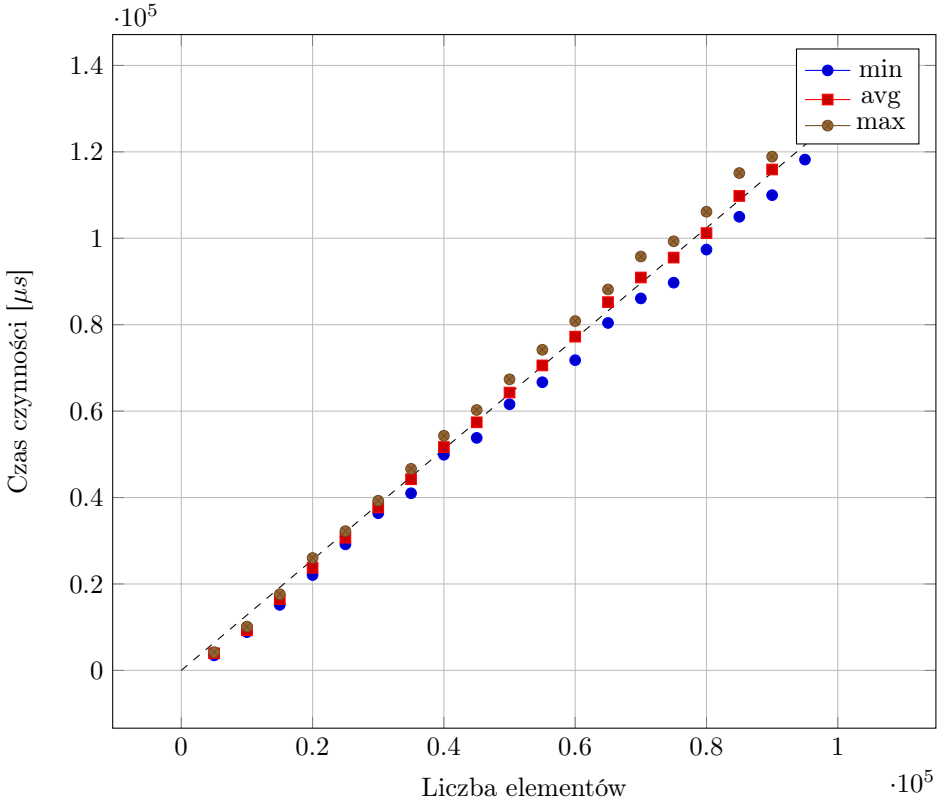
n	AddFirst [μs]	AddLast [μs]	Random Iμsert [μs]	DelFirst [μs]	DelLast [μs]	Random Remove [μs]	Find [μs]
5000	20.0	20.0	3954.0	0.0	18.0	3866.0	7470.0
10000	20.0	20.0	9290.0	10.0	11.0	9153.0	17374.0
15000	23.6	24.8	16503.6	13.0	13.0	16724.4	27881.1
20000	25.0	25.0	23741.0	15.0	15.0	23612.5	35676.5
25000	24.0	24.0	30709.6	16.0	16.0	30294.8	46339.2
30000	38.8	36.8	37732.7	16.0	17.2	37250.0	56825.1
35000	25.0	25.0	44239.4	17.0	17.0	42769.8	67128.4
40000	25.0	25.0	51682.4	17.0	17.0	49977.8	73543.3
45000	34.0	33.2	57411.6	15.0	17.3	55836.7	85230.4
50000	26.0	26.0	64294.4	16.0	18.0	61107.4	93570.2
55000	27.0	25.0	70582.0	16.0	18.2	68542.9	100825.6
60000	33.2	33.2	77238.7	16.0	18.0	75854.6	112595.9
65000	26.0	26.6	85217.0	16.0	18.0	80452.3	120839.0
70000	25.0	25.0	90927.8	17.0	18.0	88313.0	130423.2
75000	32.5	32.3	95534.9	16.8	18.2	95124.3	144187.0
80000	31.5	31.9	101187.9	16.0	18.0	100576.9	148036.9
85000	25.0	25.0	109794.7	16.0	18.0	108131.7	157748.6
90000	31.7	32.2	115923.2	16.0	18.4	115146.7	169231.3
95000	31.2	30.9	123785.5	16.0	18.0	120992.6	170466.6



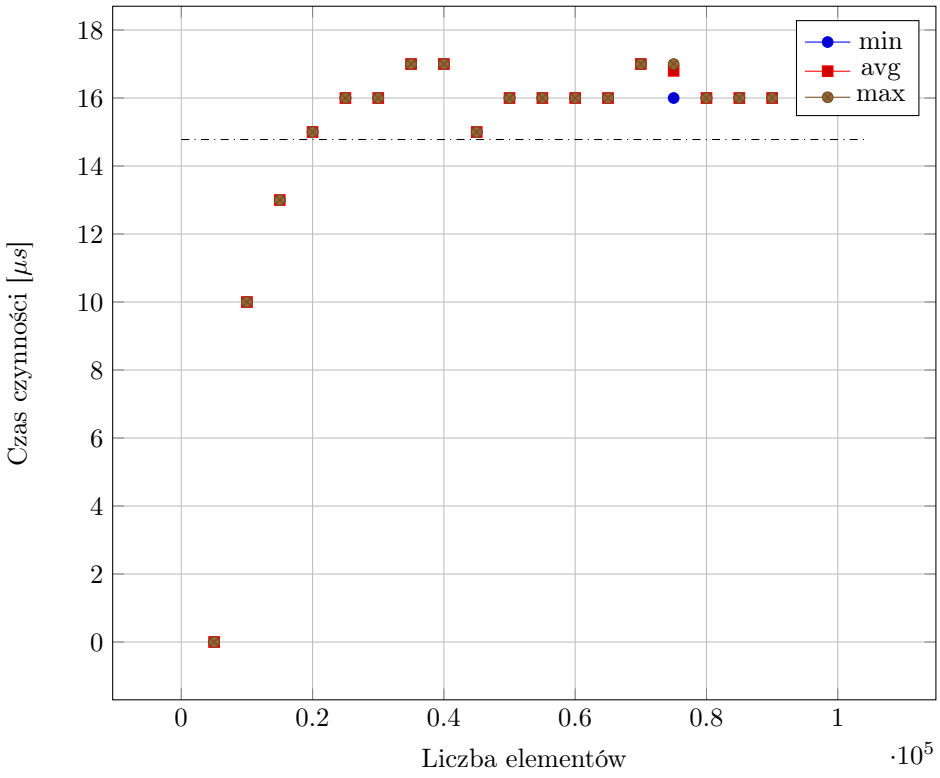
Rysunek 9: Wykres zależności czasu dodawania elementów na początek listy od liczby elementów w strukturze



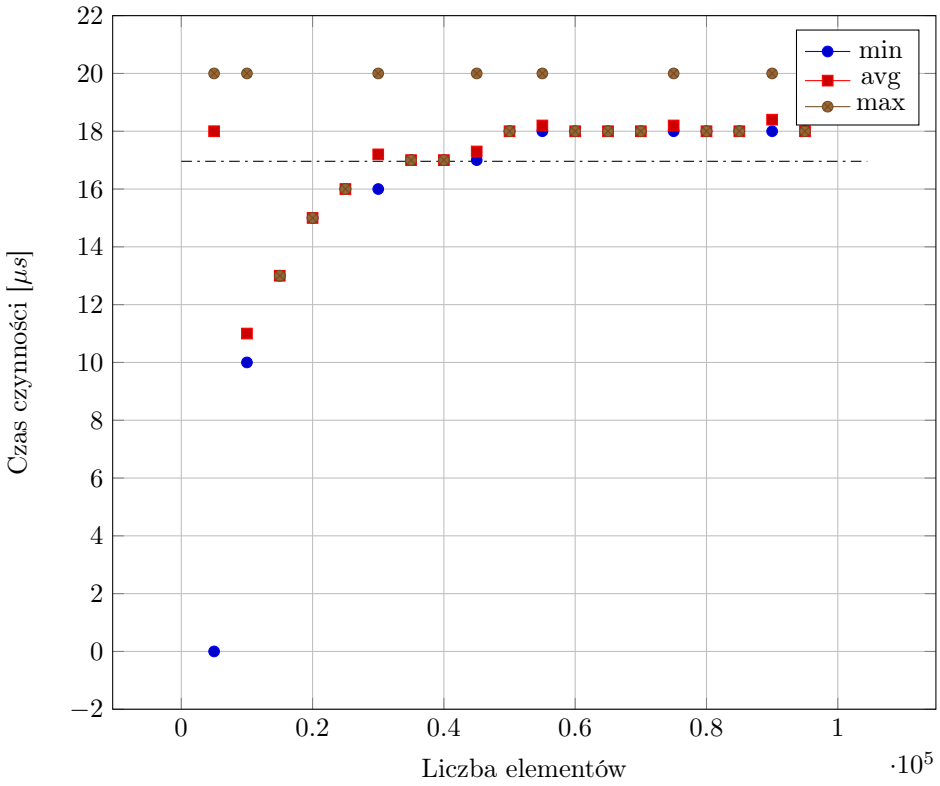
Rysunek 10: Wykres zależności czasu dodawania elementów na koniec listy od liczby elementów w strukturze



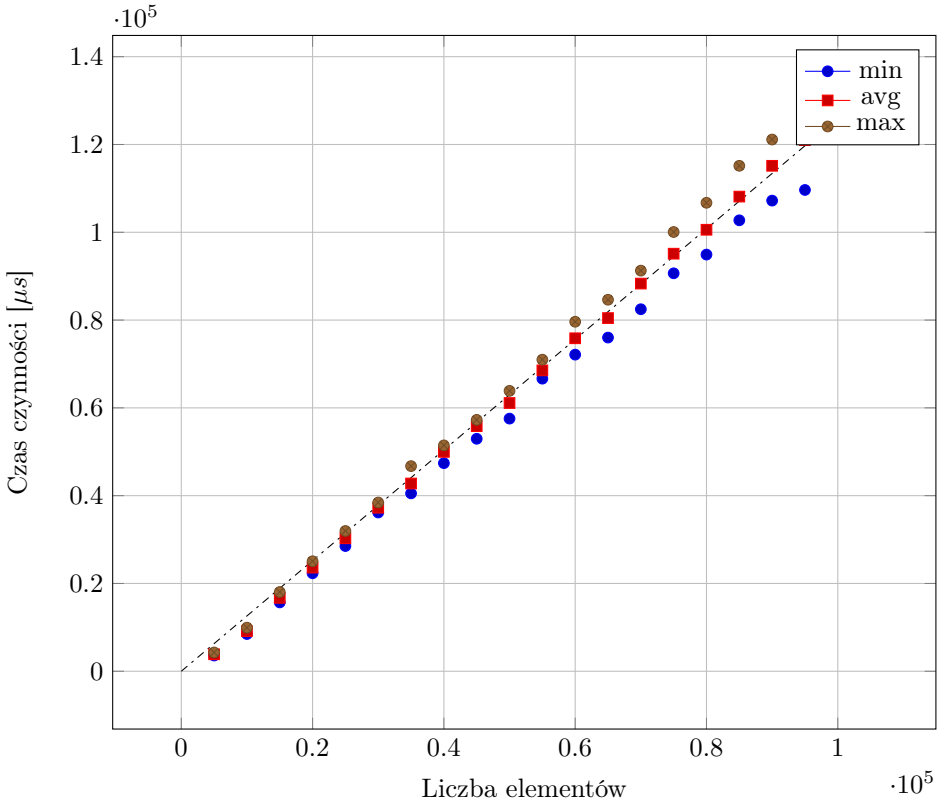
Rysunek 11: Wykres zależności czasu dodawania elementów w losowe miejsca w liście od liczby elementów w strukturze



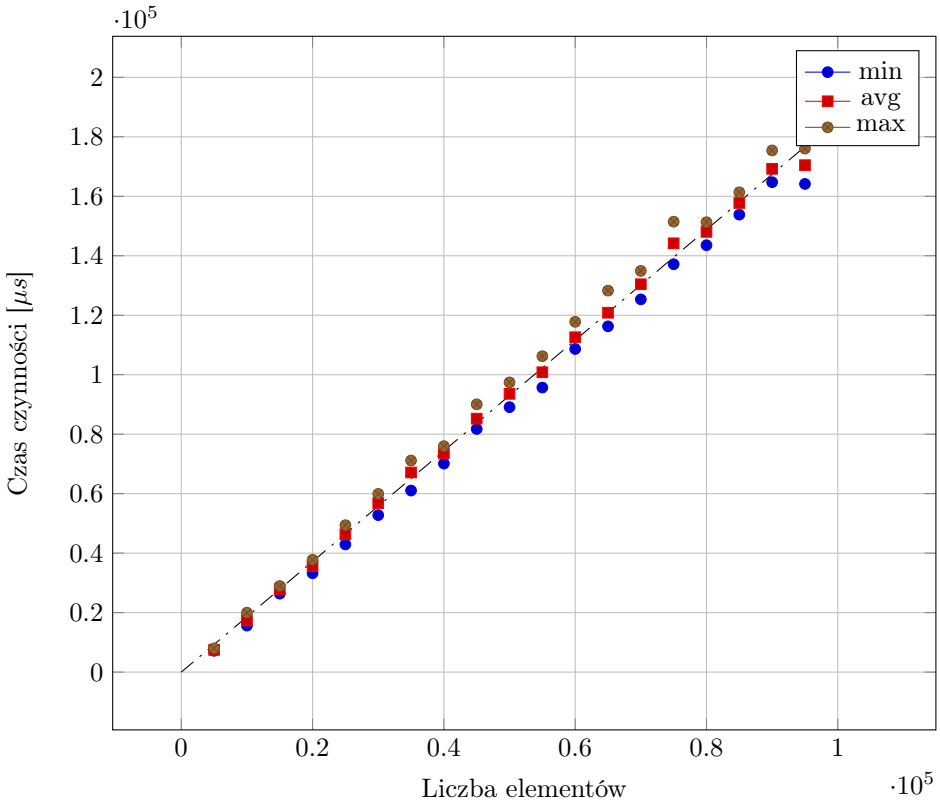
Rysunek 12: Wykres zależności czasu usuwania elementów z początku listy od liczby elementów w strukturze



Rysunek 13: Wykres zależności czasu usuwania elementów z końca listy od liczby elementów w strukturze



Rysunek 14: Wykres zależności czasu usuwania elementów z losowych pozycji listy od liczby elementów w strukturze



Rysunek 15: Wykres zależności czasu wyszukiwania elementów w liscie od liczby elementów w strukturze

6.1 Dodawanie elementu do listy

Analizując dane tabelaryczne można zauważyć, że czas dodawania na początek oraz na koniec listy jest bardzo mały, różnice pomiędzy kolejnymi pomiarami są praktycznie pomijalne, a wartości oscylują koło stałej wartości. Ze względu na bardzo krótki czas wykonywania operacji dodawania na początek lub koniec listy dwukierunkowej, bardzo ciężko zmierzyć dokładnie czas. Dlatego w tym przypadku szczególnie istotne jest dopasowanie odpowiedniej prostej, która ukaże jaki przebieg ma prosta utworzona z punktów, które są zależnością czasu wykonywania operacji od liczby elementów w strukturze.

Sytuacja zmienia się diametralnie w przypadku dodawania w losowe miejsca w liście. W tej sytuacji widać silną zależność wzrostu czas wraz ze wzrostem liczby elementów przed wykonaniem operacji. Zjawisko to jest spowodowane koniecznością wyszukania pozycji, na którą jest wstawiany nowy element. Lista w przeciwieństwie do tablicy nie ma indeksów pozycji i na początku należy znaleźć pozycje w liście, a dopiero następnie wstawić tam element.

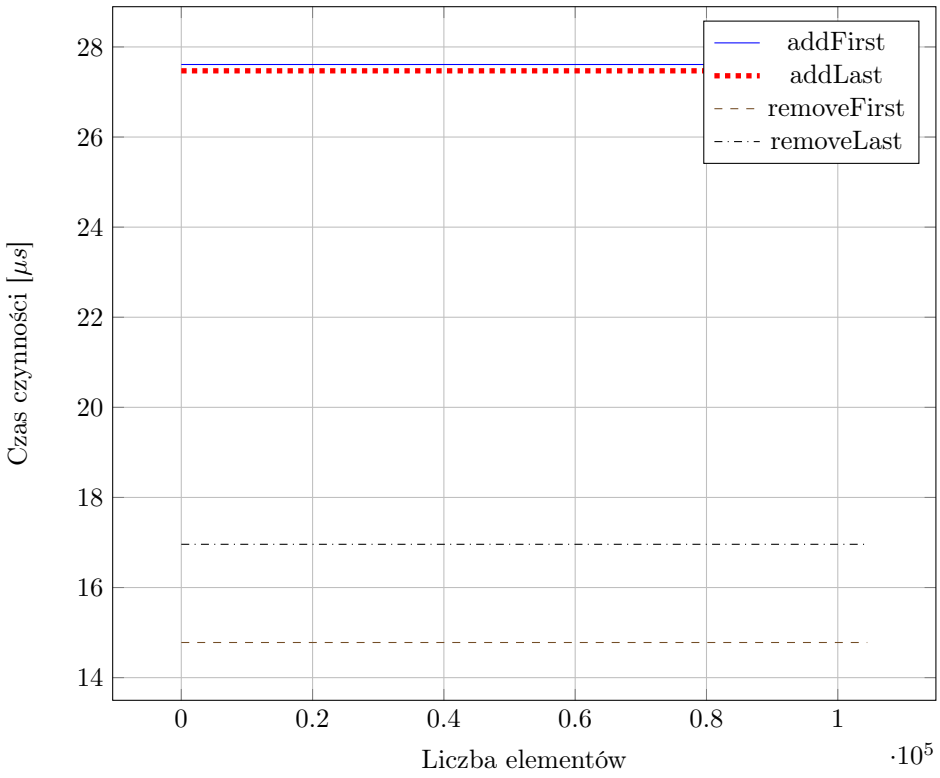
6.2 Usuwanie elementu z tablicy

Operacja usuwania z listy jest bardzo podobna do operacji dodawania elementów, tak więc można wysunąć identyczne wnioski.

6.3 Wyszukiwanie elementu

Ostatnią testowaną operacją na liście dwukierunkowej jest wyszukiwanie w niej zadanego elementu. Jak można zauważyć wykres przedstawiający wyszukiwanie elementu w strukturze jest liniowy, co jest spowodowane koniecznością szukania element po elemencie.

6.4 Porównanie



Rysunek 16: Zestawienie zależności czasu wykonywania operacji na liście dwukierunkowej

Na Rysunku 16 zestawiono ze sobą średnie czasy dodawania i usuwania z pierwszej i ostatniej pozycji listy dwukierunkowej. Są to najszybsze operacje i są realizowane w czasie O(1). Pozostałe operacje pominięto ze względu na silną liniową zależność od liczby elementów.

6.5 Wnioski

Spoglądając na wykresy porównawcze operacji dodawania na początek i koniec listy oraz usuwania z początku i końca listy, można wywnioskować, że usuwanie trwa krócej niż dodawanie. Prawdopodobnie jest to spowodowane różnicami czasu przeprowadzania procesu alokacji pamięci, który występuje podczas dodawania, oraz procesu dealokacji pamięci, który występuje podczas usuwania elementu. Jest to spowodowane faktem, iż alokacja trwa dłużej niż dealokacja. Doświadczenie, utwierdziło w przekonaniu, że korzystanie z listy jest dobrym pomysłem w przypadku gdy dane ze struktury pobierane są po kolei, od początku do końca lub od końca do początku, a nie istotne jest pobieranie konkretnych wartości czy konkretnych pozycji listy. Co więcej lista nie jest dobrą strukturą w przypadku, gdy istotne jest wyszukiwanie elementów.

7 Kopiec

Kopiec jest strukturą jest drzewem binarnym zupełnym. Najczęściej implementowany jest w postaci tablicy statycznej, w której pozycje poszczególnych elementów są ściśle określone - wartości spełniają warunki kopca. Każdy element w kopcu posiada "rodzica". Tak nazwany element musi mieć wartość większą lub równą od potomka. Celem zadania projektowego było zbadanie czasu obliczeń w zależności od liczby elementów w strukturze dla ww. struktury. Na strukturze przebadano następujące operacje, których teoretyczna złożoność czasowa wynosi:

- dodawanie elementu - $O(log(n))$,
- usuwanie elementu - $O(log(n))$,
- wyszukiwanie elementu - $O(n)$ (optymistyczna $O(1)$, pesymistyczna $O(n)$).

Badania przeprowadzono dla początkowej liczby elementów wynoszącej od 0 do 1 000 000 ze skokiem 50 000. W Tabeli 4 przedstawiono średnie czasy działania funkcji add, remove, find. Natomiast na Rysunkach 17, 18, 19 przedstawiono wartość minimalną, średnią oraz maksymalną czasu operacji w zależności od liczby elementów oraz zależność czasu obliczeń od rozmiaru problemu wyestymowaną przy pomocy metody najmniejszych kwadratów.

Tabela 4: Pomiary czasow wykonywania operacji na kopcu

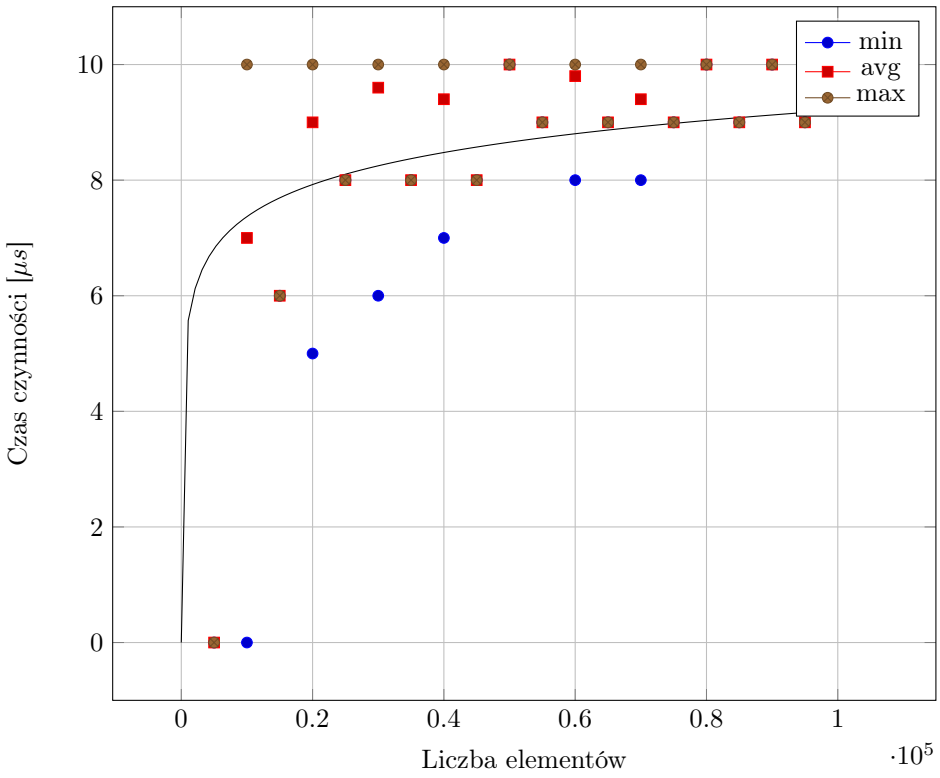
n	Add $[\mu s]$	Delete $[\mu s]$	Find $[\mu s]$
50000	10.0	46.2	8868.2
100000	9.8	53.0	18020.9
150000	9.5	54.4	26975.9
200000	9.0	57.9	36041.5
250000	9.0	59.1	44846.6
300000	9.3	60.1	54570.1
350000	9.2	62.6	63643.3
400000	9.0	70.0	71918.5
450000	9.0	63.1	81933.9
500000	9.1	65.6	89807.3
550000	9.0	65.4	99217.2
600000	9.0	65.3	110440.5
650000	9.2	67.1	118850.4
700000	9.0	69.2	128417.9
750000	9.0	66.5	139313.2
800000	9.0	69.2	146038.7
850000	9.0	68.5	156596.3
900000	9.1	67.8	167410.2
950000	9.0	69.7	174224.8

7.1 Dodawanie elementu do kopca

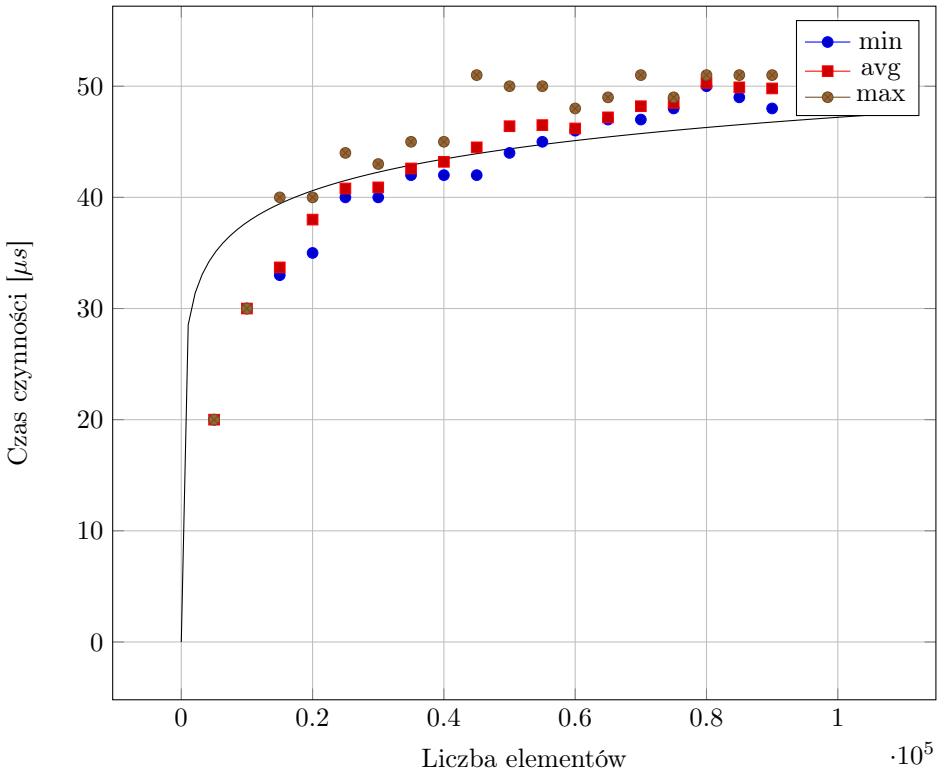
Czas działania algorytmu dodawania do kopca jest bardzo krótki co powoduje duże błędy w pomiarze czasu wykonywania operacji. Dla dużych wartości n wykres pokrywa się z krzywą logarytmiczną wyznaczoną za pomocą metody najmniejszych kwadratów. Wykres ten znajduje się na Rysunku 17.

7.2 Usuwanie elementu z kopca

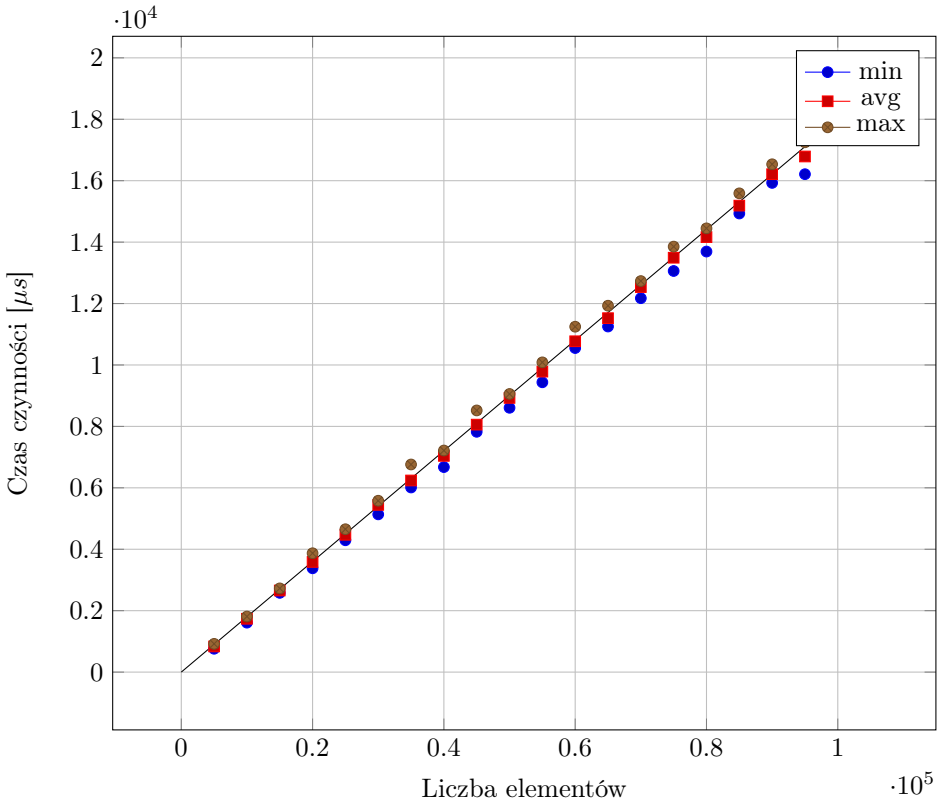
Wykres usuwania elementów z kopca, został przedstawiony na Rysunku 18 i również jak w przypadku dodawania najlepiej dopasowana krzywa jest logarytmiczna.



Rysunek 17: Wykres zależności czasu dodawania elementów do kopca od liczby elementów w strukturze



Rysunek 18: Wykres zależności czasu usuwania elementów z kopca od liczby elementów w strukturze



Rysunek 19: Wykres zależności czasu wyszukiwania elementów w kopcu od liczby elementów w strukturze

7.3 Wyszukiwanie elementu

Ostatnią testowaną operacją na kopcu jest wyszukiwanie w nim zadanego elementu. Czasy wyszukiwania została przedstawiona na wykresie na Rysunku 19. W odróżnieniu od poprzednich operacji jest ona liniowa.

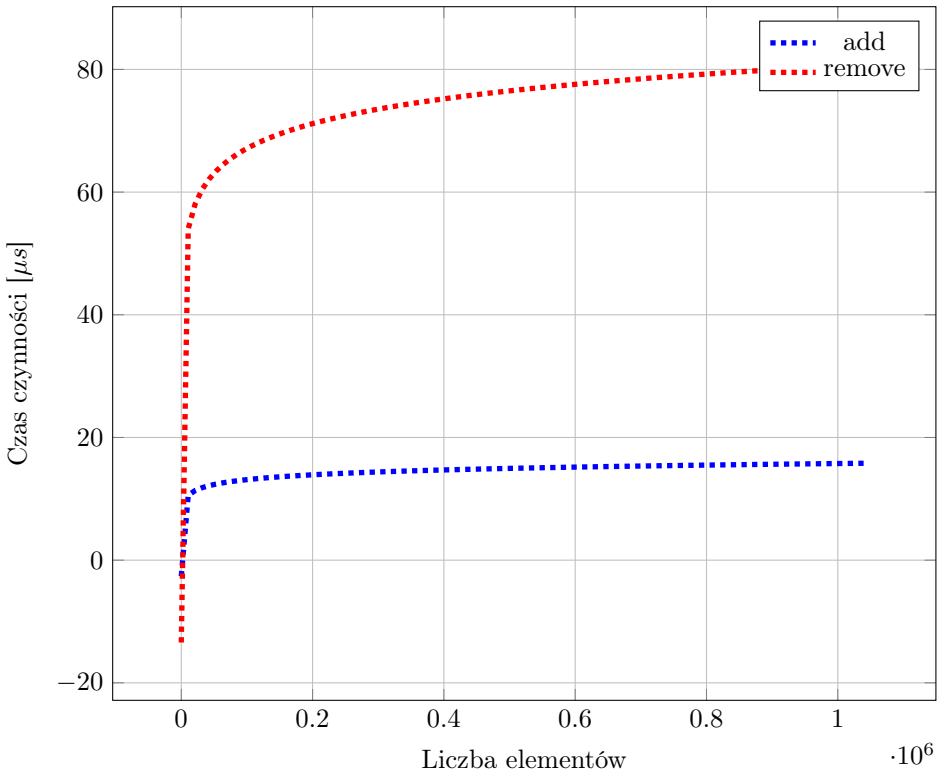
Przeszukiwanie kopca można również zaimplementować wykorzystując własność kopca tj. jeżeli wyszukiwany element jest większy od korzenia wówczas można przeszukiwania zakończyć. W przeciwnym przypadku należy (np. rekurencyjnie) przeszukać drzewa mające korzenie w liściach (węzłach potomnych) rozważanego węzła. Zarówno złożoność średnia jak i pesymistyczna takiego postępowania jest rzędu $O(n)$, dodatkowo wyznaczenie istnienia i pozycji potomków zajmuje dodatkowy czas w porównaniu do przeglądania elementów tablicy implementującej kopiec po kolei. Z tego punktu widzenia zaniechano implementacji i testowania tego podejścia.

7.4 Porównanie operacji

Na Rysunku 20 zestawiono ze sobą średnie czasy ww. operacji na kopcu. Ze względu na znacznie dłuższy czas operacji wyszukiwania względem operacji dodawania czy usuwania elementu z kopca, wyszukiwanie nie zostało przedstawione na wykresie.

7.5 Wnioski

Przyglądając się operacji dodawania i usuwania elementu do struktury można z łatwością dostrzec, że dodawanie jest średnio sześć, siedem razy krótsze od usuwania. Sytuacja ta jest spowodowana koniecznością naprawy kopca w dół. Naprawa kopca w dół wymaga wyznaczenia pozycji dwóch synów i w związku z tym porównania dwóch wartości. W przypadku naprawy w górę wyznaczamy tylko pozycję rodzica. Wyszukiwanie elementu jest najdłuższą operacja testowaną na kopcu i jako jedyna jest liniowa. Jest to spowodowane koniecznością szukania element po elemencie co w najgorszym przypadku skutkuje przeszukaniem całej tablicy, na której jest zaimplementowany kopiec. Porównując ze sobą czasy



Rysunek 20: Zestawienie zależnośći czasu wykonywania operacji na kopcu

wszystkich operacji można wywnioskować, że kopiec nie nadaje się do implementacji, w sytuacjach gdy istotne jest wyszukiwanie elementów.

8 Drzewo przeszukiwań binarnych

Drzewo przeszukiwań binarnych jest strukturą w postaci drzewa binarnego. Zaimplementowana jest z użyciem elementów, które poza wartością zawierają wskaźnik na rodzica oraz dwóch potomków. Cechą drzewa BST jest fakt, iż lewy potomek musi być mniejszy, natomiast prawy potomek musi być większy od niego.

Celem zadania projektowego było zbadanie czasu obliczeń w zależności od liczby elementów w strukturze dla ww. struktury. Na strukturze przebadano następujące operacje, których teoretyczna złożoność czasowa wynosi:

- dodawanie elementu - $O(n)$ (optymistyczna $O(1)$, pesymistyczna $O(n)$, średnia $O(log(n))$),
- usuwanie elementu - $O(n)$ (optymistyczna $O(1)$, pesymistyczna $O(n)$, średnia $O(log(n))$),
- wyszukiwanie elementu - $O(n)$ (optymistyczna $O(1)$, pesymistyczna $O(n)$, średnia $O(log(n))$).

Badania przeprowadzono dla początkowej liczby elementów wynoszącej od 5000 do 100 000 ze skokiem co 5000. W Tabeli 5 przedstawiono średnie czasy działania funkcji add, remove, find. Natomiast na Rysunkach 21, 22, 23 przedstawiono wartość minimalną, średnią oraz maksymalną czasu operacji w zależności od liczby elementów oraz zależność czasu obliczeń od rozmiaru problemu wyestymowaną przy pomocy metody najmniejszych kwadratów.

Tabela 5: Pomiary czasów wykonywania operacji na drzewie BST

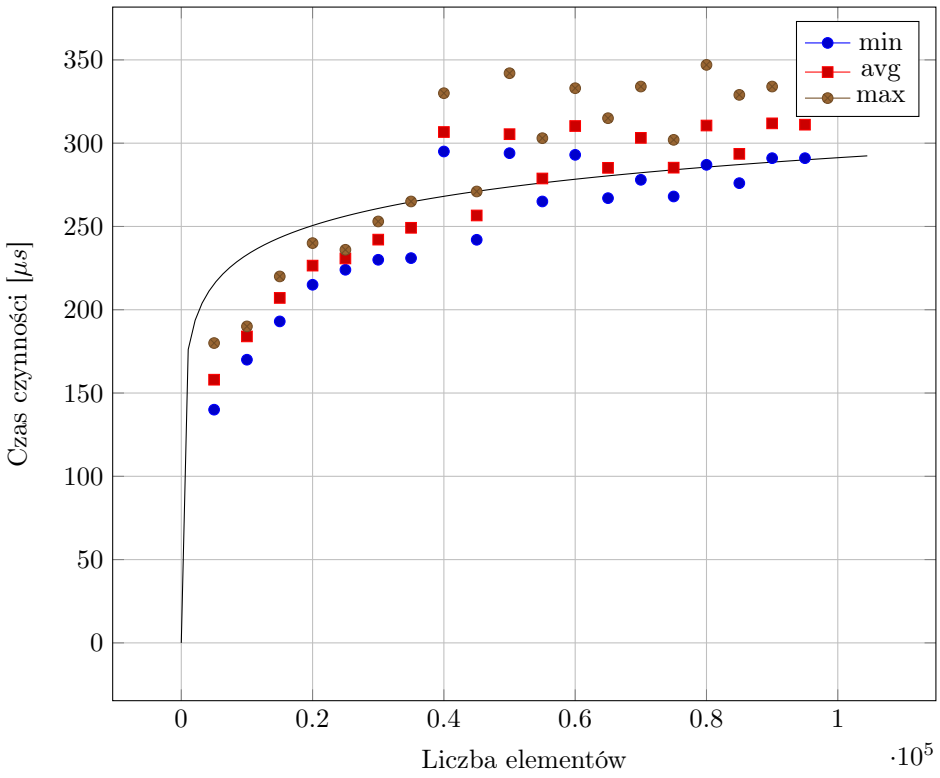
n	Add $[\mu s]$	Delete $[\mu s]$	Find $[\mu s]$
50000	264.6	52.6	122.8
100000	313.3	53.5	153.7
150000	335.8	56.1	192.6
200000	377.4	58.2	219.0
250000	408.3	64.6	242.3
300000	436.3	68.1	279.7
350000	467.0	77.5	294.1
400000	496.4	81.4	337.7
450000	509.0	84.3	327.3
500000	537.4	89.2	333.6
550000	559.5	93.7	352.7
600000	575.5	92.2	377.1
650000	599.9	97.2	382.5
700000	602.6	109.5	422.2
750000	545.7	110.5	409.9
800000	561.3	114.3	436.7
850000	562.2	114.5	467.9
900000	580.5	112.6	490.3
950000	589.7	115.7	476.1

8.1 Dodawanie elementu do drzewa BST

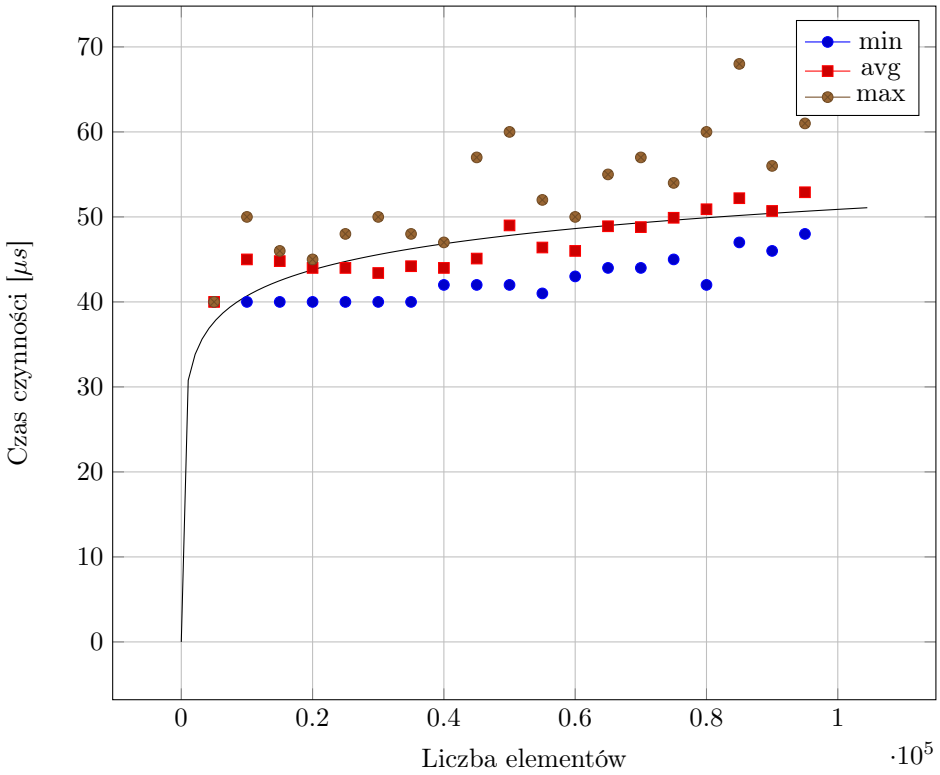
Obserwując dane znajdujące się na Rysunku 21 można dostrzec, że czas wykonania operacji nieznacznie wzrasta wraz ze wzrostem liczby elementów w strukturze. Krzywa logarytmiczna wyestymowana metodą najmniejszych kwadratów dla wartości średnich cechowała się najmniejszym błędem, zatem wyniki pomiarów potwierdzają średnią złożoność czasową operacji dodawania do drzewa binarnego.

8.2 Usuwanie korzenia z drzewa BST

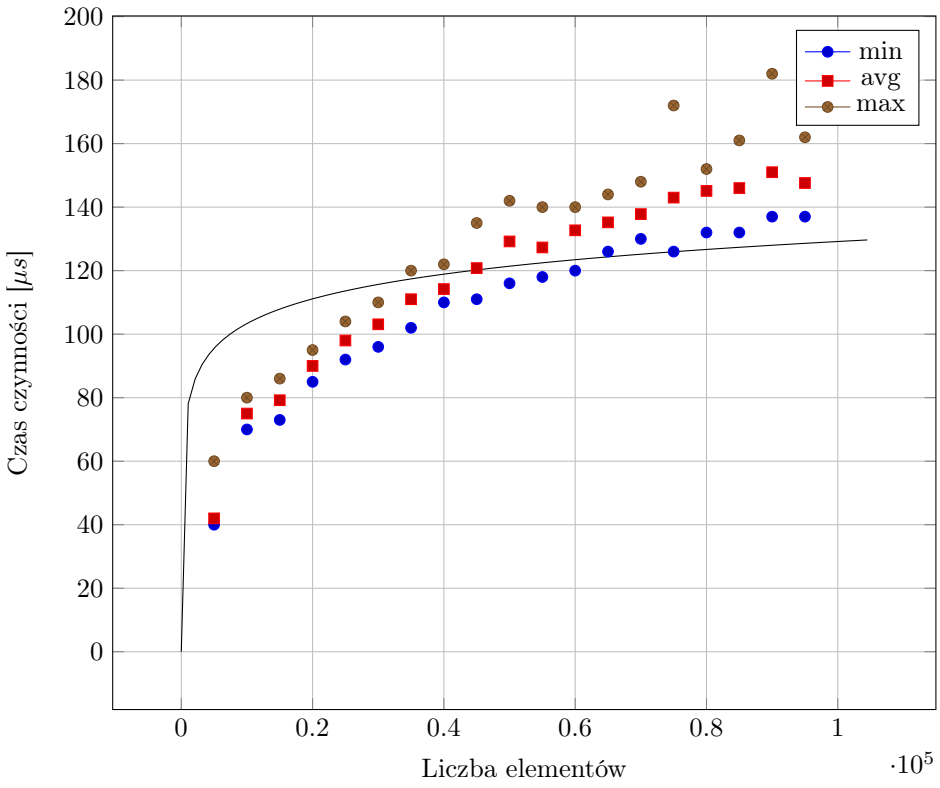
Wykres usuwania elementów z BST został przedstawiony na Rysunku 22.



Rysunek 21: Wykres zależności czasu dodawania elementów do drzewa przeszukiwań binarnych od liczby elementów w strukturze



Rysunek 22: Wykres zależności czasu usuwania elementów z drzewa przeszukiwań binarnych od liczby elementów w strukturze



Rysunek 23: Wykres zależności czasu wyszukiwania elementów w drzewie przeszukiwań binarnych od liczby elementów w strukturze

8.3 Wyszukiwanie elementu

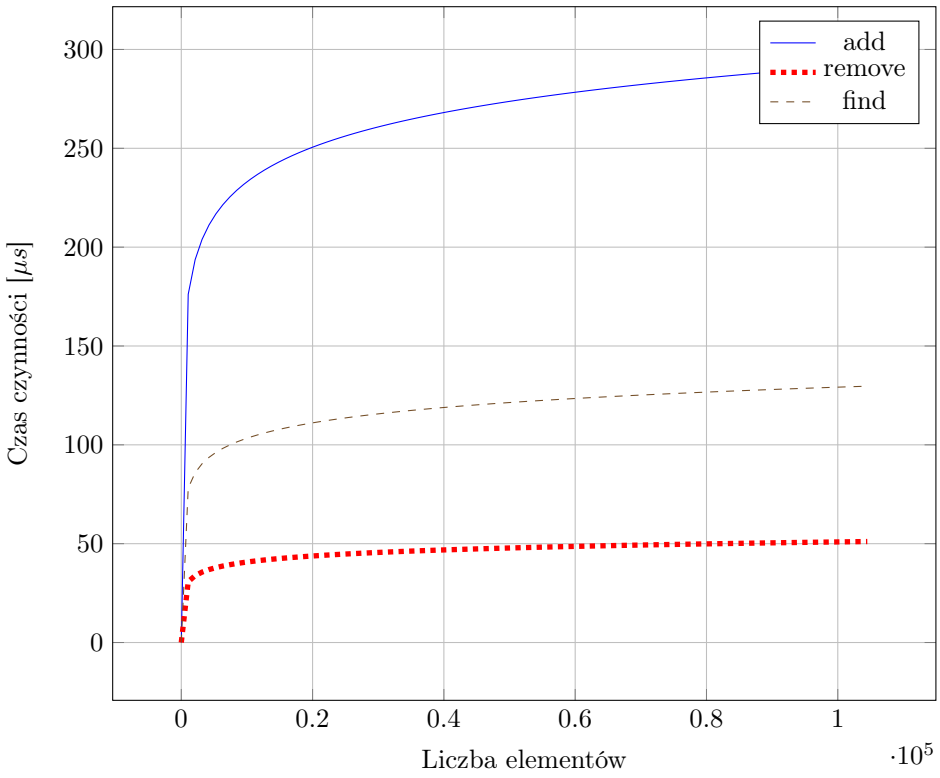
Ostatnią testowaną operacją na BST jest wyszukiwanie w wybranego elementu. Zestawienie czasów dotyczących tej operacji zestawiono na Rysunku 23.

8.4 Porównanie

Na Rysunku 24 zestawiono ze sobą krzywe wyznaczone metodą najmniejszych kwadratów dla średnich czasów wykonywania ww. operacji na drzewie przeszukiwań binarnych. Najdłużej trwa dodawanie, które jest nawet sześć razy dłuższe od usuwania korzenia.

8.5 Wnioski

Porównując krzywe z wykresu z Rysunku 24 można dostrzec spore różnice czasu wykonywania poszczególnych operacji. Szczególnie ciekawą zależność można dostrzec porównując dodawanie i usuwanie elementów, gdzie dodawanie elementu trwa około sześć razy dłużej niż usuwanie elementu znajdującego się w korzeniu drzewa. Jest to spowodowane między innymi tym, że w przypadku usuwania elementu z korzenia drzewa wyszukanie elementu do usunięcia wykonywane jest w czasie O(1) ponieważ wyszukiwany element znajduje się w korzeniu drzewa. Powyższa struktura nie jest równoważona, spowodowane jest to ogromną czasochłonnością operacji równoważenia drzewa BST, która polega na wyprostowaniu drzewa, a następnie wykonania odpowiednich rotacji, co skutkuje złożonością obliczeniową O(nlogn). Struktura ta jest przydatna, gdy nie istotne jest zrównoważenia drzewa, gdy jest ono istotne lepiej zaimplementować drzewo RB lub AVL.



Rysunek 24: Zestawienie zależnośći czasu wykonywania operacji w drzewie przeszukiwań binarnych

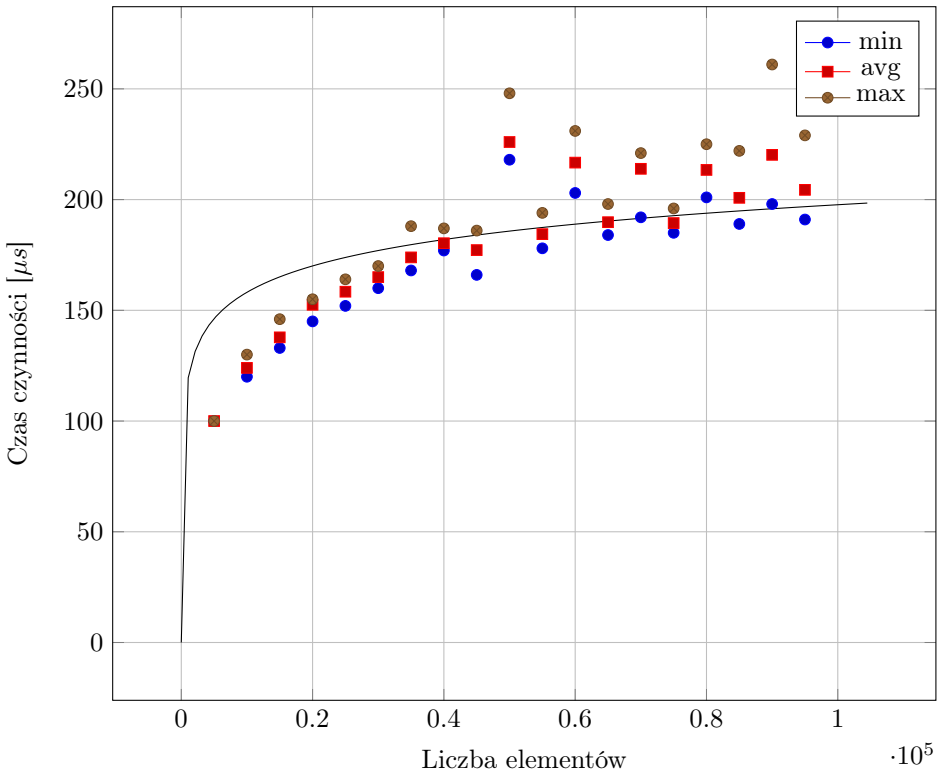
9 Drzewo czerwono - czarne

Drzewo czerwono-czarne (RB) jest strukturą w postaci drzewa binarnego. Zaimplementowany jest z użyciem elementów, które poza wartością zawierają wskaźnik na rodzica oraz dwóch potomków i w odróżnieniu od drzewa BST, kolor (czerwony lub czarny). Cechą drzewa RB jest fakt, iż lewy potomek musi być mniejszy, natomiast prawy potomek musi być większy od niego. Kolory przyporządkowane węzłom służą do kontroli wyważenia drzewa i uruchomienia naprawy drzewa polegającej na rotacjach węzłów w drzewie poprawiających stopień wyważenia drzewa. Naprawa drzewa następuje w przypadku dodawania i usuwania elementów do i z struktury.

Celem zadania projektowego było zbadanie czasu obliczeń w zależności od liczby elementów w strukturze dla ww. struktury. Na strukturze przebadano następujące operacje, których teoretyczna złożoność czasowa wynosi:

- dodawanie elementu - $O(log(n))$,
- usuwanie elementu - $O(log(n))$,
- wyszukiwanie elementu - $O(log(n))$.

Badania przeprowadzono dla początkowej ilości elementów wynoszącej od 5000 do 100 000 ze skokiem 5000. W Tabeli 6 przedstawiono średnie czasy działania funkcji add, remove, find. Natomiast na Rysunkach 25, 26, 27 przedstawiono wartość minimalną, średnią oraz maksymalną czasu operacji w zależności od liczby elementów oraz zależność czasu obliczeń od rozmiaru problemu wyestymowaną przy pomocy metody najmniejszych kwadratów.



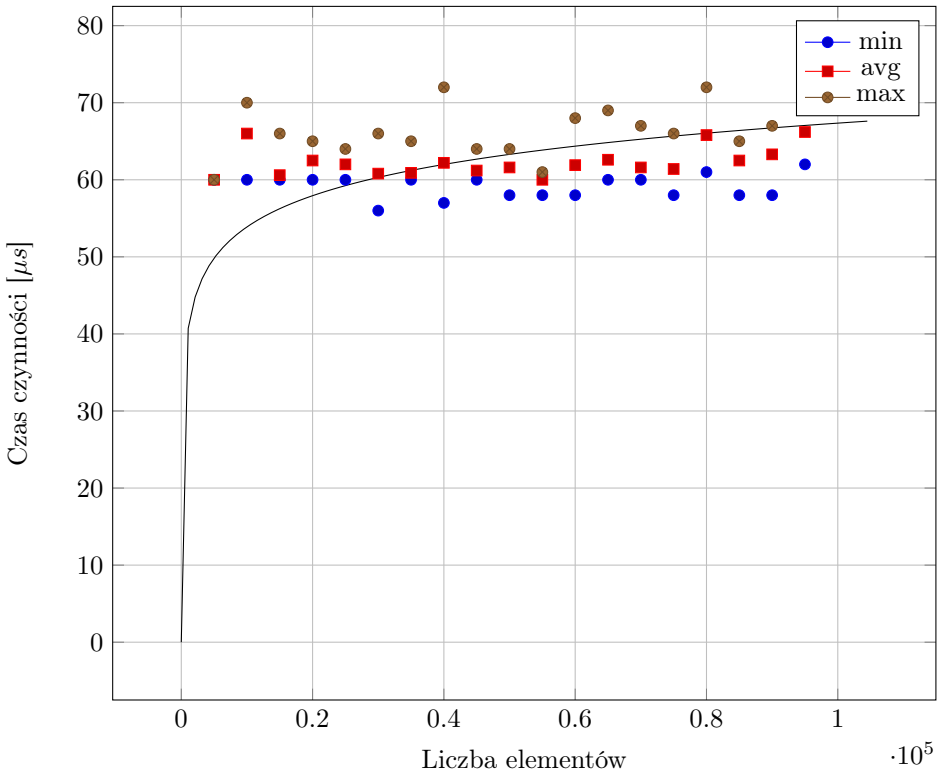
Rysunek 25: Wykres zależności czasu dodawania elementów do drzewa czerwono-czarnego od liczby elementów w strukturze

Tabela 6: Pomiary czasów wykonywania operacji na drzewie RB

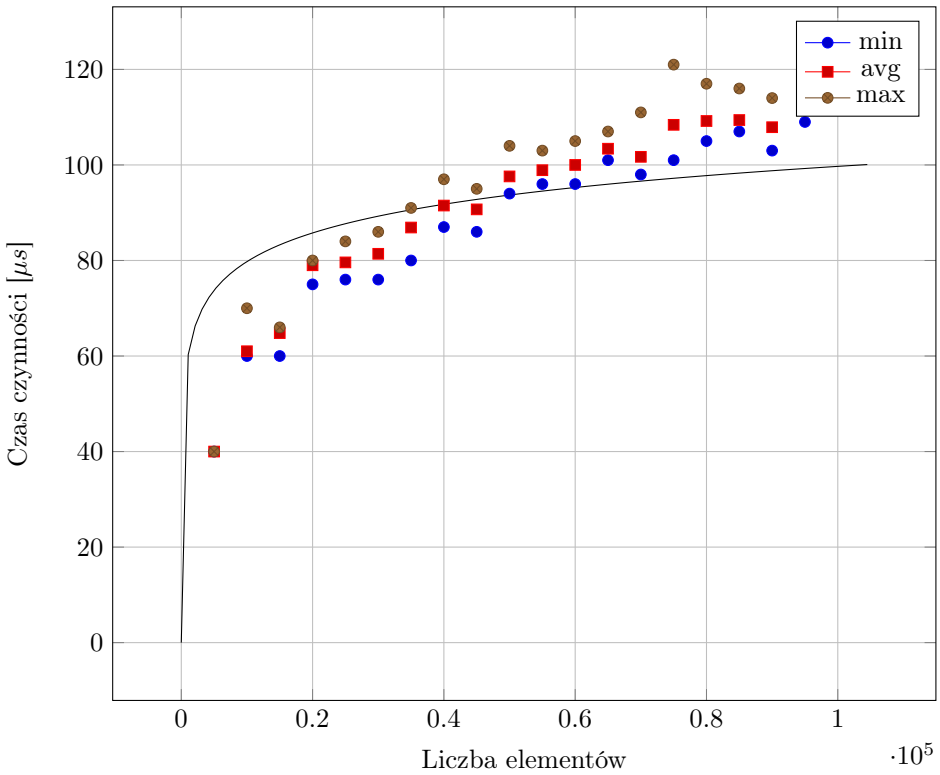
n	Add [μs]	Delete [μs]	Find [μs]
50000	205.0	60.0	93.4
100000	211.6	64.0	107.8
150000	228.8	69.3	114.1
200000	250.7	78.3	128.6
250000	272.7	83.4	139.9
300000	310.8	83.7	145.4
350000	338.0	87.6	155.7
400000	365.3	92.1	173.4
450000	382.2	94.4	179.8
500000	416.1	96.5	199.1
550000	429.4	98.3	206.0
600000	437.4	98.5	213.1
650000	460.6	100.7	232.0
700000	465.2	100.8	239.2
750000	475.3	102.5	244.7
800000	502.0	103.1	259.6
850000	520.6	101.9	276.8
900000	512.8	97.8	266.5
950000	541.8	101.2	277.4

9.1 Dodawanie elementu do drzewa RB

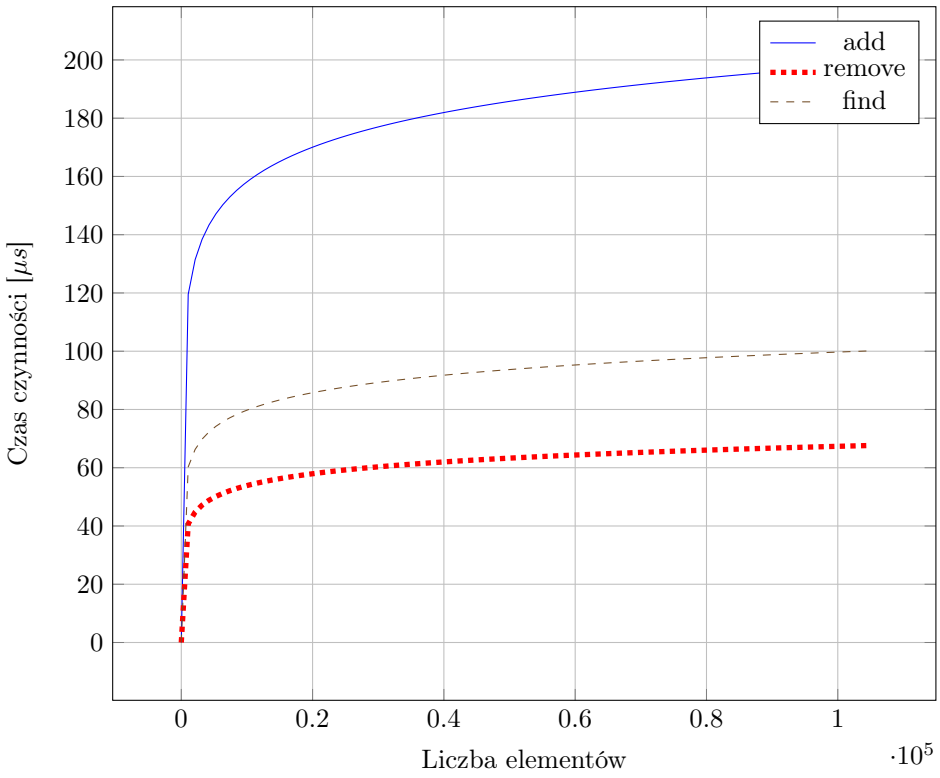
Spoglądając na wykres znajdujący się na Rysunku 25 można zauważyć, że punkty wyznaczające zależność czasu wykonywania operacji od liczby elementów w strukturze rośnie logarytmicznie.



Rysunek 26: Wykres zależności czasu usuwania elementów z drzewa czerwonow-czarnego od liczby elementów w strukturze



Rysunek 27: Wykres zależności czasu wyszukiwania elementów w drzewie czerwono-czarnym od liczby elementów w strukturze



Rysunek 28: Zestawienie zależnośći czasu wykonywania operacji w drzewie czerwono-czarnym

9.2 Usuwanie elementu z drzewa RB

Wykres usuwania elementów z drzewa czerwono-czarnego, został przedstawiony na Rysunku 26.

9.3 Wyszukiwanie elementu

Ostatnią testowaną operacją na drzewie czerwono-czarnym jest wyszukiwanie w nim zadanego elementu. Wykres przedstawiający tą operacje został przedstawiony na Rysunku 27. Jak można zauważyć jest zbliżony do krzywej logarytmicznej wyznaczonej za pomocą metody najmniejszych kwadratów.

9.4 Porównanie

Na Rysunku 28 zestawiono ze sobą średnie czasy ww. operacji na drzewie RB. W przeciwieństwie do drzewa BST różnica średnich czasów pomiędzy usuwaniem korzenia, a dodawaniem elementu jest ponad dwukrotnie mniejsza. Czas wyszukiwania jest nieco dłuższy od czasu usuwania korzenia.

9.5 Wnioski

Drzewo czerwono-czarne jest bardzo zbliżone do drzewa BST. Różnica pomiędzy tymi drzewami polega na tym, iż drzewo RB jest zrównoważone, więc operacje wyszukiwania elementów następują szybciej. Wnioski dotyczące dodawania, usuwania czy wyszukiwania są bardzo zbliżone do BST przez co nie będą rozpatrywane.

10 Porównanie efektywności czasowej operacji na strukturach danych

W sekcji przedstawiono porównanie efektywności czasowej podstawowych operacji na strukturach danych. Celem wskazania najbardziej efektywnych czasowo struktur. Na Rysunku 29 przedstawiono czasy dodawania elementów do listy, kopca, drzewa BST oraz drzewa RB. Natomiast na Rysunku 30 przedstawiono czasy usuwania elementów z tych struktur. Na Rysunku 31 przedstawiono czasy wyszukiwania elementów w drzewie BST i RB, ze względu na długi czas (liniowo zależny od liczby elementów) nie przedstawiono wykresów przedstawiających zależność czasów wyszukiwania w liście oraz kopcu.

Porównując średnie czasy dodania elementów do struktur, należy zauważyć, że najkrócej wykonywana jest czynność dodawania elementu do kopca, natomiast najdłużej do drzewa BST. Pewną anomalią jest dłuższy czas dodawania do listy niż do kopca, jednak jest to spowodowane koniecznością alokacji pamięci przy każdym dodaniu elementu do listy. Czas ten również wpływa na zmniejszenie efektywności struktur BST i RB, ponieważ dodawane elementy również alokowane są dynamicznie. Czas dodania elementów do najbardziej efektywnego drzewa (RB) jest około 8 - krotnie dłuższy od czasu dodania do listy dwukierunkowej dla liczby elementów w drzewie bliskiej 100 000 (dla mniejszych liczb jest on nieznacznie mniejszy).

Najszybciej usuwane są elementy z listy. Czas usunięcie elementu z listy wynosi około 15 microsekund i jest on mniej więcej czterokrotnie mniejszy niż czas usunięcia z najmniej-efektywnej struktury pod tym aspektem czyli RB. Usunięcie elementu z kopca i drzewa BST zajmuje podobną ilość czasu i jest około trzy razy dłuższe od czasu usunięcia z listy.

Porównując wykres wyestymowanych czasów wyszukania elementów w drzewie BST i RB należy zauważyć, ze wyszukiwanie w drzewie RB jest istotnie krótsze niż wyszukiwanie w drzewie BST, co jest spowodowane zrównoważeniem drzewa. Dla liczby elementów w strukturze równej 100 000 różnica ta jest największa i wynosi około 30

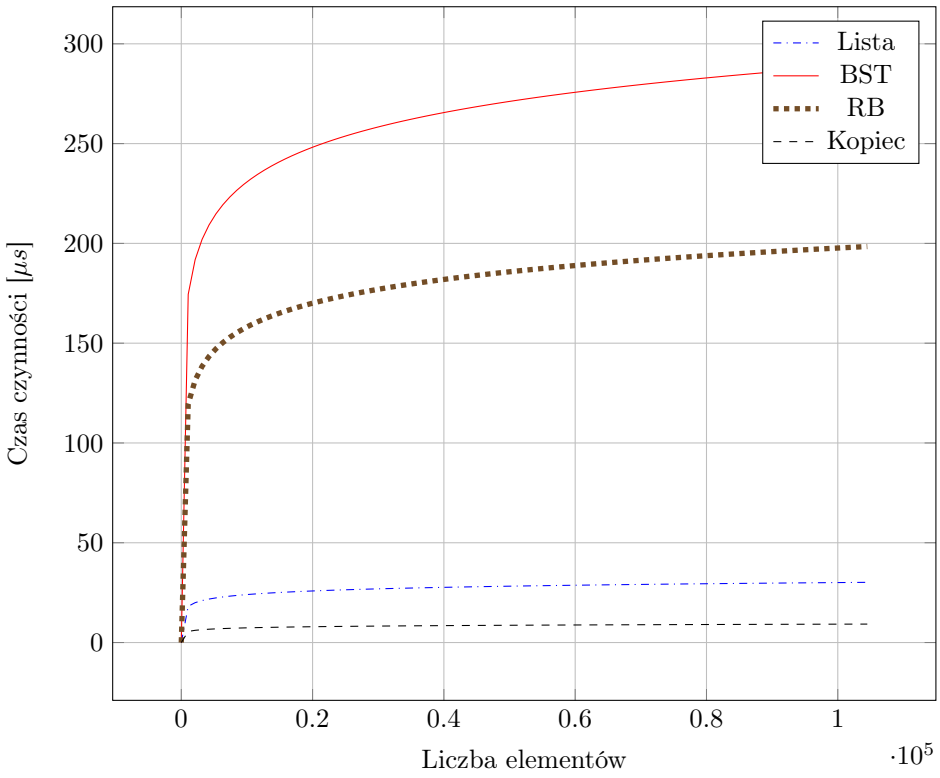
Podsumowanie

Najefektywniejszą strukturą do przechowywania danych alokowanych dynamicznie jest lista, której operacje dodawani i usuwania elementów na koniec lub początek są wykonywane najszybciej. Może być ona podstawą implementacji takich struktur danych jak kolejka i stos oraz zbiorów z danymi, których nie jest istotna kolejność elementów. Struktura nie nadaje się do wyszukiwania elementów, ponieważ istnieją inne struktury znacznie efektywniejsze.

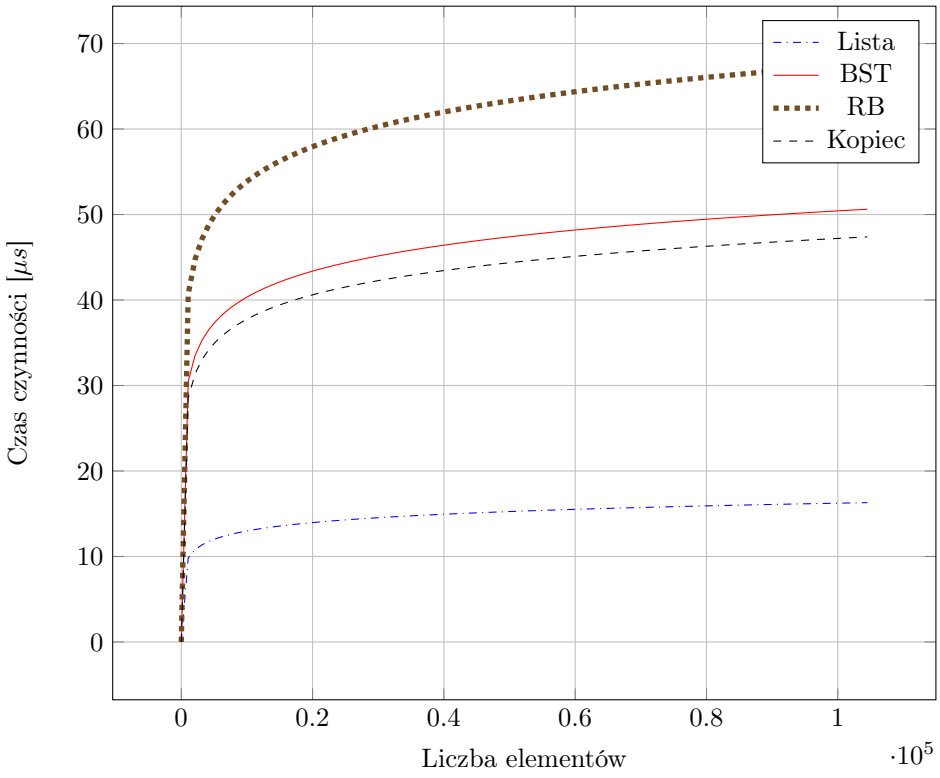
Jeżeli chodzi o wyszukiwanie danych najefektywniejszymi strukturami są struktury drzewiaste BST i RB. Dodawanie elementów i wyszukiwanie w drzewie RB jest istotnie krótsze niż w drzewie BST. Usuwanie elementów z drzewa BST jest nieznacznie krótsze niż w przypadku drzewa RB. Zatem w przypadku wykorzystania struktur danych do częstego wyszukiwania danych lub porządkowania, najlepszą strukturą spośród badanych jest drzewo RB.

11 Bibliografia

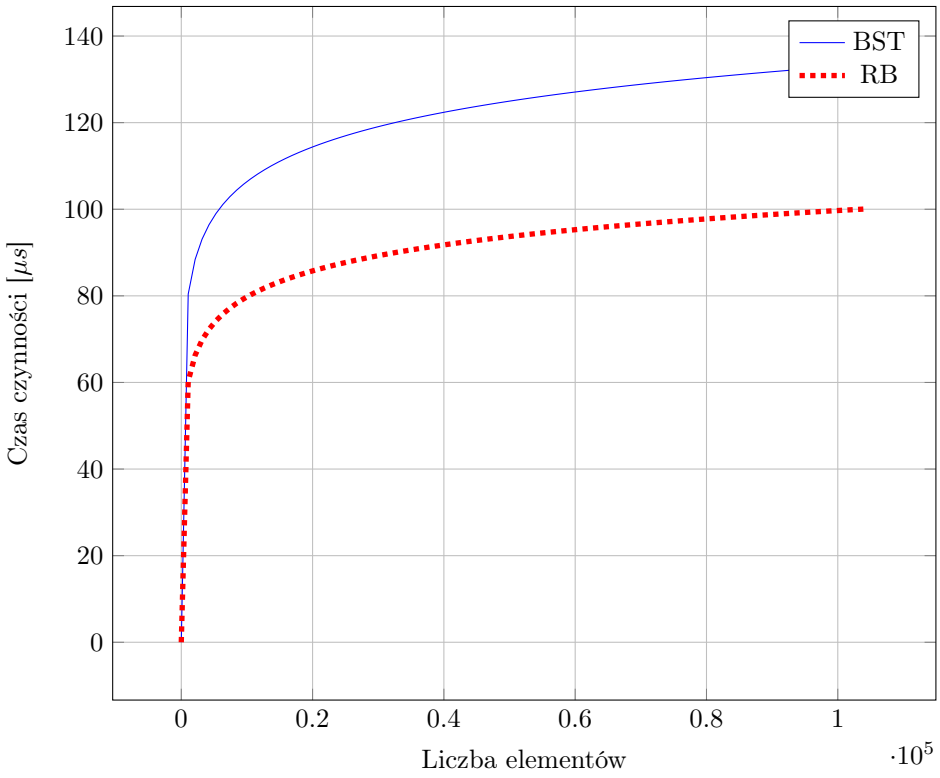
- https://www.algorytm.edu.pl/matura-informatyka/zlozonosc-algorytmu
- Cormen Thomas H., Leiserson Charles E., Rivest Ronald L, Clifford Stein, Wprowadzenie do algorytmów, Wydawnictwo Naukowe PWN, Warszawa, 2022.



Rysunek 29: Zestawienie zależności czasu dodawania elementów do listy, drzewa BST, RB i kopca



Rysunek 30: Zestawienie zależności czasu usuwania elementów z listy, drzewa BST, RB i kopca



Rysunek 31: Zestawienie zależności czasu wyszukiwania elementów w liście, drzewie BST, RB i kopcu