Struktury Danych i Złożoność Obliczeniowa

Zadanie projektowe nr 1: Badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w różnych strukturach danych.

Prowadzący: Dr inż. Zbigniew Buchalski
Wykonał: Miron Oskroba, 236705

Spis treści

1. Opis zadania projektowego

1.1 Cel projektu

Celem zadania projektowego było zaimplementowanie różnego typu struktur danych, oraz funkcji wykonujących na nich operacje dodawania elementów, usuwania elementów, oraz wyszukiwania elementu zawierającego podaną wartość. Zaimplementowane zostały następujące struktury:

1)Tablica
2)Lista dwukierunkowa
3)Kopiec binarny
4)Drzewo BST

2. Założenia projektowe

Podczas pisaniu programu w języku C++, zaimplementawano podane struktury. Zachowane zostały zasady podane w założeniu projektowym:

a) Kopiec binarny jest typu maksimum – element maksymalny jest w korzeniu

b) Podstawową liczbą używaną przy tworzeniu struktur jest 4bajtowa liczba całkowita – integer

c) Wszystkie struktury danych są alokowane dynamicznie

d) Tablica służąca do przechowywania wartości kopca nie jest relokowana po każdej operacji, przyjmuje stałą wartość nadmiarową, aby można było na niej wykonać operacje dodawania

e) W przypadku tablicy i listy rozpatrzono osobno operacje dodawania i usuwania elementu na następujących pozycjach:

1. początek tablicy/listy
2. koniec tablicy/listy
3. losowe miejsce tablicy/listy

f) W przypadku drzewa BST zaimplementowany został algorytm równoważenia drzewa, który jest automatycznie wywoływany po operacjach takich jak: dodawanie elementu, usuwanie elementu

g) Utworzenie struktury z liczb zapisanych w pliku tekstowym – każda liczba jest w osobnej linii, natomiast pierwsza liczba określa ilość zapisanych liczb w pliku.

h) Użytkownik jest pytany o nazwę pliku do wczytania

i) Przy tworzeniu struktury z pliku poprzednie dane są usuwane

j) W przypadku tablic i listy kolejność danych jest taka sama jak w pliku. W przypadku drzew, kolejne elementy z pliku są kolejno dodawane w odpowiednie miejsce drzewa, odpowiednie operacje są automatycznie wywoływane aby zachować poprawną strukturę

k) W przypadku drzew wyświetlane są w przejrzystej formie, w przypadku kopca jest dodatkowo wyświetlany w postaci tablicy
l) Możliwość wykonania operacji na strukturze z tym, że w przypadku:

1. Usuwania z tablicy - zostanie podany indeks liczby, którą należy usunąć, indeksowanie od zera
2. Wstawianie do tablicy – zostanie podany indeks i wartość, którą należy wstawić na podaną pozycję. W razie konieczności tablica jest 'rozsuwana' – w menu nie zrobiono dodatkowej pozycji dodawania na początek i koniec
3. Usuwanie z listy – zostanie podana tylko wartość, którą należy usunąć
4. Dodawanie do listy – zostanie podana wartość, którą należy usunąć
5. Usuwanie i dodawanie dla drzew – zostanie podany klucz, który należy usunąć lub dodać. Dla kopca zostało wdrożone identyczne postępowanie.

f. Wyszukiwanie(dla wszystkich struktur) – zostanie podana liczba, którą należy znaleźć – wyświetlona zostanie wówczas informacja czy liczba jest w strukturze, czy nie

m) Nie użyto gotowych rozwiązań, jedynie podstawowe biblioteki języka programowania C++
n)Program napisano w wersji obiektowej
o)Wygenerowano plik .exe

3. Menu programu

Program został napisany konsolowo, dlatego opisane operacje zrealizowano w formie menu dla każdej struktury. W tym podrozdziale opisane zostanie menu każdej ze struktur oraz menu główne programu. Każda struktura zawiera dodatkowe opcje: powrót do menu oraz zapisz do pliku.

3.1 Menu ---Główne---

1. Tablica
2. Lista
3. Kopiec
4. Drzewo BST
5. Wyjście z programu

3.2 Menu ---Tablica---

1. Zbuduj z pliku – elementy pojawiają się w tablicy w takiej kolejności jak w pliku.

2. Stwórz losowo – użytkownik zostaje zapytany o ilość elementów, następnie jest tworzona losowa tablica o podanej długości.

3. Dodaj element na pozycje – użytkownik zostaje zapytany o indeks, następnie o wartość, jeżeli wartości te są poprawne, program wstawia podaną wartość na podany indeks jednocześnie 'rozsuwając' tablicę. Zwiększa się jej długość.

4. Usuń element z pozycji – użytkownik zostaje zapytany o indeks, jeżeli indeks jest poprawny, program automatycznie usuwa wartość pod wskazanym indeksem 'zsuwając' tablicę, zmiejszając jednocześnie jej długość.

5. Znajdź element – użytkownik jest pytany o wartość do znalezienia, algorytm

szukania znajduje podaną wartość lub nie i wypisuje stosowny komunikat.

6.Wyświetl – wyświetla tablicę w jednej linii, elementy są rozdzielone spacją.

7. Usuń tablicę – usuwa tablicę, ustawia jej długość na 0

0. Powrót do menu

s. Zapisz do pliku – opcja dodatkowa, zapisuje tablicę zgodnie z założeniami projektowymi do pliku (pierwsza wartość w pliku to długość tablicy, następnie w nowych liniach są kolejne elementy)

3.3 Menu ---Lista---

1. Zbuduj z pliku - elementy pojawiają się w tablicy w takiej kolejności jak w pliku.

2. Stworz losowo - użytkownik zostaje zapytany o ilość elementów, następnie jest tworzona losowa lista o podanej długości.

3. Dodaj element na pozycję – użytkownik jest pytany o indeks oraz wartość dodawanego elementu, jeżeli wartości te są poprawne, lista jest 'rozsuwana', element wstawiany a długość listy inkrementowana.

4. Usuń element z pozycji – użytkownik jest pytany o indeks usuwanego elementu, jeżeli jest on poprawny, element jest usuwany, lista 'zsuwana', a długość listy dekrementowana

5. Znajdź element – użytkownik jest pytany o wartość do znalezienia, program przyrównuje szukaną wartość do kolejnych wartości listy, jeżeli wartości będą sobie równe, przerywa szukanie i wypisuje stosowny komunikat.

6. Wyświetl – lista jest wyświetlana od początku do końca, elementy rozdzielone spacją, następnie znak nowej linii jest dodawany a lista ponownie wyświetlana lecz tym razem od końca do początku, elementy również rozdzielone spacją.

7. Usuń listę

0. Powrót do menu

s. Zapisz do pliku

3.4 Menu ---Kopiec---

1. Zbuduj z pliku – buduje z pliku kopiec maksymalny, kopiec jest naprawiany po każdy kolejnym wczytywanym elemencie.

2. Stwórz losowo – zapełnia losowymi wartościami tablicę kopca, kopiec jest naprawiany w górę

3. Dodaj klucz – użytkownik zapytany o wartość, jeżeli jest poprawna klucz jest

wpisywany w odpowiednie miejsce w kopcu i wykonuje odpowiednie zamiany jeżeli wystąpiła taka potrzeba

4. Usuń klucz – użytkownik zapytany o wartość do usunięcia, jeśli jest poprawna, automatycznie uruchamiany jest inteligentny algorytm sprawdzający które 'poddrzewa' należy przeszukać, znajduje ostatni powtarzający się element i usuwa go. Automatycznie na jego miejscu pojawia się odpowiedni element.

5. Znajdź klucz – użytkownik zapytany o klucz do znalezienia. Znajduje pierwszy element, wypisuje stosowny komunikat

6. Wyświetl - wyświetlane są dwie reprezentacje kopca – najpierw tablica, w której elementy rozdzielone są spacją, następnie w nowej linii wyświetlana jest 'drzewiasta', przejrzysta struktura kopca. Wyświetlany w ten sposób kopiec zostaje wyświetlony z rotacją o 90stopni przeciwnie do ruchu wskazówek zegara.

7. Usuń kopiec – usuwa elementy tablicy kopca, ustawia jego długość na 0

0. Powrót do menu

s. Zapisz do pliku

3.5 Menu ---Drzewo BST---

1. Zbuduj z pliku – użytkownik zapytany o pełną nazwę pliku z danymi, po poprawnym wczytaniu pliku, elementy są kolejno dodawane w odpowiednie miejsca w drzewie, odpowiednie rotacje automatycznie wykonywane w razie zaistniałej potrzeby. Uwaga: Po takim wczytaniu pliku drzewo nie jest zrównoważone, można to zrobić ręcznie używając innej opcji zawartej w owym menu.

2. Stwórz losowo – tworzy losowe drzewo. Uwaga: Po takim wczytaniu pliku drzewo nie jest zrównoważone, można to zrobić ręcznie używając innej opcji zawartej w owym menu.

3. Dodaj wierzchołek – użytkownik zostaje zapytany o wartość do dodania. Wartość jest dodawana, następnie automatycznie drzewo jest równoważone algorytmem DSW.

4. Usuń wierzcholek – użytkownik zostaje zapytany o wartość, która ma zostać usunięta. Inteligentny algorytm znajduje ostatnią powtarzającą się wartość, po czym usuwa ją i automatycznie wykonuje odpowiednie operacje tak, że własności drzewa są zachowane. Następnie automatycznie drzewo jest równoważone algorytmem DSW.

5. Znajdź wierzchołek – użytkownik zostaje zapytany o wartość, która ma zostać znaleziona. Algorytm szuka pierwszą powtarzającą się wartość i wypisuje stosowny komunikat.

6. Wyświetl – wyświetlana jest struktura aktualnego drzewa z rotacją o 90 stopni przeciwnie do ruchu wskazówek zegara(tzn. Prawa strona drzewa jest na górze ekranu, a lewa strona drzewa na dole ekranu)

7. Usuń drzewo – usuwa korzeń oraz ustawia długość drzewa na 0

0. Powrót do menu

s. Zapisz do pliku – zapisuje najpierw ilość elementów do pierwszej linii, a następnie domyślnie zapisuje kolejne elementy algorytmem preorder.

4.Złożoności obliczeniowe poszczególnych operacji w zaimplementowanych strukturach

4.1 Opis ogólny

Złożoność obliczeniowa – ilość zasobów niezbędnych do wykonania programu wykonującego instrukcje. Wyróżniamy złożoność czasową oraz pamięciową.

Złożoność pamięciowa – ilość pamięci jaka jest wykorzystana do wykonania algorytmu lub ilość pamięci wykorzystana do przechowania pewnej określonej ilości danych. Często wyrażana w bajtach lub liczbie zmiennych jednego z typów elementarnych – w przypadku integer – 4 bajty, w przypadku short – 2 bajty itp. Jest to funkcja ściśle określona przez rozmiar danych na których realizowane są algorytmy.

Złożoność czasowa – czas potrzebny na wykonanie danego algorytmu, wyrażany często w podstawowych jednostkach czasu odpowiednio dopasowanych do rzędu wielkości czasu wykonywanej operacji, lub liczbie cykli procesora. Ściśle zależna od ilości danych oraz ich rozmiarów.

Dla algorytmu wyróżnić można także złożoności zależne od postaci wykonywanego algorytmu:

Złożoność optymistyczna i pesymistyczna – odpowiednio: najkorzystniejszy czas wykonania zbioru instrukcji dla odpowiednich danych wejściowych, w przypadku pesymistycznej jest to czas najdłuższy.

Złożoność średnia – estymowane średnie zużycie zasobów dla grupy losowych zbiorów danych. Średnia wszystkich pomiarów.

4.2. Złożoności operacji struktur

4.2.1 Złożoność operacji dla tablicy

Funkcja	Średnia	Pesymistyczna
Dodanie wartości	O(n)	O(n)
Usunięcie wartości	O(n)	O(n)
Wyszukanie wartości	O(n)	O(n)
Dostęp do elementu	O(1)	O(1)

Tabela 1. Złożoności obliczeniowe dla tablicy

4.2.2 Złożoność operacji dla listy dwukierunkowej

Funkcja	Średnia	Pesymistyczna
Dodanie wartości	O(-)	O(n)
Usunięcie wartości	O(-)	O(n)
Wyszukanie wartości	O(n)	O(n)
Dostęp do elementu	O(n)	O(n)

Tabela 2. Złożoności obliczeniowe dla listy

4.2.3 Złożoność operacji dla kopca bianrnego

Funkcja	Średnia	Pesymistyczna
Dodanie wartości	O(1)	O(1)
Usunięcie wartości	O(1)	O(1)
Wyszukanie wartości	O(n)	O(n)
Dostęp do elementu	O(n)	O(n)

Tabela 3. Złożoności obliczeniowe dla kopca

4.2.4 Złożoność operacji dla drzewa BST

Funkcja	Średnia	Pesymistyczna
Dodanie wartości	O(log(n))	O(log(n))
Usunięcie wartości	O(log(n))	O(log(n))
Wyszukanie wartości	O(log(n))	O(log(n))
Dostęp do elementu	O(log(n))	O(log(n))

Tabela 4. Złożoności obliczeniowe dla drzewa BST

4.2.5 Bibliografia:

https://en.wikipedia.org/wiki/Time_complexity
https://en.wikipedia.org/wiki/Complexity
https://pl.wikipedia.org/wiki/Tablica_(informatyka) https://www.samouczekprogramisty.pl/struktury-danych-lista-wiazana/
https://pl.wikipedia.org/wiki/Kopiec_binarny
https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwa%C5%84

5. Plan eksperymentu

Zawiera założenia co do wielkości struktur, sposobu generowania elementów tych strukur, sposobie pomiaru czasu.

Do pomiaru czasu została użyta biblioteka "chrono" zawarta w zbiorze standardowych bibliotek języka c++. Biblioteka operuje na czasie procesora, co pozwala na uzyskanie dokładności pomiarowej równej jednemu okresowi przebiegu procesora. Założona dokładność pomiarów to jedna nanosekunda.

```
chrono::high_resolution_clock::time_point start_time;
chrono::high_resolution_clock::time_point stop_time;
void startTimer()
{
    start_time = chrono::high_resolution_clock::now();
}
long stopTimer()
{
    stop_time = chrono::high_resolution_clock::now();
    return chrono::duration_cast<chrono::nanoseconds>(stop_time-start_time).count();
}
```

Zdjęcie 1 – Funkcje pomiaru czasu

Biblioteki chrono użyto także do generowania ziarna generatora liczb całkowitych:

```
unsigned get_seed()
{
    return chrono::system_clock::now().time_since_epoch().count();
}
```

```
    srand(time_t(get_seed()));
```

Zdjęcia 2 i 3 – funkcje generowania ziarna generatora liczb losowych

Do generowania liczb z z przedziału (min,max) zaimplementowana została funkcja get_rand_int(min,max)

```
//returns random integer from range min max
int get_rand_int(int min, int max)
{
    if(min==max) return max;
    if((min<0   && max<0) || (min>0 &&   max>0))//if min and max are same signs
        if(min  < 0)
            return rand() % (min-max) + min;//negative
        else
            return rand() % (max-min) + min;//positive
    else return rand() % (abs(max)+abs(min)) + min;//if different signs
}
```

Zdjęcie 4 – funkcja generowania losowej liczby całkowitej w przedziale

Do wszystkich operacji struktury będą przechowywać elementy typu integer z przedziału od 0 do 100.000 celem spowolnienia operacji. Przedstawione zostaną tabele oraz wykresy uśrednionych zależności czasu od ilości elementów w danej strukturze. Celem zautomatyzowania procesu tworzenia pomiarów stworzone zostały pliki z przykładowymi danymi o różnej ilości elementów z przedziału od 0 do 100tys. Ilość tych elementów na potrzeby testowania została opisana: 100,1000,2500,5000,10.000, 25.000. Wszystkie struktury zostały testowane na tych samych zbiorach plików celem zwiększenia wiarygodności otrzymanych wyników i zwiększenia prawdopodobieństwa na wyciągnięcie poprawnych wniosków.

6. Wyniki pomiarów

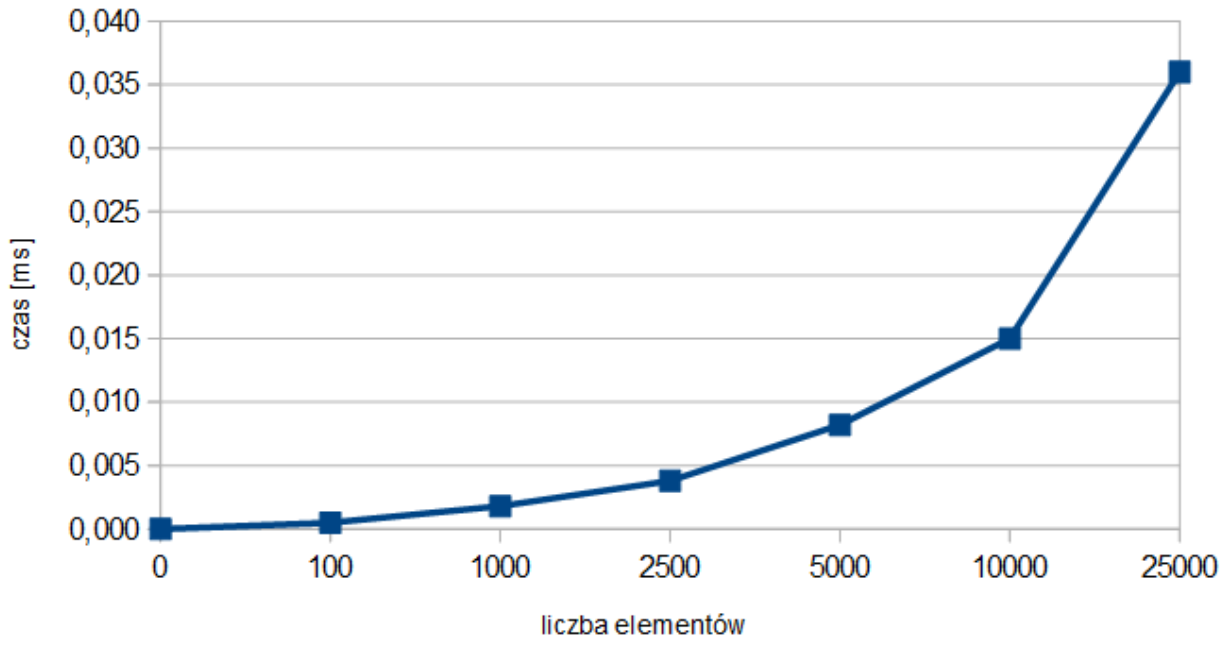
Dla wszystkich pomiarów użyto danych z zakresu 0-100000.

6.1 Tablica

6.1.1 Dodawanie na początek tablicy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-0,001ms]
1	100	0,001
2	1000	0,002
3	2500	0,004
4	5000	0,009
5	10000	0,015
6	25000	0,036

Tabela 5. Dodawanie na początek tablicy

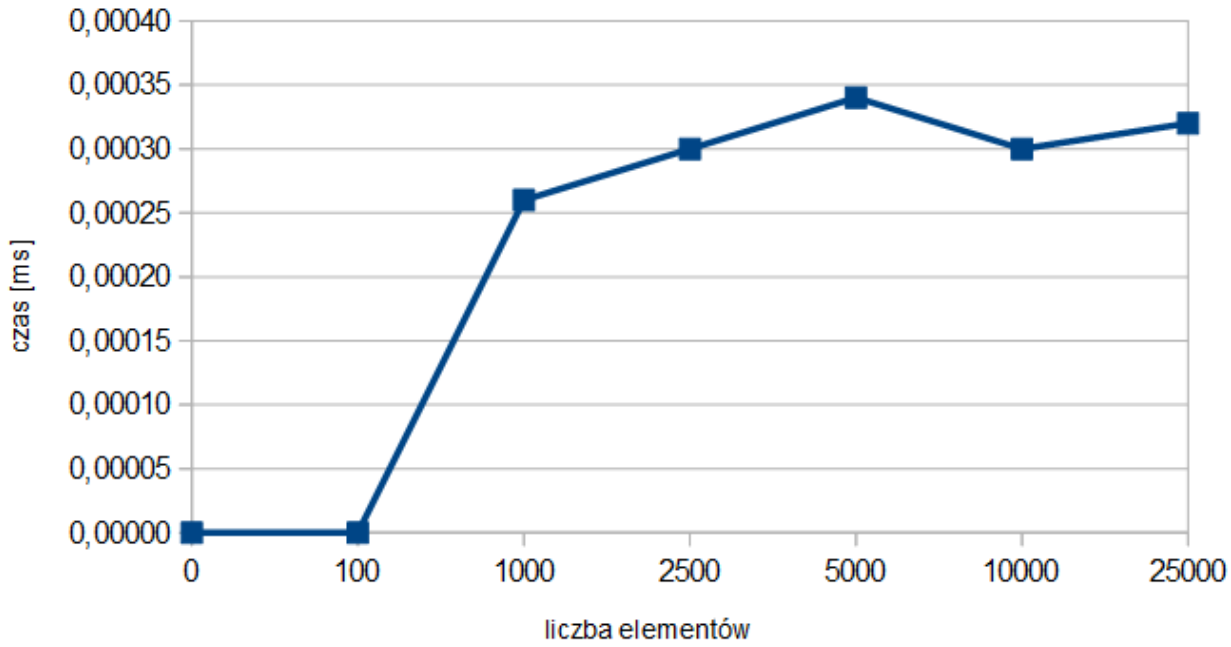


Wykres 1. Czas dodawania na początek tablicy

6.1.2 Dodawanie na koniec tablicy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-5}ms]
1	100	0,00000
2	1000	0,00026
3	2500	0,00030
4	5000	0,00034
5	10000	0,00030
6	25000	0,00032

Tabela 6. Dodawanie na koniec tablicy

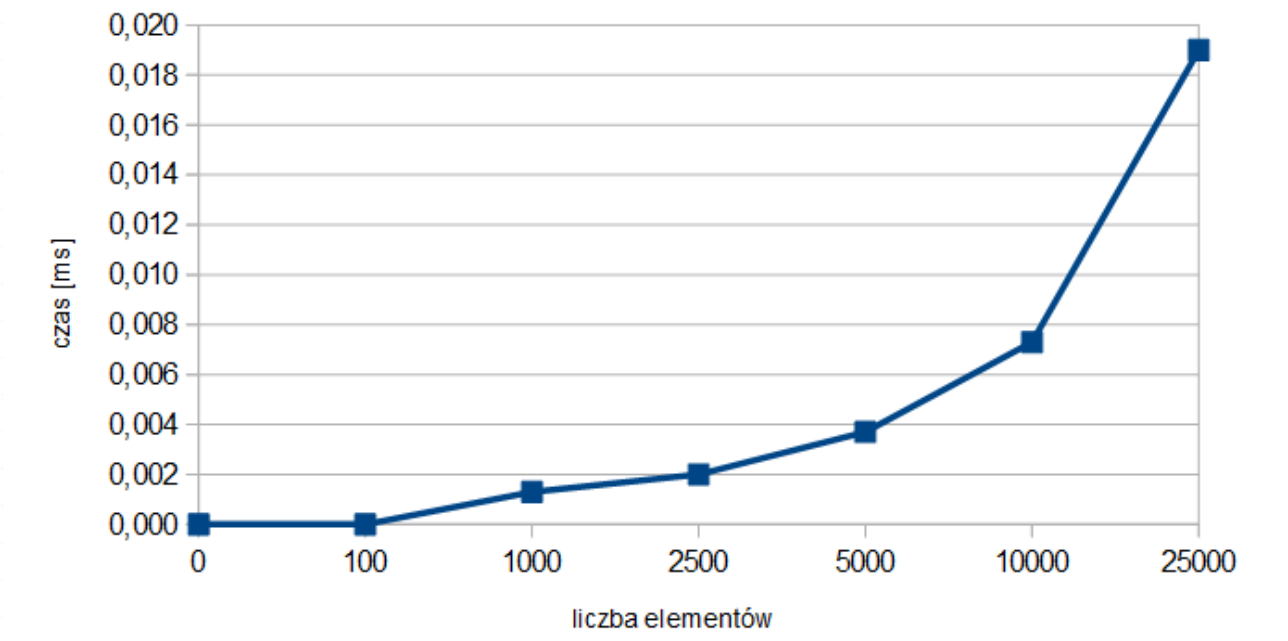


Wykres 2. Dodawanie na koniec tablicy

6.1.3 Dodawanie na losową pozycję w tablicy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-3}ms]
1	100	0,000
2	1000	0,002
3	2500	0,002
4	5000	0,004
5	10000	0,008
6	25000	0,019

Tabela 7. Dodawanie w losowe miejsce w tablicy

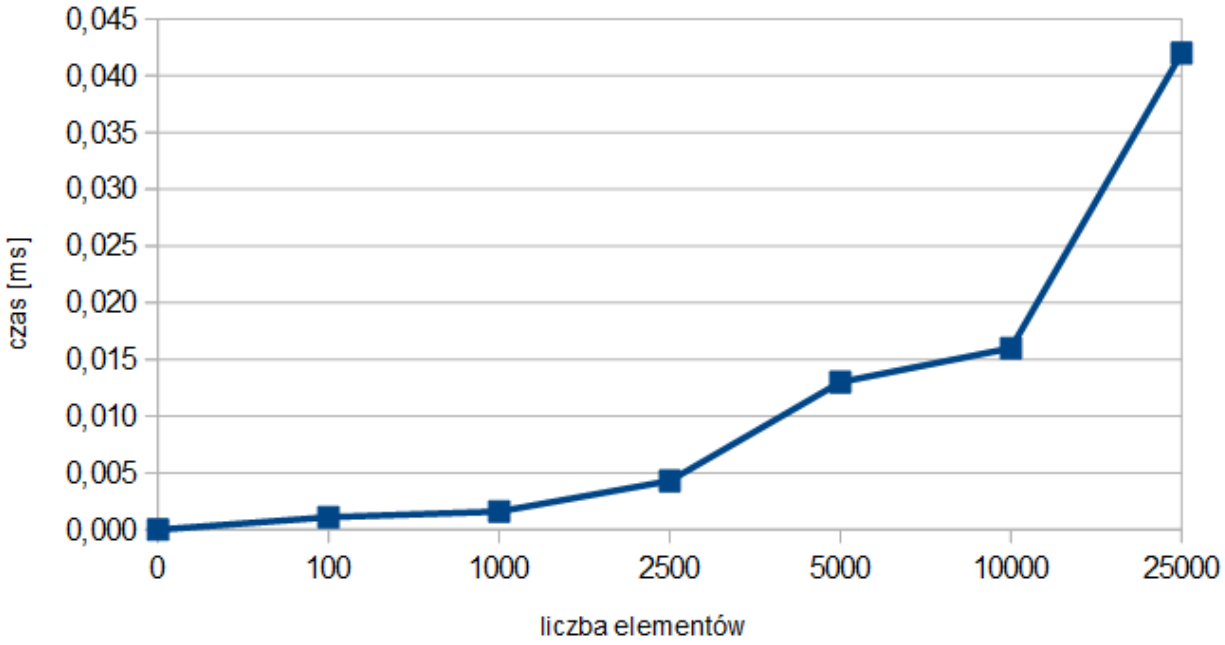


Wykres 3. Dodawanie na losową pozycję

6.1.4 Usuwanie z początku tablicy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-3}ms]
1	100	0,002
2	1000	0,002
3	2500	0,005
4	5000	0,013
5	10000	0,016
6	25000	0,042

Tabela 8. Usuwanie z początku tablicy

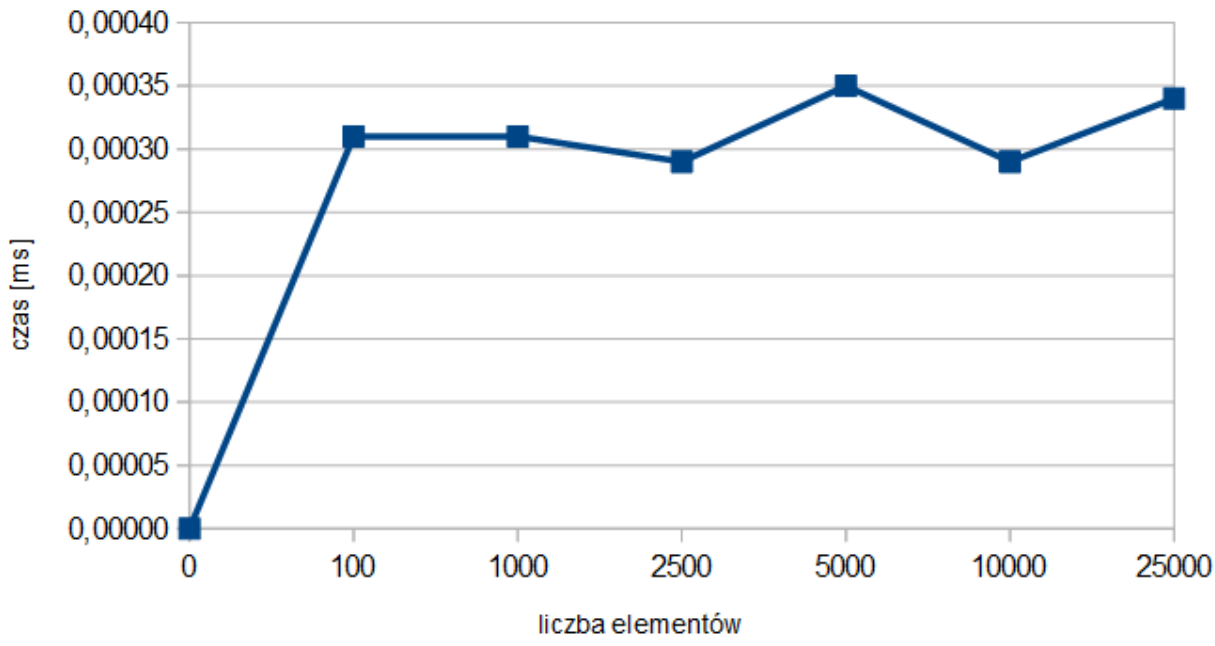


Wykres 4. Usuwanie z początku tablicy

6.1.5 Usuwanie z końca tablicy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-5}ms]
1	100	0,00031
2	1000	0,00031
3	2500	0,00029
4	5000	0,00035
5	10000	0,00029
6	25000	0,00034

Tabela 9. Usuwanie z końca tablicy

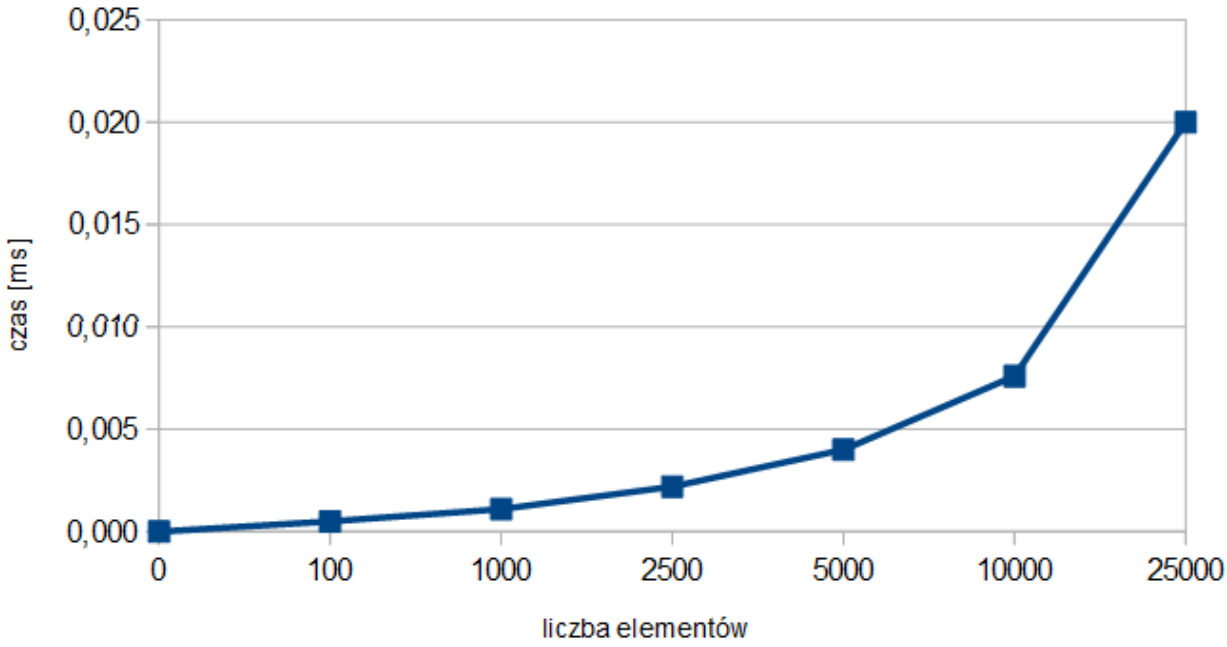


Wykres 5. Usuwanie z końca tablicy

6.1.6 Usuwanie losowego miejsca w tablicy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-3}ms]
1	100	0,001
2	1000	0,002
3	2500	0,003
4	5000	0,004
5	10000	0,008
6	25000	0,020

Tabela 10. Usuwanie losowego miejsca w tablicy

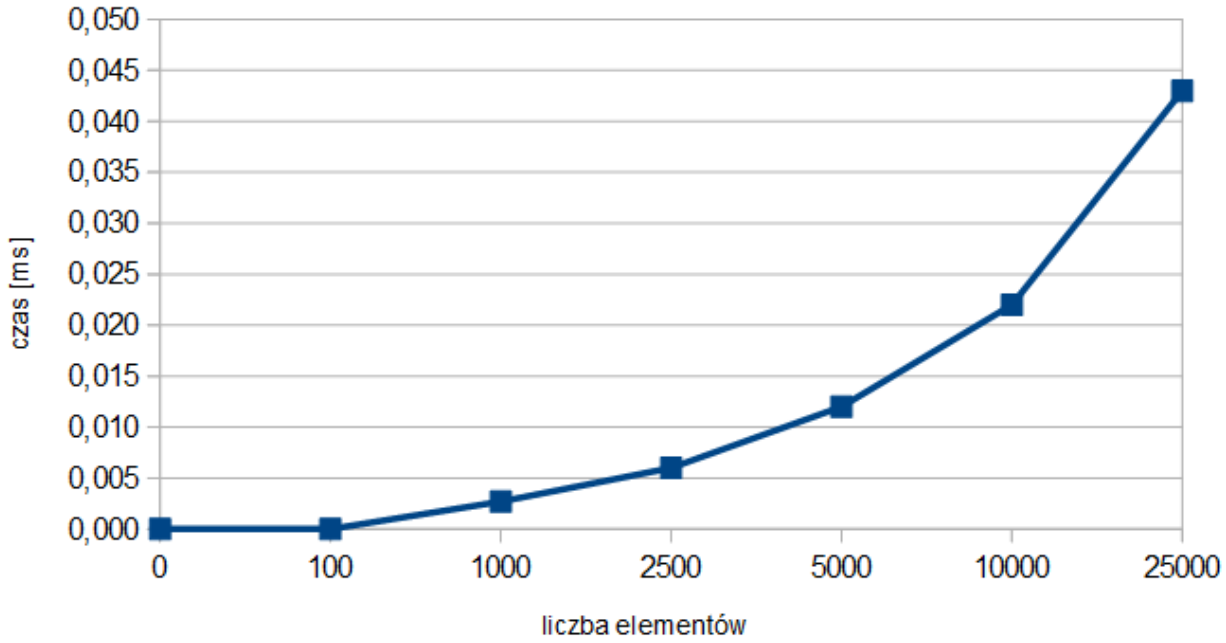


Wykres 6. Usuwanie losowego miejsca w tablicy

6.1.7 Wyszukiwanie w tablicy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-3}ms]
1	100	0,000
2	1000	0,003
3	2500	0,006
4	5000	0,012
5	10000	0,022
6	25000	0,043

Tabela 11. Wyszukiwanie w tablicy



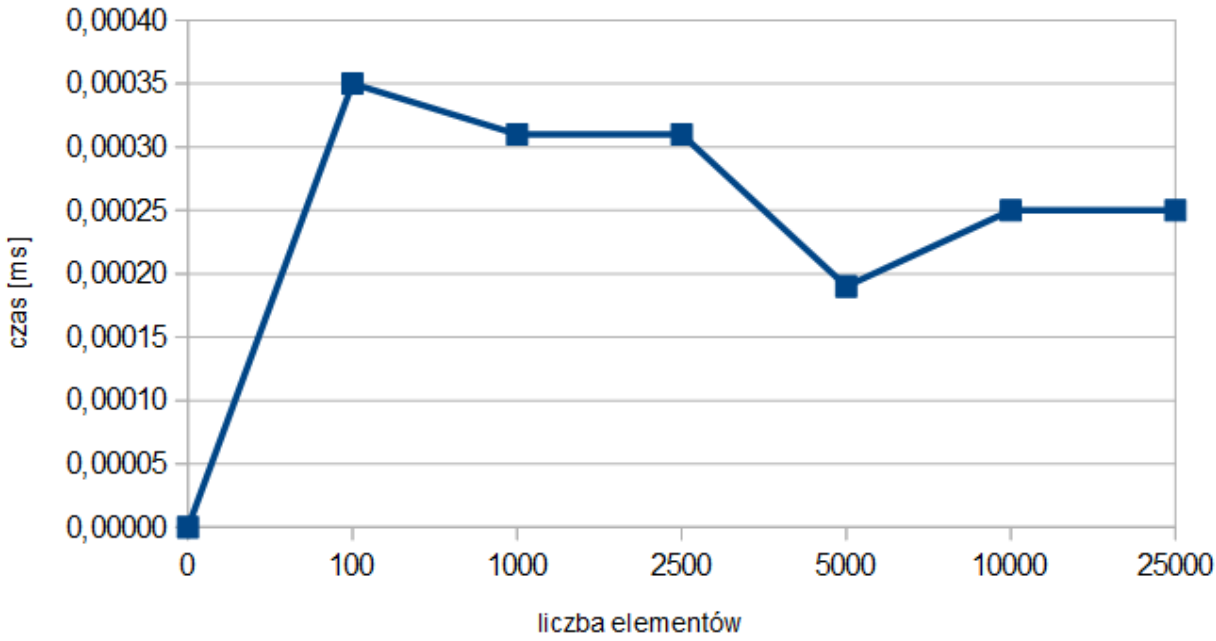
Wykres 7. Wyszukiwanie w tablicy

6.2 Lista

6.2.1 Dodawanie na początek listy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-5}ms]
1	100	0,00035
2	1000	0,00031
3	2500	0,00031
4	5000	0,00019
5	10000	0,00025
6	25000	0,00025

Tabela 12. Dodawanie na początek listy

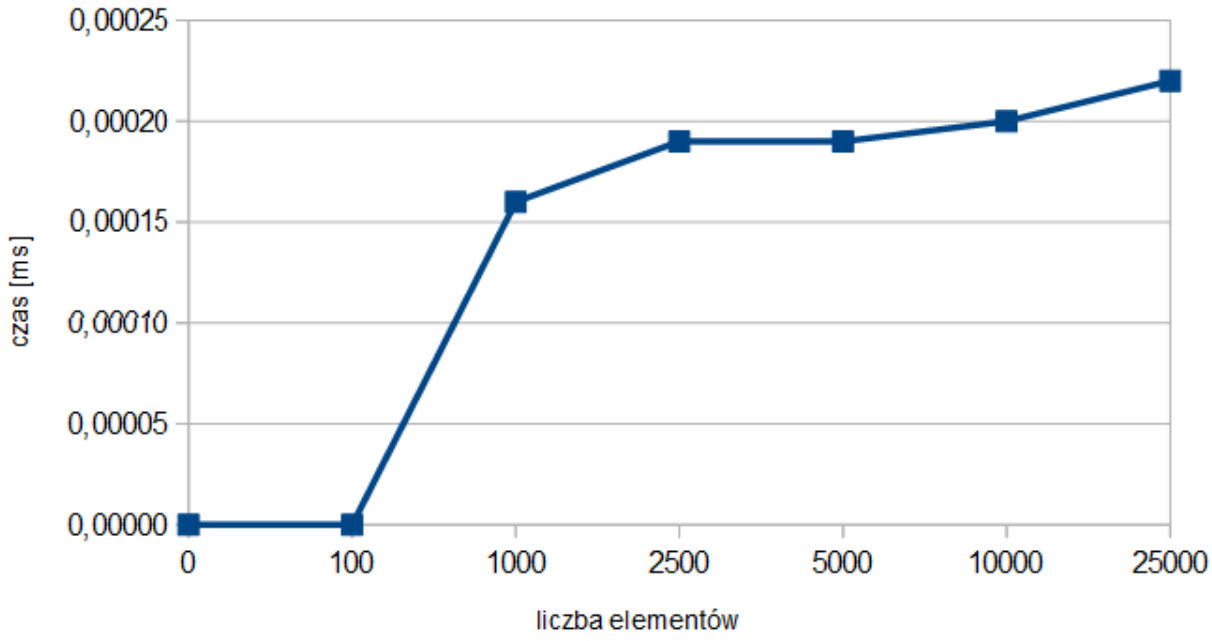


Wykres 8. Dodawanie na początek listy

6.2.2 Dodawanie na koniec listy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-5}ms]
1	100	0,00000
2	1000	0,00016
3	2500	0,00019
4	5000	0,00019
5	10000	0,00020
6	25000	0,00022

Tabela 13. Dodawanie na koniec listy

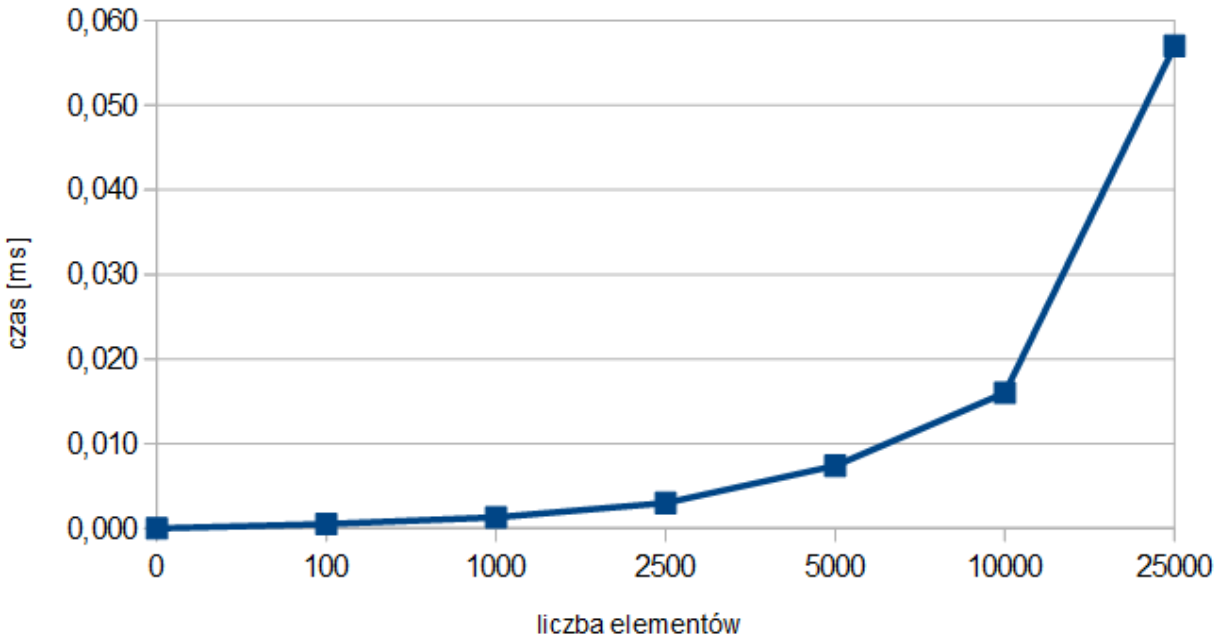


Wykres 9. Dodawanie na koniec listy

6.2.3 Dodawanie w losowe miejsce w listy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-3}ms]
1	100	0,001
2	1000	0,002
3	2500	0,003
4	5000	0,008
5	10000	0,016
6	25000	0,057

Tabela 14. Dodawanie w losowe miejsce w listy

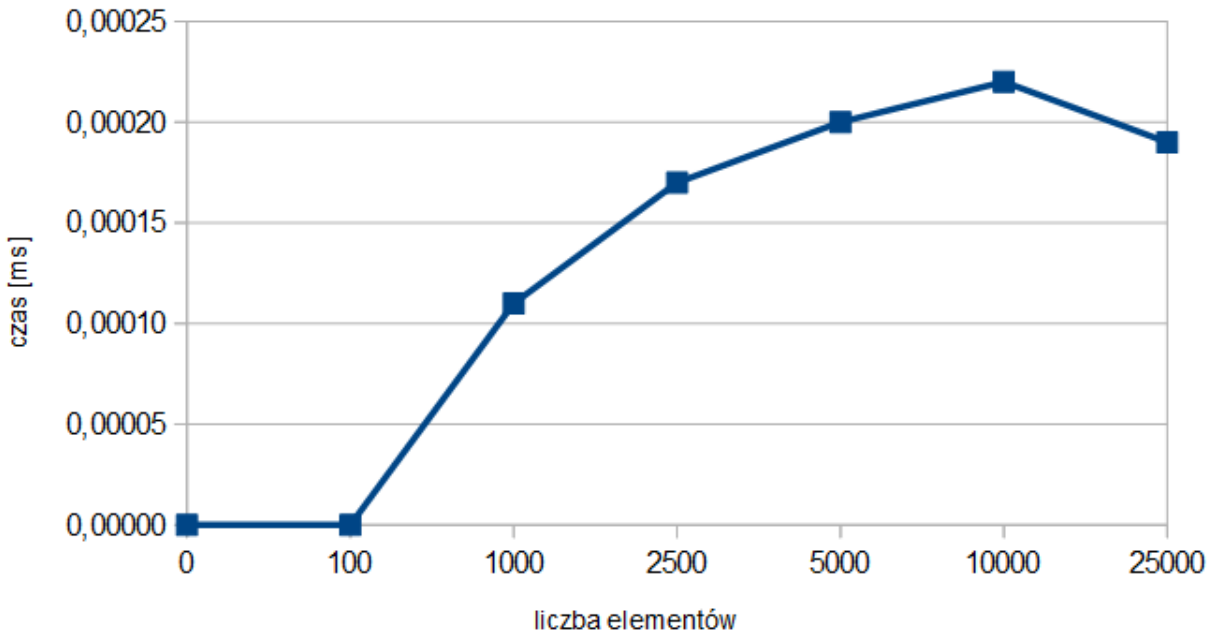


Wykres 10. Dodawanie w losowe miejsce w listy

6.2.4 Usuwanie z początku listy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-5}ms]
1	100	0,00000
2	1000	0,00011
3	2500	0,00017
4	5000	0,00020
5	10000	0,00022
6	25000	0,00019

Tabela 15. Usuwanie z początku listy

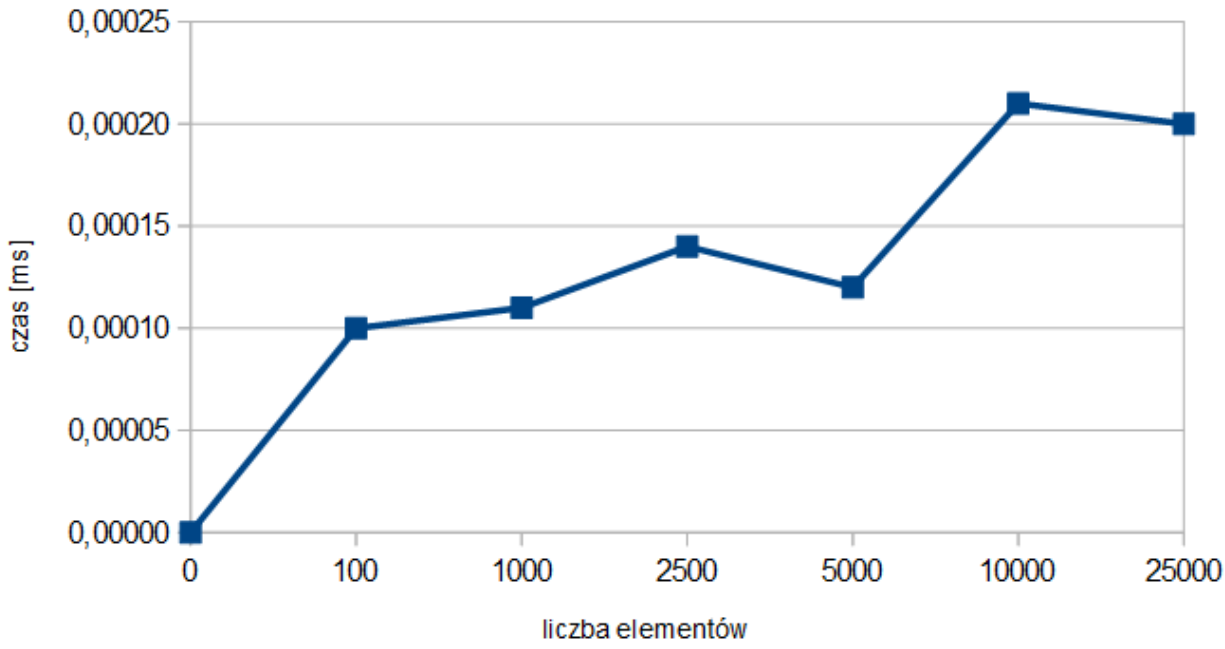


Wykres 11. Usuwanie z początku listy

6.2.5 Usuwanie z końca listy

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-5}ms]
1	100	0,00010
2	1000	0,00011
3	2500	0,00014
4	5000	0,00012
5	10000	0,00021
6	25000	0,00020

Tabela 16. Usuwanie z końca listy

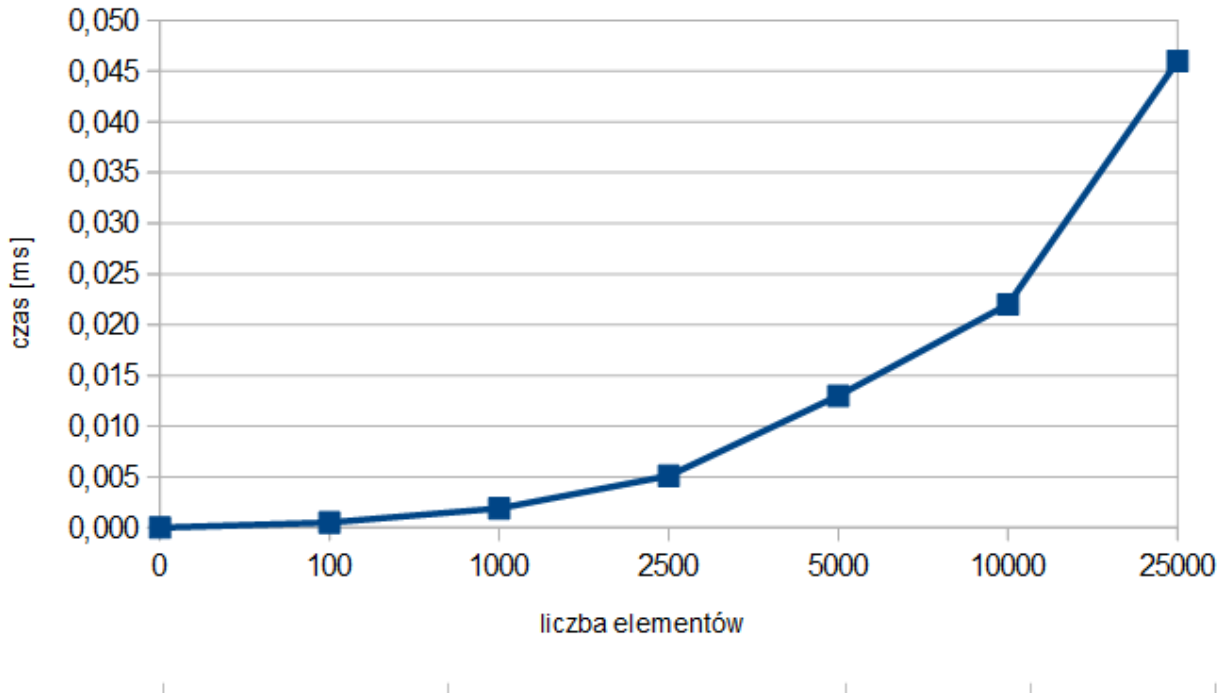


Wykres 12. Usuwanie z końca listy

6.2.6 Usuwanie losowego miejsca w liście

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-3}ms]
1	100	0,001
2	1000	0,002
3	2500	0,006
4	5000	0,013
5	10000	0,022
6	25000	0,046

Tabela 17. Usuwanie losowego miejsca w liście

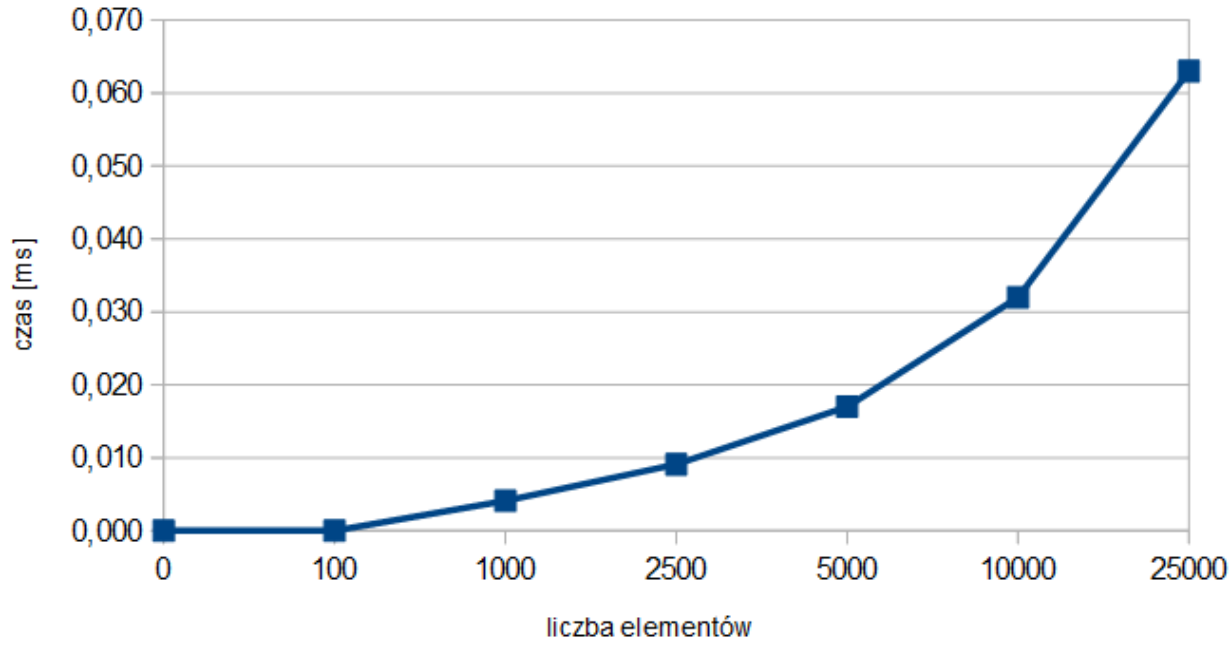


Wykres 13. Usuwanie losowego miejsca w liście

6.2.7 Wyszukiwanie w liście

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-3}ms]
1	100	0,000
2	1000	0,005
3	2500	0,010
4	5000	0,017
5	10000	0,032
6	25000	0,063

Tabela 18. Wyszukiwanie w liście



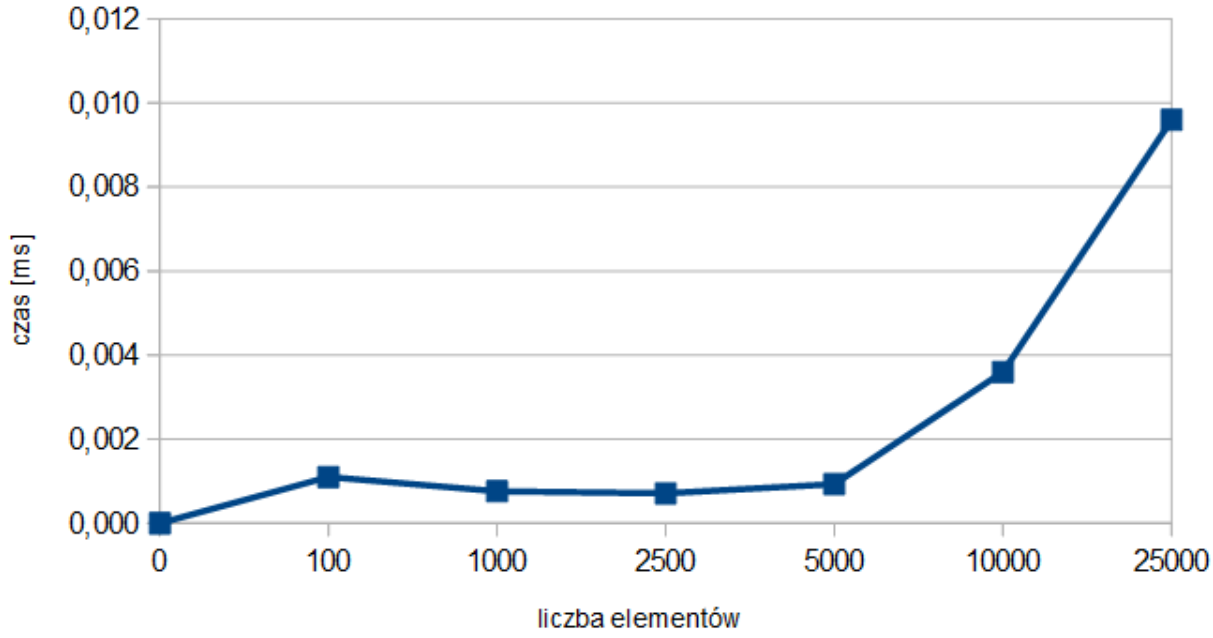
Wykres 14. Wyszukiwanie w liście

6.3 Kopiec binarny

6.3.1 Dodawanie klucza

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-4}ms]
1	100	0,0011
2	1000	0,0008
3	2500	0,0008
4	5000	0,0010
5	10000	0,0036
6	25000	0,0096

Tabela 19. Dodawanie klucza

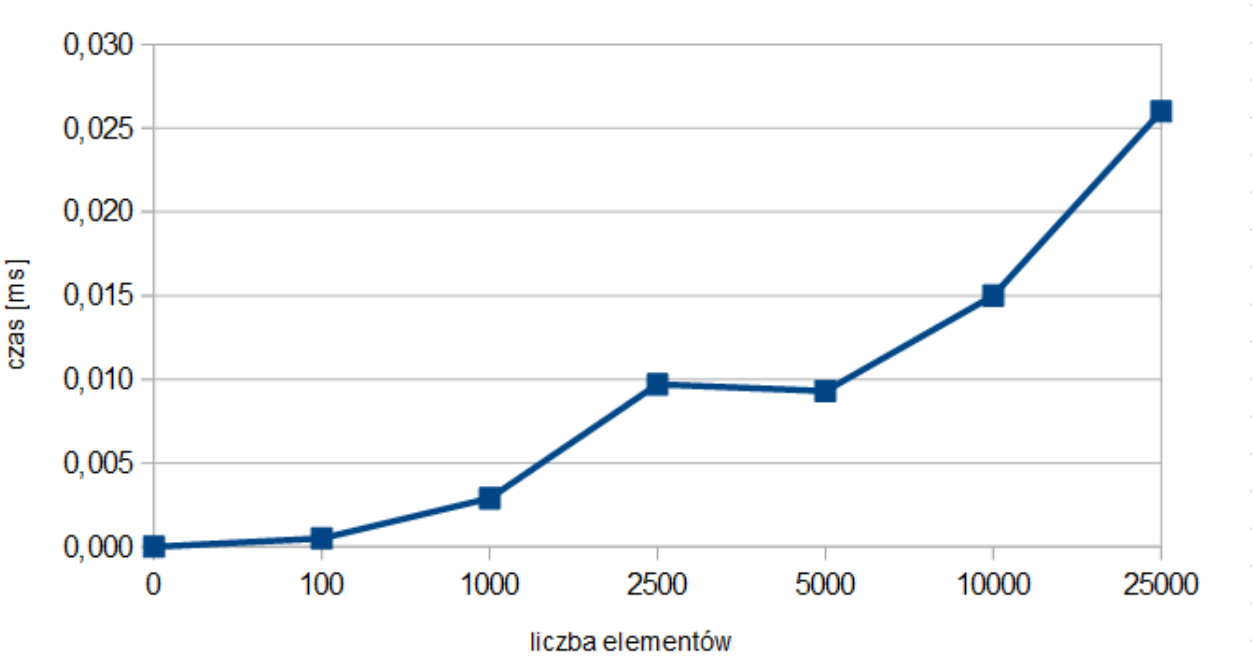


Wykres 15. Dodawanie klucza

6.3.2 Usuwanie klucza

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-3}ms]
1	100	0,001
2	1000	0,003
3	2500	0,010
4	5000	0,094
5	10000	0,015
6	25000	0,026

Tabela 20. Usuwanie klucza

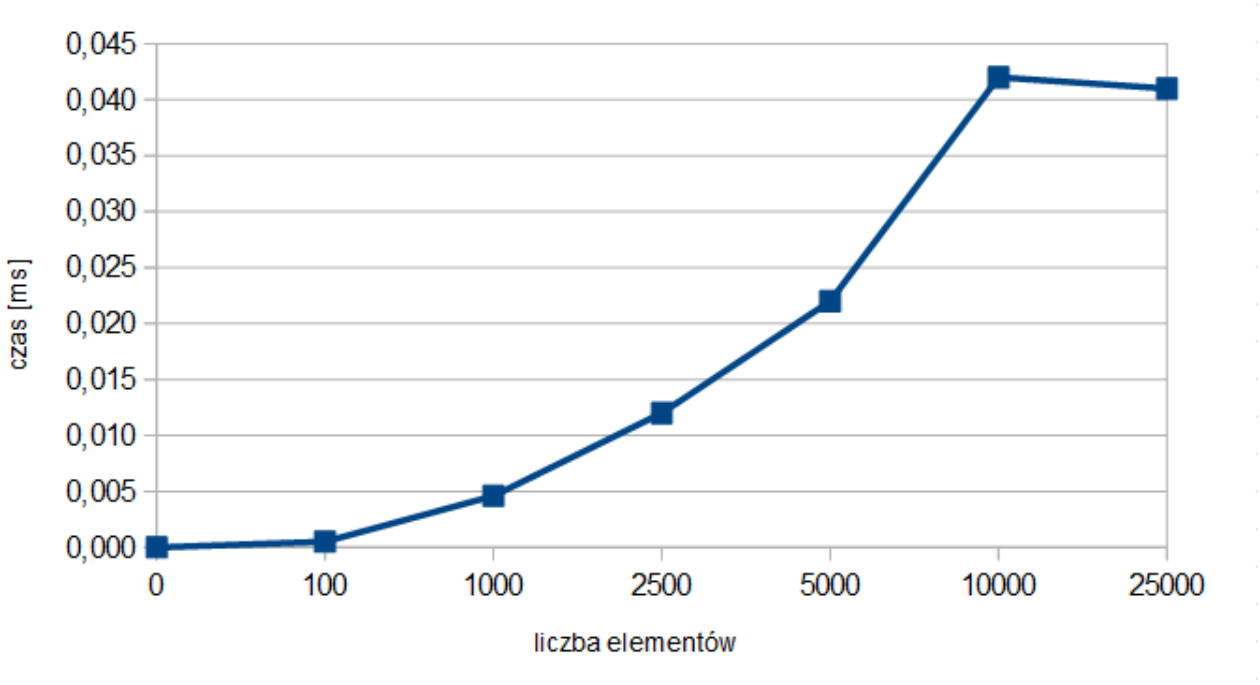


Wykres 16. Usuwanie klucza

6.3.3 Wyszukiwanie klucza

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-3}ms]
1	100	0,001
2	1000	0,005
3	2500	0,012
4	5000	0,022
5	10000	0,042
6	25000	0,041

Tabela 21. Wyszukiwanie klucza



Wykres 17. Wyszukiwanie klucza

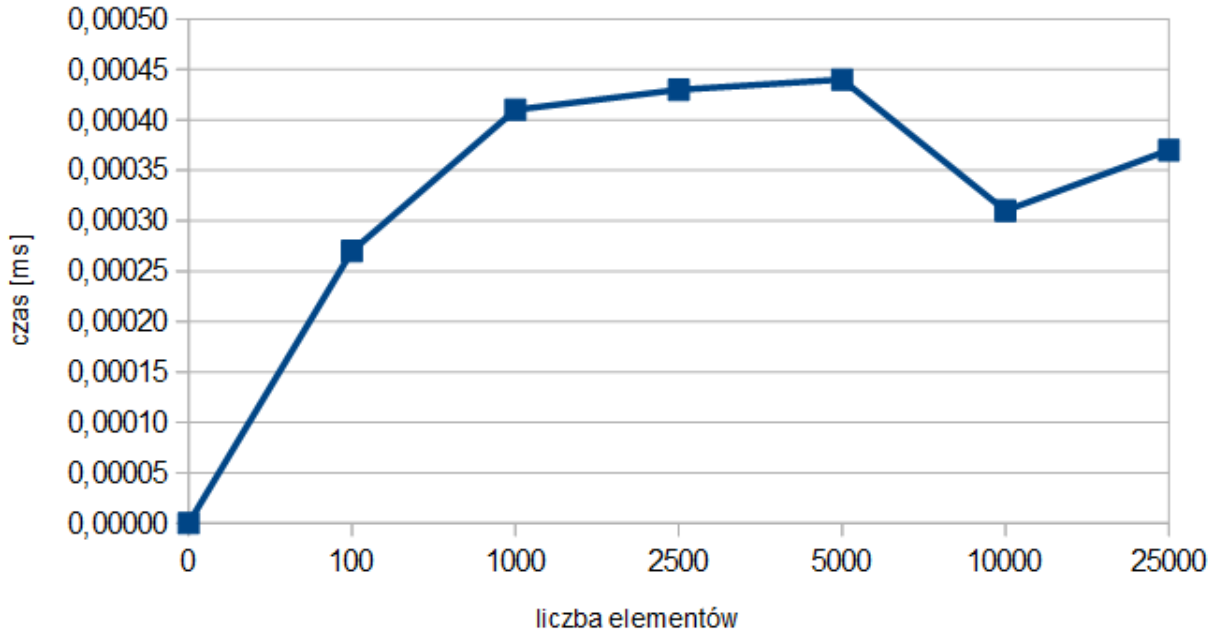
6.4 Drzewo BST

Zgodnie z założeniami projektowymi podczas dodawania lub usuwania klucza algorytm DSW – równoważenia drzewa także jest obejmowany przez pomiary czasów.

6.4.1 Dodawanie klucza

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-5}ms]
1	100	0,00027
2	1000	0,00041
3	2500	0,00043
4	5000	0,00044
5	10000	0,00031
6	25000	0,00037

Tabela 22. Dodawanie klucza

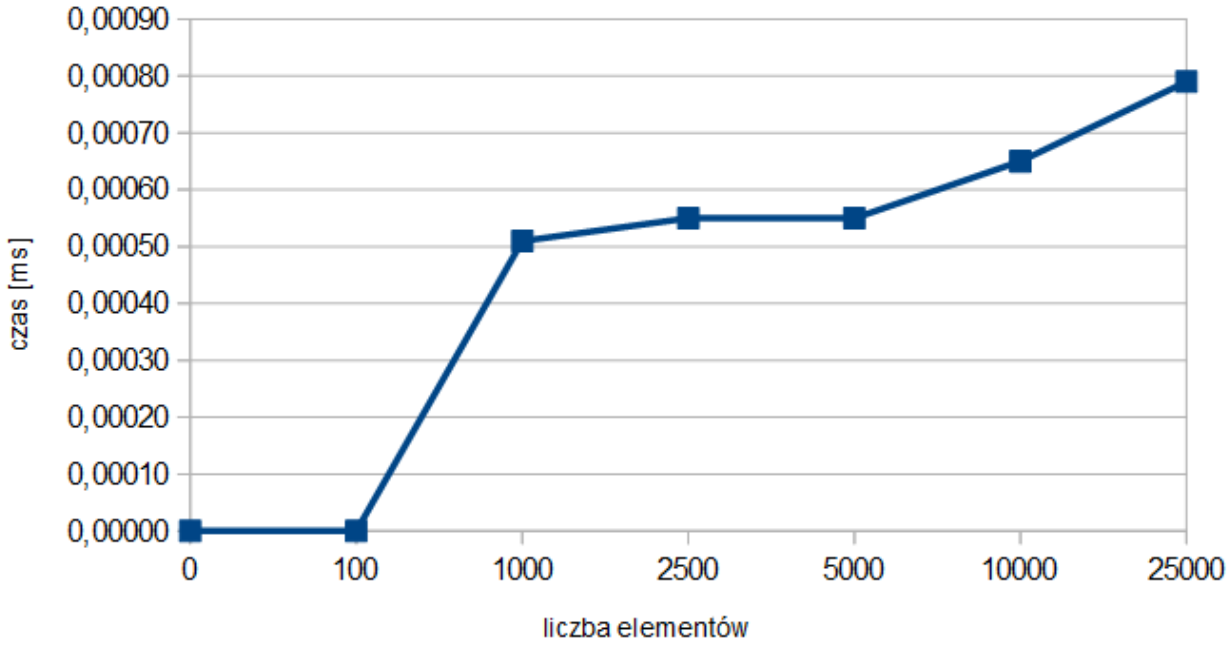


Wykres 18. Dodawanie klucza

6.4.2 Usuwanie klucza

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-5}ms]
1	100	0,00000
2	1000	0,00051
3	2500	0,00055
4	5000	0,00055
5	10000	0,00065
6	25000	0,00079

Tabela 23. Usuwanie klucza

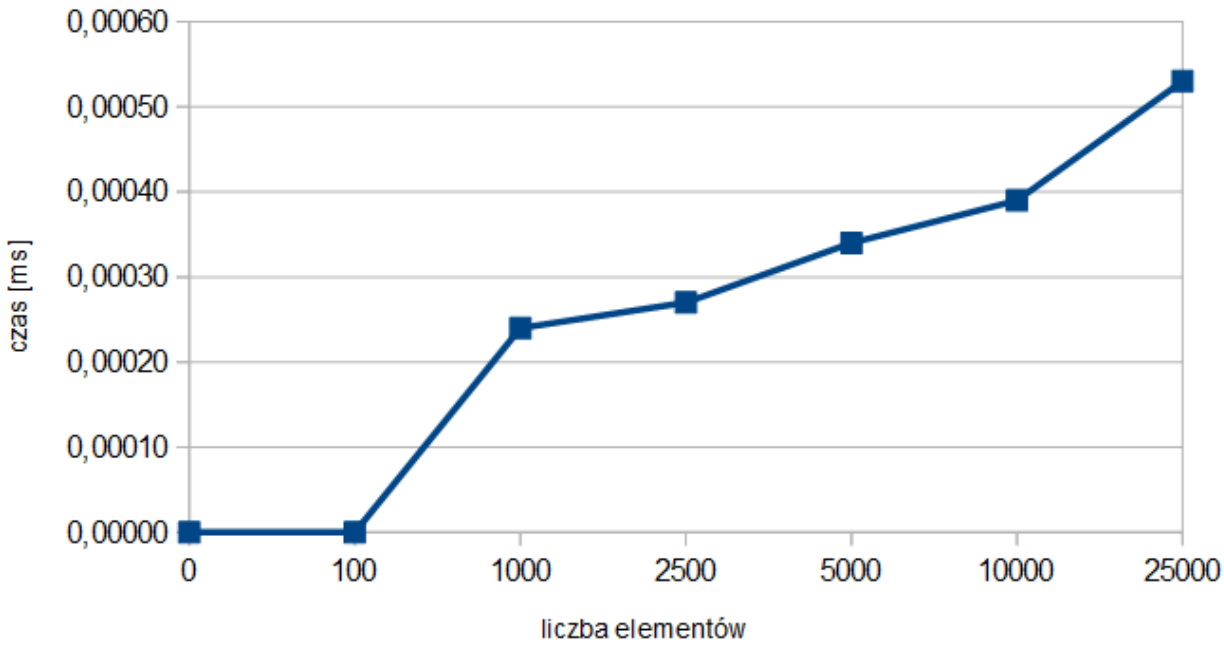


Wykres 19. Usuwanie klucza

6.4.3 Wyszukiwanie klucza

l.p.	l.el.	Czas średni 20 pomiarów[ms][+-10^{-5}ms]
1	100	0,00000
2	1000	0,00024
3	2500	0,00027
4	5000	0,00034
5	10000	0,00039
6	25000	0,00053

Tabela 24. Wyszukiwanie klucza



Wykres 20.Wyszukiwanie klucza

7. Wnioski

Na podstawie wykonanych pomiarów można stwierdzić, że wyniki w dużej mierze pokrywają się z teoretycznymi założeniami. W tablicy najszybciej wykonane zostały operacje: usuwanie i dodawanie na końcu - dla 25000 elementów uzyskano średni czas operacji rzędu o dokładności 10^{-4}ms. Dużo dłużej natomiast (aż o dwa rzędy wielkości więcej) wypadły operacje dodawania i usuwania z początku tablicy. Prawdopodobnie wynika to z faktu, że w pierwszym przypadku wystarczy wstawić element na koniec uprzednio zwiększając rozmiar tablicy, kiedy w drugim należy dodatkowo przesunąć każdy element. Jeśli chodzi o wyszukiwanie, w przypadku tablicy jest to powolny proces ponieważ każdy element musi zostać sprawdzony po kolei. Dla większej ilości elementów czas średni owej operacji znacznie się wydłuża.

Zaletą listy dwukierunkowej jest fakt, że wystarczy jedna komórka w pamięci aby przechować całą listę, głowa wskazuje na kolejne elementy listy, a ostatni element jest ogonem. Wystarczy więc znać adres w pamięci głowy i za sprawą kolejnych wskaźników można mieć dostęp do każdego elementu listy. Operacje takie jak: dodawanie i usuwanie na początku i na końcu listy dały pomiary tego samego rzędu 10^{-4}ms, nieznacznie różniące się od siebie. W przypadku dodawania lub usuwania elementu ze środka listy czas ten jest wydłużony, ponieważ należy najpierw dostać się element po elemencie do szukanego indeksu (rząd 10^{-2}ms). W przypadku wyszukiwania lista wypadła bardzo podobnie do tablicy - ten sam rząd. Wynika to z faktu, że w liście zarówno jak i w tablicy wyszukiwanie elementu jest element po elemencie.

Struktura kopca binarnego jest połączeniem drzewiastej strukury drzewa binarnego oraz tablicy. W badanym przypadku należało zaimplementować strukturę kopca binarnego maksymalnego. W przypadku dodawania i usuwania klucza czas średni pomiarów jest tego samego rzędu i jest w granicy 10^{-3}ms. Operacje dodawania i usuwania klucza w kopcu jest porównywalna lub wolniejsza niż w tablicy i liście. Natomiast jeśli chodzi o wyszukiwanie, kopiec dla dużej ilości elementów zajmujących większą ilość bajtów w pamięci jest lepszym rozwiązaniem czasowym, co widać na wykresie 17 - czas średni wykonywaej operacji wyszukiwania stabilizuje się.

Drzewo BST podobnie jak lista nie potrzebuje tablicy do implementacji, wystarczy wskaźnik na korzeń. Z założenia w teorii drzewo BST powinno wypaść czasowo najlepiej ze wszystkich zaimplementowanych struktur. Widać to na wykresach: 18,19,20. Dodawanie oraz usuwanie wskazanego klucza są tego samego rzędu (10^{-4}ms). **Wyszukiwanie - jest to struktura, która ze wszystkich badanych struktur ma najszybszy dostęp do elementu rzędu (10^{-4}ms)**. Jej możliwości czasowe są dużo większe niż możliwości pozostałych struktur, czas średni tej operacji jest około 100razy lepszy niż w przypadku pozostałych struktur (rząd 10^{-4}ms).

8. Proponowana ocena

4.5 - Eksperymenty na tablicy, liście i kopcu binarnym i drzewie BST, z równoważeniem drzewa algorytmem DSW

9. Literatura

"Wprowadzenie do algorytmów" - Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., Clifford Stein