STRUKTURY DANYCH I ZŁOŻONOŚĆ OBLICZENIOWA

AUTOR:

PATRYCJA LANGKAFEL, NR 252744

ZADANIE PROJEKTOWE 1

TERMIN: PONIEDZIAŁEK 9.15, PARZYSTY

1. WSTĘP

Celem wykonanego zadania projektowego było zbadanie efektywności podstawowych operacji takich jak wstawianie, wyszukiwanie oraz usuwanie. Struktury jakie zostały użyte to : tablica dynamiczna, lista dwukierunkowa i kopiec binarny. Język programowania, jaki został użyty do przeprowadzenia zadania i testów to C++. Badaną wielkością było tempo wzrostu konkretnych operacji (zapisywane jako O), aby opisać złożoność obliczeniową konkretnych operacji.

1.1 TABLICA DYNAMICZNA

Tablica jest strukturą danych, gdzie każdy element jest identyfikowalny przez konkretny indeks tablicy. Aby uzyskać dostęp do jakiegoś elementu potrzebujemy wskaźnik na początek tablicy oraz jego indeks.

- **Wstawianie**

  - na początek tablicy

Aby wstawić element na początek tablicy tworzymy nową tablicę o rozmiarze o 1 większym, w której wszystkie elementy zostaną przesunięte o jeden do przodu. Złożoność O(n).

  - na koniec tablicy

Przy tej operacji nie musimy przesuwać elementów tablicy (krótszy czas wykonania). Tworzymy nową tablicę o indeksie o jeden większym i przepisujemy po kolei elementy, a na ostatnim zapisujemy nowy element. Złożoność O(n).

  - na określone miejsce tablicy

Sytuacja podobna do wstawiania na początek, ale jest mniejsza liczba elementów, które trzeba przesunąć. Złożoność O(n).

- **Usuwanie**

  - na początek tablicy

Usuwanie jest operacją podobną do wstawiania, z tą różnicą, że przy usuwaniu z początku tablicy, tworzymy nową o rozmiarze o jeden mniejszym i przesuwamy wszystkie elementy o w dół i pomijamy wartość z indeksem 0. Złożoność O(n).

  - na środek tablicy

Sytuacja podobna do usuwania pierwszego elementu, ale jest mniejsza liczba elementów, które trzeba przesunąć. Złożoność O(n).

  - na koniec tablicy

W tym przypadku po prostu przepisujemy starą tablicą do nowej bez ostatniego elementu. Złożoność O(n).

- **Wyszukiwanie**

  - po indeksie

Każdy kolejny indeks elementu jest porównywany z szukanym indeksem do momentu jego znalezienia i zwracana jest wartość z podanego indeksu. W najgorszym przypadku trzeba przeszukać całą tablicę. Złożoność O(n).

  - po wartościach

Każdą kolejną wartość elementu porównujemy z szukanym elementem do momentu jego znalezienia i zwracany jest indeks. W najgorszym przypadku trzeba przeszukać całą tablicę. Złożoność O(n).

1.2 LISTA DWUKIERUNKOWA

Lista dwukierunkowa jest strukturą danych składającą się z sekwencyjnie połączonych ze sobą elementów. Każdy element zawiera przechowywaną wartość oraz dwa wskaźniki na poprzedni i następny element listy.

- **Wstawianie**

  - na początek listy

W przypadku wstawiania na początek listy, nowy element staje się pierwszym elementem na liście, a wcześniejszy pierwszy element zostaje elementem kolejnym. Złożoność O(1).

  - na koniec listy

W przypadku wstawiania na koniec listy, nowy element staje się ostatnim elementem na liście, a wcześniejszy ostatni element zostaje elementem przedostatnim. Złożoność O(1).

  - na określone miejsce listy

W przypadku kiedy chcemy wstawić nowy element w określone miejsce listy, musimy najpierw odnaleźć element który obecnie się tam znajduje. Następnie tworzymy nowe połączenia między nowym elementem, a jego następnikiem i poprzednikiem. Złożoność O(n).

- **Usuwanie**
  - na początek listy

W przypadku usuwania elementu z początku listy, następny element staje się pierwszym. Złożoność O(1).

- na koniec listy

W przypadku usuwania elementu z końca listy, przedostatni element staje się ostatnim. Złożoność O(1).

- na określone miejsce listy

Tak samo jak w przypadku wstawiania elementu w określone miejsce listy tutaj również musimy odnaleźć konkretny element, co wraz z wzrostem elementów staje się coraz to bardziej czasochłonne. Po jego odnalezieniu, tworzymy nowe wiązania między jego następnikiem i poprzednikiem i usuwamy określony element. Złożoność O(n).

- **Wyszukiwanie**

- po indeksie

Każdy kolejny indeks listy jest porównywany z szukanym indeksem do momentu jego znalezienia, czyli w najgorszym wypadku trzeba przeszukać całą listę. Złożoność O(n).

- po wartościach

Każdy kolejna wartość z listy jest porównywany z szukana wartość do momentu jego znalezienia, czyli w najgorszym wypadku trzeba przeszukać całą listę. Złożoność O(n).

1.3 KOPIEC BINARNY

Kopiec jest strukturą danych przybierającą formę drzewiastą, kopiec maksymalny ma w korzeniu wartość największą. Kopiec jest jednym ze sposobów implementacji kolejki priorytetowej. Przy implementacji kopca wykorzystałam tablicę.

- **Wstawianie**

Wartość wstawiana jest zawsze na miejsce za ostatnim elementem kopca, a następnie krok po kroku nowa wartość jest przesuwana w górę drzewa, aby naprawić kopiec. W najgorszym przypadku złożoność jest równa O(log(n)), a w najlepszym - O(1).

- **Usuwanie**

Usuwanie elementu jest wykonywane dla pierwszego elementu, który jest zamieniamy miejscami z ostatnim elementem kopca, a potem jest naprawiany w dół. Złożoność O(log(n)).

- **Wyszukiwania**

- po indeksie

Każdy kolejny indeks kopca jest porównywany z szukanym indeksem do momentu jego znalezienia, czyli w najgorszym wypadku trzeba przeszukać cały kopiec. Złożoność O(n).

- po wartościach

Każdy kolejna wartość z kopca jest porównywany z szukana wartość do momentu jego znalezienia, czyli w najgorszym wypadku trzeba przeszukać cały kopiec. Złożoność O(n).

2. PRZEBIEG EKSPERYMENTU

W programie zaimplementowałam przyjęte struktury, a przyjęte zakresy liczb to: 10000, 20000, 30000, 40000, 50000. Do pomiaru czasu wykorzystałam QueryPerformanceCounter(). Struktury były generowane w następujący sposób:

-Losowano określoną liczbę wartości w programie funkcją rand().

-Całość dodawano do pliku, z którego były pobierane wartości do określonej struktury.

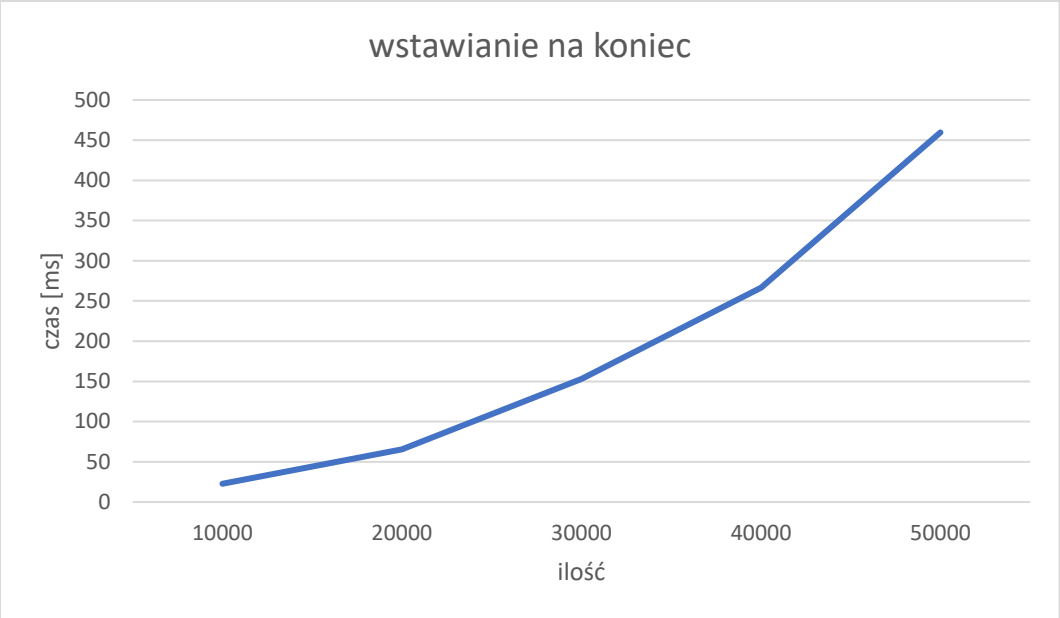
-Wykonywano funkcję, przez pobieranie wartości z pliku

-Na końcu obliczam czas dla określonej metody i zapisuję pomiary.

2.1 TABLICA DYNAMICZNA

- **Wstawianie**

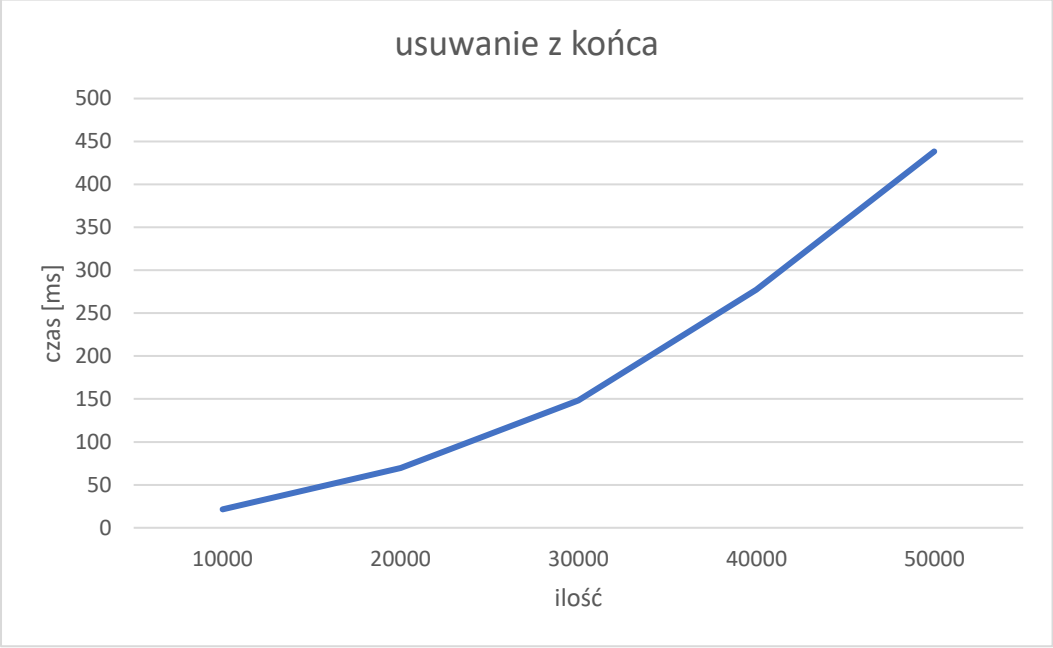
	add_front [ms]	add_back [ms]	add [ms]
10000	74,24	22,68	71,24
20000	65,93	65,38	288,09
30000	146,38	152,99	677,88
40000	261,52	266,48	1148,71
50000	450,84	2659,9	1774,16

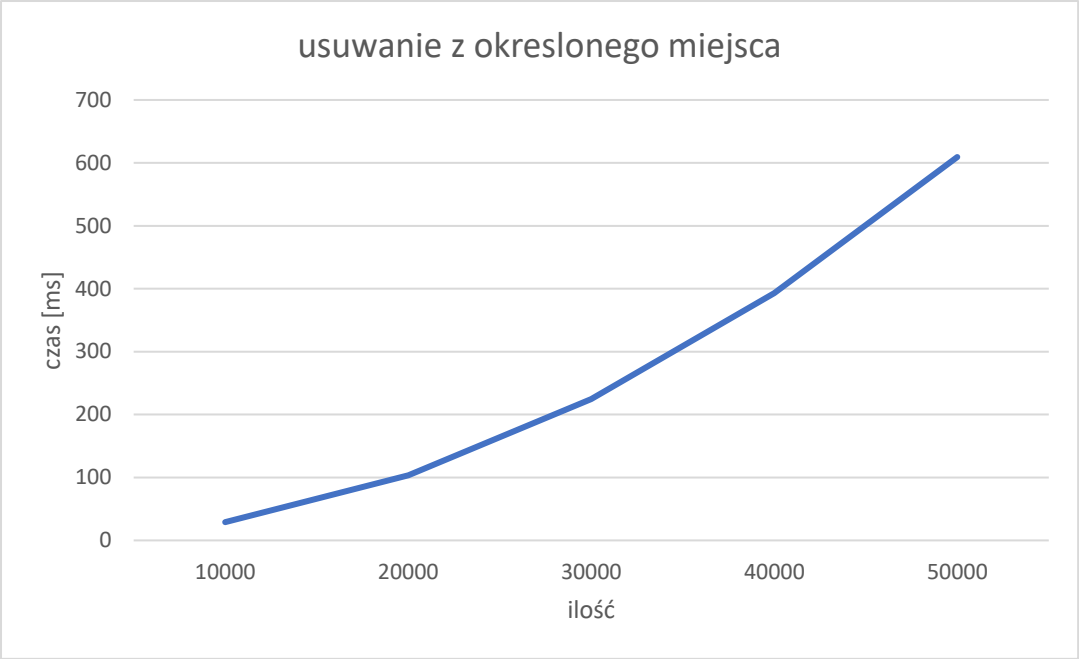


- **Usuwanie**

	delete_front [ms]	delete_back [ms]	remove [ms]
10000	34,77	21,48	29,04
20000	74,35	69,73	103,43
30000	147,9	148,3	224,8
40000	281,89	277,41	392,83
50000	449,95	438,33	609,26

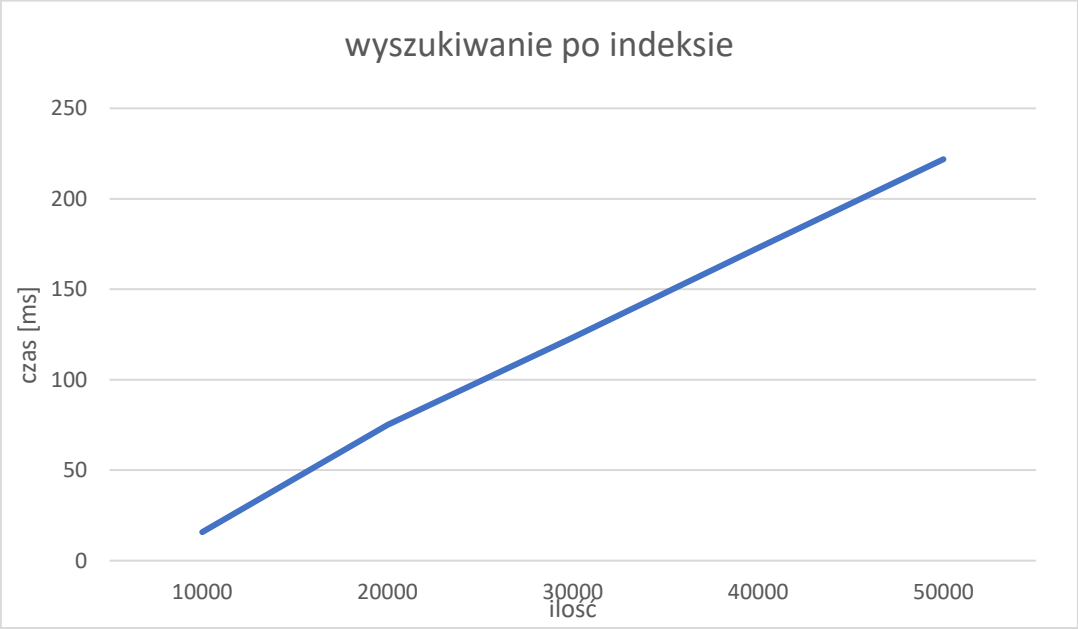

usuwanie z początku
500
450
400
350
300
250
200
150
100
50
0
czas [ms]
10000
20000
30000
40000
50000
ilość


usuwanie z końca
500
450
400
350
300
250
200
150
100
50
0
czas [ms]
10000
20000
30000
40000
50000
ilość



- **Wyszukiwanie**

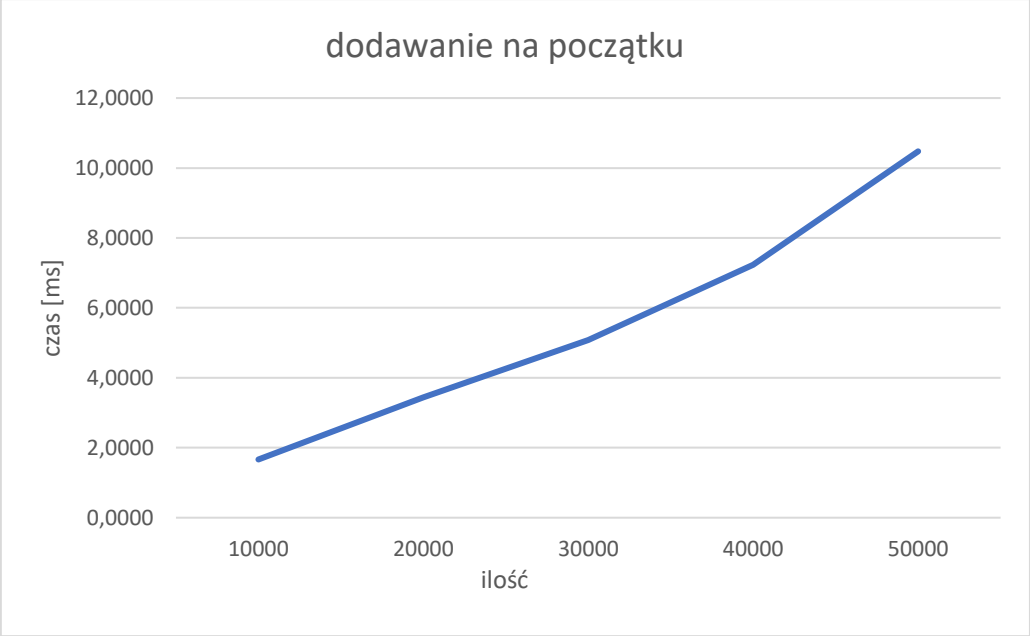
	find_elem [ms]	find_elem_value [ms]
10000	15,77	24,6
20000	75,07	96,49
30000	123,37	209,87
40000	172,82	391,75
50000	221,86	598,23

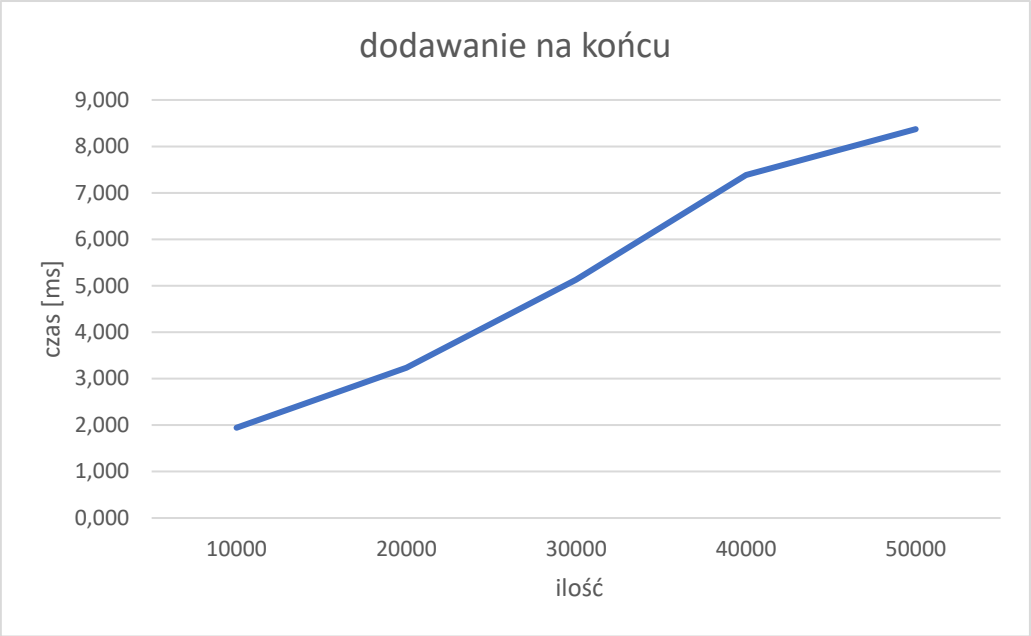


2.2 LISTA DWUKIERUNKOWA

- **Wstawianie**

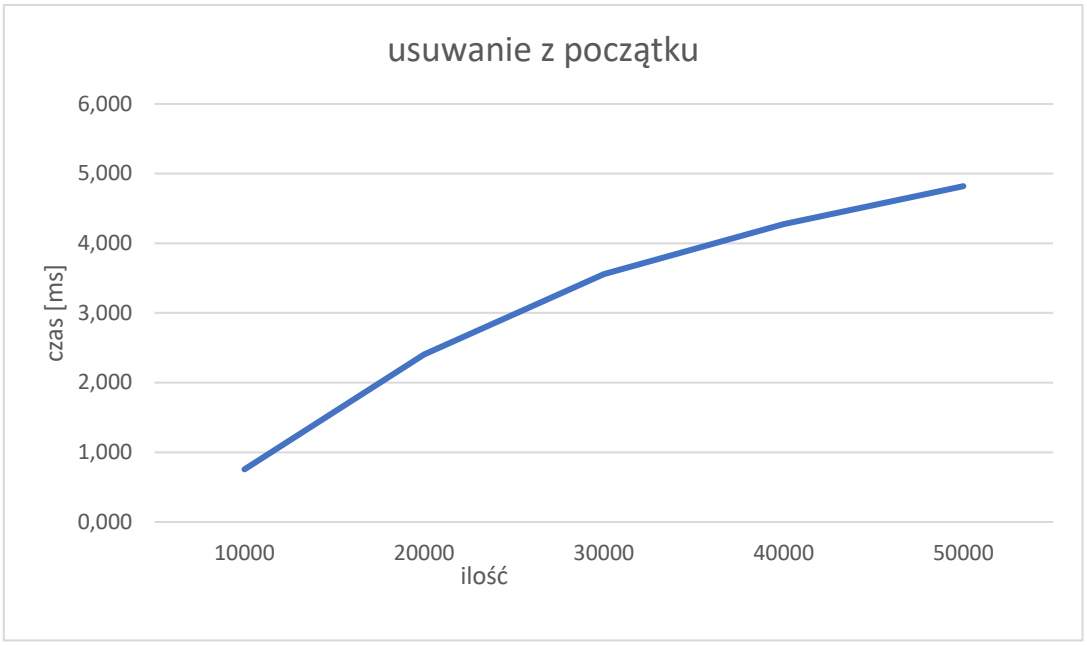
	add_front [ms]	add_back [ms]	add [ms]
10000	1,66	1,94	155,49
20000	3,44	3,24	561,78
30000	5,08	5,13	1650,92
40000	7,23	7,39	3042,4
50000	10,48	8,37	4502,95

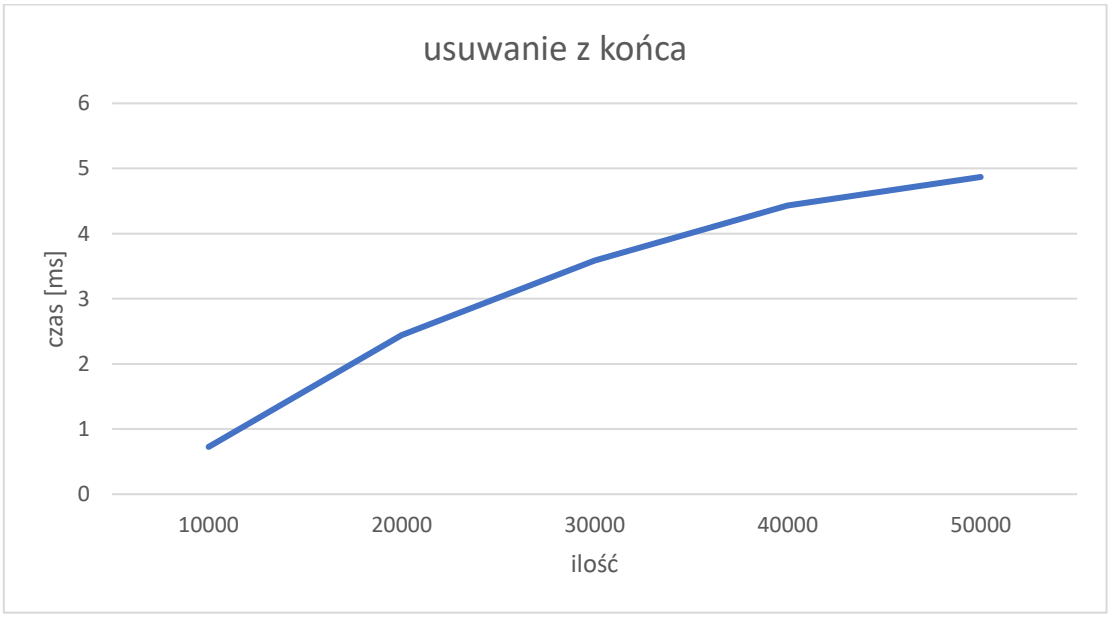




- **Usuwanie**

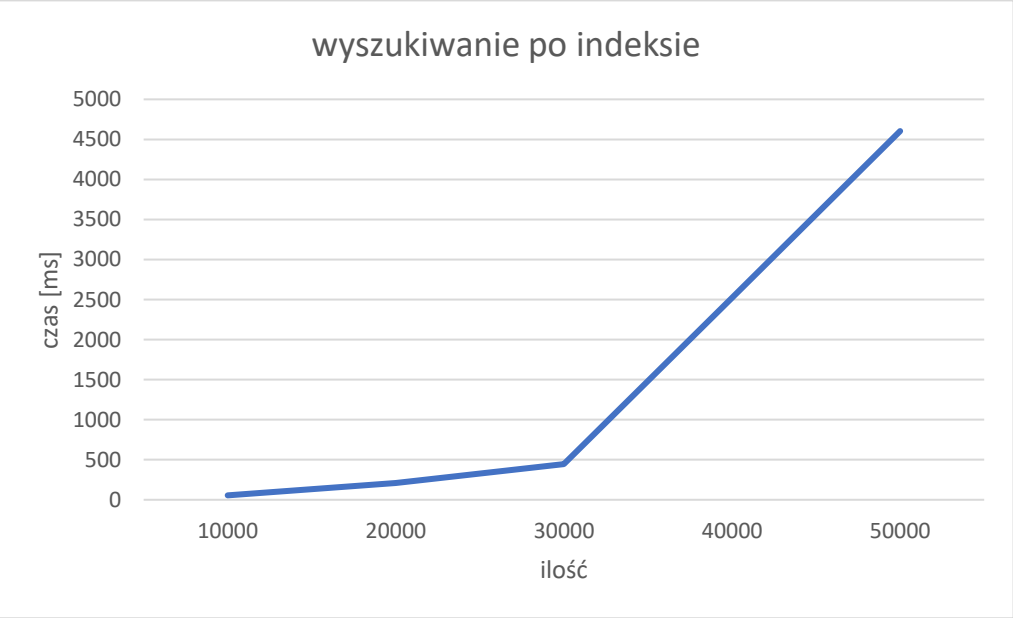
	delete_front [ms]	delete_back [ms]	remove [ms]
10000	0,76	0,73	25,40
20000	2,41	2,44	304,21
30000	3,56	3,59	425,46
40000	4,28	4,43	1147,74
50000	4,82	4,87	1733,48

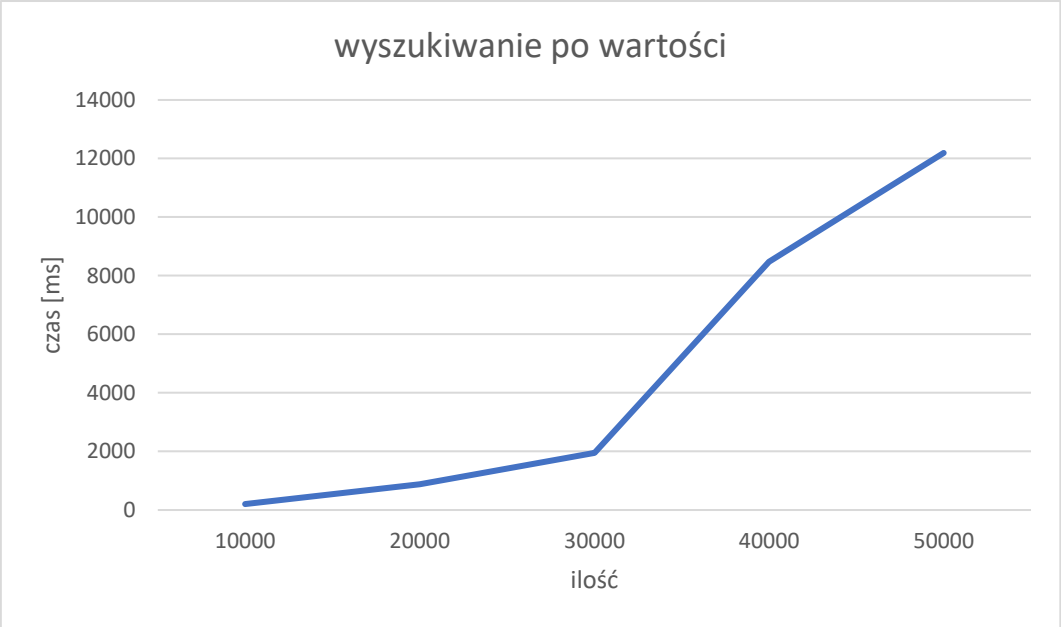

usuwanie z początku
6,000
5,000
4,000
3,000
2,000
1,000
0,000
czas [ms]
10000
20000
30000
40000
50000
ilość


usuwanie z końca
6
5
4
3
2
1
0
czas [ms]
10000
20000
30000
40000
50000
ilość

- **Wyszukiwanie**

	find_elem [ms]	find_elem_value [ms]
10000	55,31	202,43
20000	208,53	875,72
30000	444,3	1952,82
40000	2519,3	8470,7
50000	4603,88	12190,76





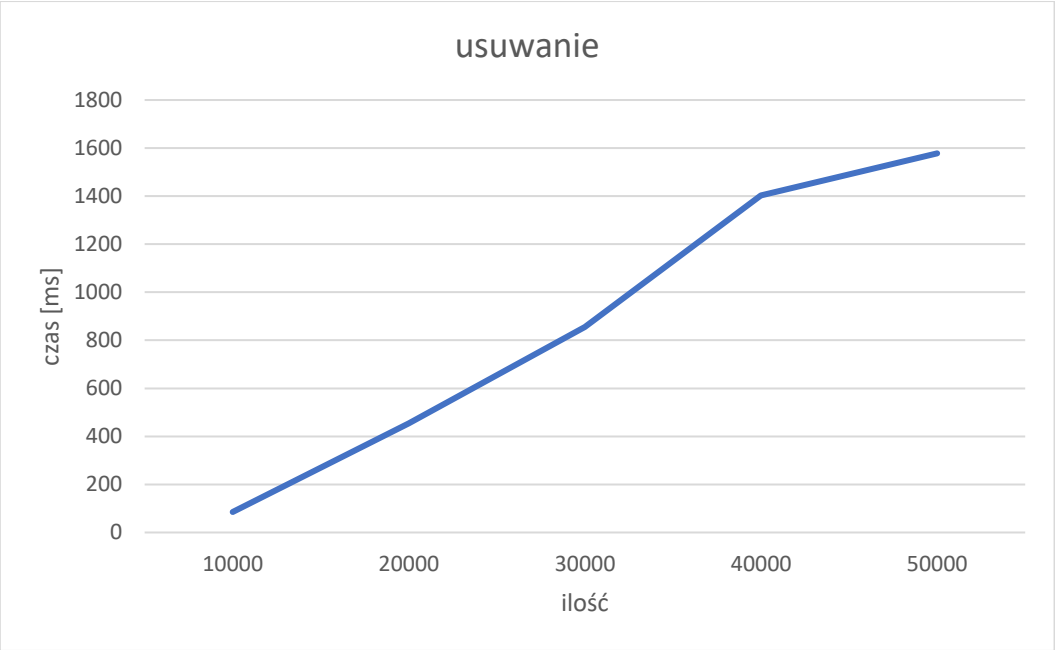
2.3 KOPIEC BINARNY

- **Wstawianie**

	add [ms]
10000	65,23
20000	254,78
30000	555,36
40000	961,02
50000	1577,96

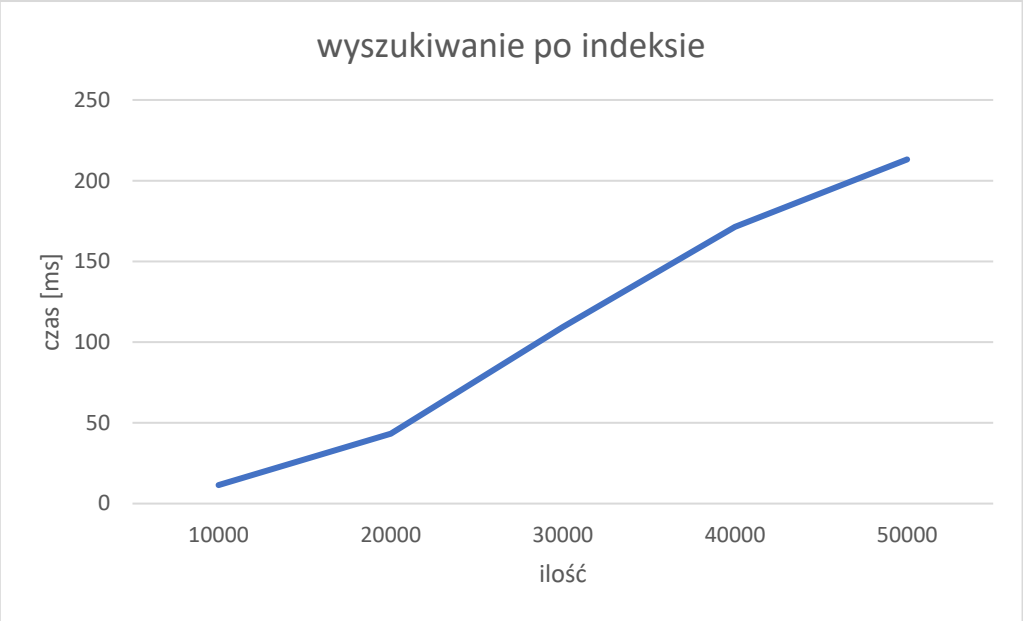
- **Usuwanie**

	remove [ms]
10000	85,23
20000	454,78
30000	855,36
40000	1403,02
50000	1577,96



- **Wyszukiwanie**

	find_elem [ms]	find_elem_value [ms]
10000	11,47	24,76
20000	43,21	96,17
30000	109,32	215,76
40000	171,4	404,02
50000	213,15	601,79



3. PODSUMOWANIE

Struktury działają poprawnie. Ze względu na różne czasy wykonywania poszczególnych operacji, różne struktury nadają się do innych celów. Operacje na tablicy dynamicznej przy dużych ilościach danych zajmują dużo czasu, lecz dzięki indeksowaniu danych, mamy możliwość odniesienia się do nich bezpośrednio. Tablica sprawdzi się, gdy dane nie będą często modyfikowane. W przypadku listy najszybciej wykonywanymi operacjami są operacje wstawiania i usuwania z początku listy(oraz na końcu). Można je zastosować, jak będziemy chcieli często zmieniać dane w pamięci np. stos. Zastosowaniem kopca może być użycie go jako kolejki priorytetowej, ponieważ w korzeniu kopca zawsze mamy wartość największą (akurat w tym przypadku) lub najmniejszą, a czas pozyskania danych nie wzrośnie za bardzo.