Politechnika Wrocławska

Projekt 1 - Struktury Danych
„Lista wiązana a tablica dynamiczna”

Wydział Informatyki i Telekomunikacji
Informatyczne Systemy Automatyki

Aleksander Żołnowski, 272536
Franciszek Jeziorowski, 272526

Spis treści

1 Wstęp teoretyczny

1.1 Struktury danych

Struktury danych są fundamentalnym elementem w programowaniu, służącym do przechowywania oraz organizowania danych w pamięci komputera. Mogą one być porównywane do pojemników lub kontenerów, które pozwalają na gromadzenie danych oraz manipulację nimi w określony sposób. Każda struktura danych ma swoje charakterystyczne cechy i zastosowania, co sprawia, że jest odpowiednia do różnych typów operacji i algorytmów. Wybór odpowiedniej struktury danych jest istotny podczas tworzenia programu i może znacząco wpłynąć na jego wydajność oraz łatwość implementacji algorytmów, które operują na strukturach danych.

1.2 Lista wiązana

Lista wiązana jest jedną z fundamentalnych struktur danych, która umożliwia przechowywanie i organizowanie danych w sposób dynamiczny. Pozwala ona elastycznie zmieniać swoją długość w trakcie działania programu, co czyni ją bardziej wszechstronną w wielu scenariuszach. Główną ideą listy wiązanej jest tworzenie sekwencji elementów zwanych węzłami, z których każdy przechowuje wartość oraz wskaźnik do następnego węzła w liście. Dzięki temu, każdy element może być przechowywany w różnych miejscach w pamięci komputera, co umożliwia dynamiczne dodawanie, usuwanie oraz modyfikowanie elementów bez konieczności przesuwania innych elementów. Lista wiązana jest szczególnie przydatna w przypadkach, gdy nieznana jest liczba elementów, które będą przechowywane, lub gdy konieczne jest częste dodawanie i usuwanie elementów.

1.3 Tablica dynamiczna

Tablica dynamiczna to struktura danych, w której dane są przechowywane w pamięci komputera jako bloki pamięci, które są lokowane obok siebie. W przeciwieństwie do tradycyjnych tablic, których rozmiar jest ustalony na etapie kompilacji i zajmują one stałą przestrzeń pamięci, tablice dynamiczne są alokowane dynamicznie podczas działania programu. Podczas tworzenia tablicy dynamicznej, alokowana jest początkowo pewna ilość pamięci dla przechowywania danych. Gdy liczba elementów w tablicy dynamicznej przekracza jej aktualną pojemność, nowa, większa pamięć jest alokowana, a istniejące dane są przenoszone do nowej lokalizacji.
Dostęp do poszczególnych elementów tablicy odbywa się za pomocą indeksów. Każdy element tablicy jest przechowywany w bloku pamięci o określonym rozmiarze, a dostęp do elementu jest wykonywany poprzez obliczenie adresu pamięci elementu na podstawie jego indeksu oraz rozmiaru elementu i przesunięcia odpowiedniej liczby bajtów od początku tablicy. Dzięki tej strukturze przechowywania danych, tablice dynamiczne pozwalają na szybki

dostęp do elementów oraz efektywne zarządzanie pamięcią w trakcie działania programu. Ich elastyczność i wydajność sprawiają, że są one powszechnie stosowane.

2 Implementacja

2.1 Lista wiązana

Pierwszym krokiem w tworzeniu listy wiązanej było przygotowanie struktury węzła „Node”, która jest niezbędną do wykonywania operacji i samej implementacji listy wiązanej.

```
template <class T>
struct Node
{
    T data;
    Node* next;
};
```

Struktura węzła zawiera w sobie dwie zmienne. Jedna z nich „T data”, to wartość elementu, który będzie częścią listy wiązanej. Typ wartości elementu jest uniwersalny dzięki użyciu szablonu (template). Drugi natomiast „Node* next”, to wskaźnik na następny węzeł, co umożliwia stworzenie listy wiązanej.

Kolejnym krokiem było utworzenie klasy listy wiązanej „LinkedList”. Wykorzystuje szablon (template) w celu umożliwienia przechowywania danych różnych typów. Klasa ta jako swój atrybut ma wskaźnik „Node<T>* head”, który wskazuje na pierwszy węzeł listy.

```
template <class T>
class LinkedList
{
private:
    Node<T>* head;
```

Klasa ta posiada wiele metod, które umożliwiają stworzenie struktury danych zwanej lista wiązana. Każda metoda jest zaprojektowana w taki sposób, aby działała z różnymi typami danych, dzięki użyciu szablonu.

```
void add_end(T);
/*
* Function to add Node at the end of the list
* @param T - value of element
*/

void add_begining(T);
/*
* Function to insert Node at the begining of the list
* @param T - value of element
*/

void add_nth(T, int);
/*
* Function to insert Node at chosen(nth) postion of the list
* @param T - value of element
* @param int -  position where we want to insert node
*/
```

Metody umożliwiają dodawania elementów w dowolnym miejscu listy

```
void remove_begining();
/*
*Function to remove Node from the begining of the list
*/

void remove_nth(int);
/*
* Function to remove Node at chosen(nth) postion of the list
* @param int -  position where we want to remove node
*/

void remove_end();
/*
* Function to remove Node from the end of the list
*/
```

Metody umożliwiają usuwanie elementów z dowolnego miejsca na liście

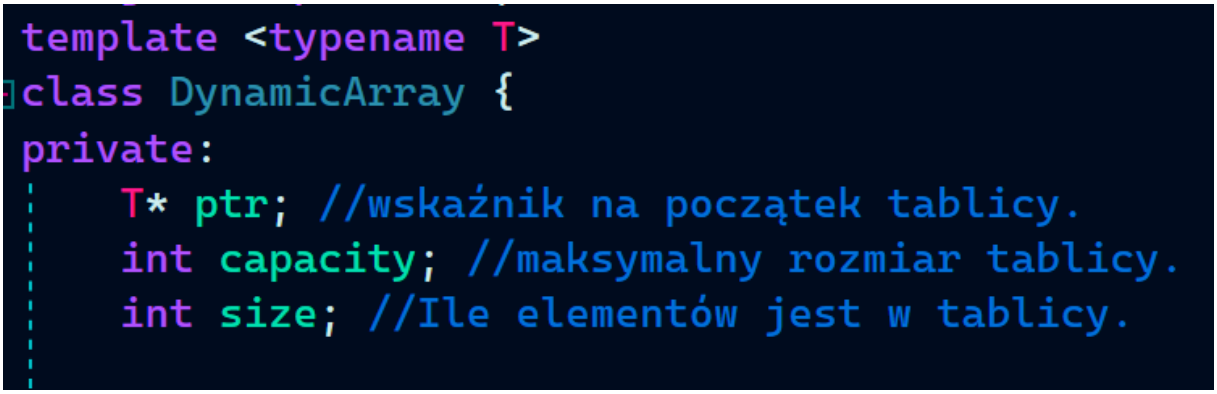
```
void check(T);
/*
* Function to check if the value is element of the list
* @param T - value of element
*/

void print();
/*
* Function to print linked list
*/

int get_size();
/*
* Function to get size of linked list
*/

void clear();
/*
* Function to clear linked list - unlock memory
*/
```

Dodatkowo są metody, dzięki którym możemy sprawdzić czy dana wartość jest elementem listy, wyświetlić wszystkie elementy, sprawdzić rozmiar listy i usunąć wszystkie elementy z listy wiązanej.
Każda metoda ma swój komentarz w pliku nagłówkowym .h zgodny z dockblockiem. Kod natomiast jest w pliku .cpp. Sposoby te zostały użyte w celu zachowania przejrzystości kodu.

Dokładna implementacji i pliki projektu będą załączone w repozytorium github https://github.com/AbaturDev/Struktury-Danych-Projekt1

2.2 Tablica dynamiczna

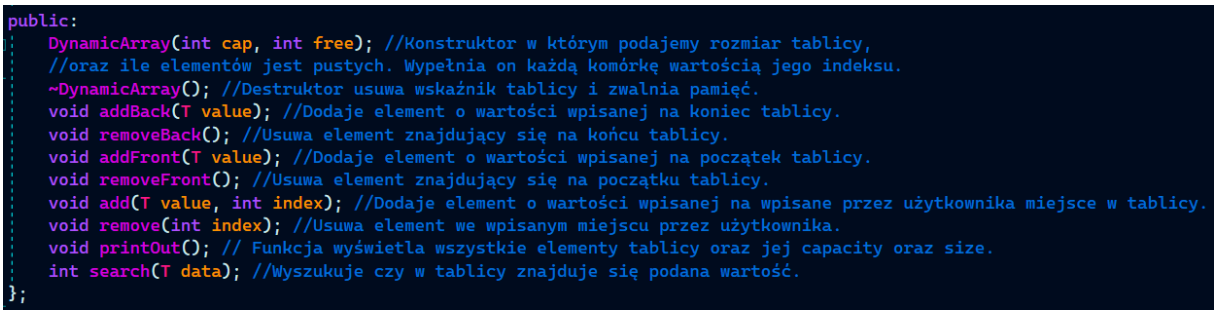
Pierwszym krokiem w tworzeniu tablicy dynamicznej jest utworzenie jej 3 podstawowych zmiennych: ptr - wskaźnik na początek tablicy, size – czyli ilość elementów w tablicy, oraz capacity – maksymalny rozmiar tablicy.



Rysunek 1 Zmienne tablicy dynamicznej

W tej samej klasie zostały również zaimplementowane metody służące do:

- dodawania i usuwania elementu na końcu tablicy (addBack, removeBack)
- dodawania i usuwania elementu na początku tablicy (addFront, removeFront)
- dodawania i usuwania elementu w dowolnym miejscu tablicy (add, remove)
- wyszukania czy konkretna wartość choć raz znajduje się w tablicy (search)
- wyświetlania wszystkich zmiennych tablicy oraz jej zawartości (printOut)



Rysunek 2 Metody występujące w implementacji tablicy dynamicznej

Dokładna implementacji i pliki projektu będą załączone w repozytorium github https://github.com/farfek/Struktury-danych

3 Badania

3.1 Lista wiązana

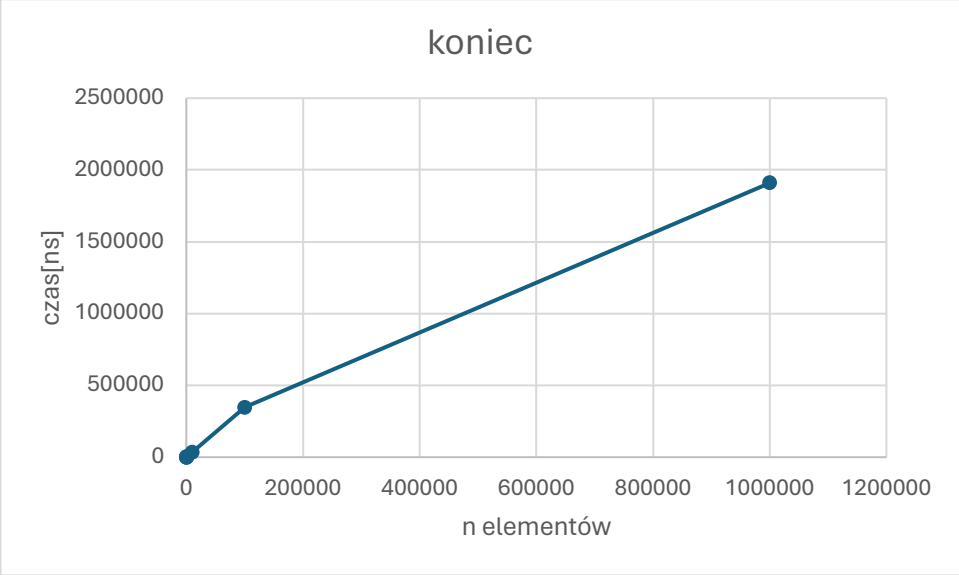
W celu ocenienia wydajność listy wiązanej, przeprowadzono serię testów operacji dodawania, usuwania i przeszukiwania elementów w zależności od jej wielkości. Testy zostały przeprowadzone na różnych rozmiarach listy, które zwiększały się logarytmicznie o 10-krotność dla każdego kroku. Zakres wielkości listy wynosił od 1 do 1 000 000 elementów. Wydajność została zmierzona w nanosekundach(ns).

3.1.1 Dodawanie elementów

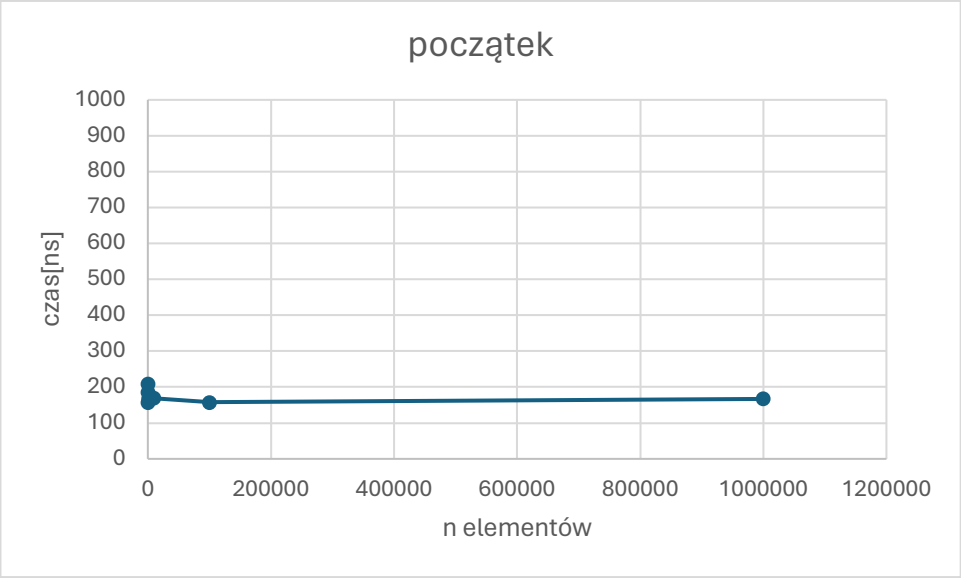
Badanie wydajności dodawania elementów na: początek, koniec i środek(n/2)

Dodawanie elementów					
Koniec		początek		dowolnie (środek n/2)	
n	czas(ns)	N	czas(ns)	n	czas(ns)
1	300	1	157	1	292
10	324	10	208	10	333
100	608	100	185	100	468
1000	3933	1000	167	1000	1958
10000	35258	10000	169	10000	18775
100000	346225	100000	158	100000	158283
1000000	1909008	1000000	167	1000000	906533

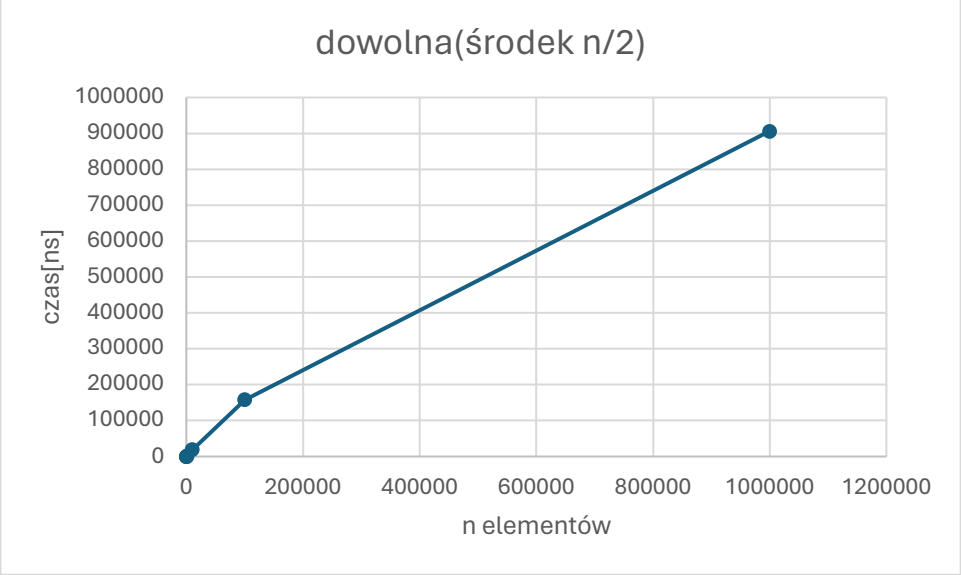
Tabela 1 Wyniki dodawania elementów na wybranych pozycjach



Rysunek 3 Charakterystyka złożoności czasowej dla dodawania na końcu



Rysunek 4 Charakterystyka złożoności czasowej dla dodawania na początku



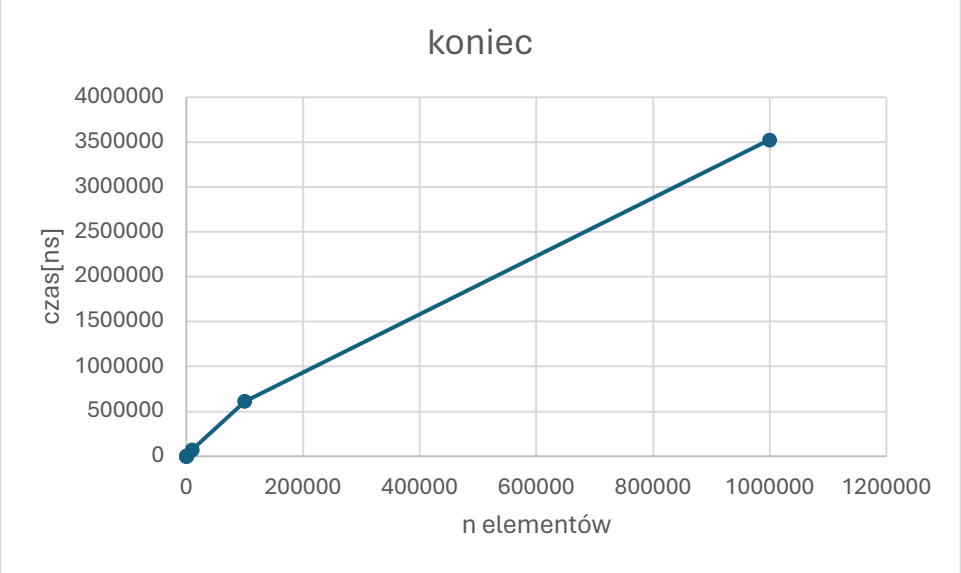
Rysunek 5 Charakterystyka złożoności czasowej dla dodawania na dowolnej pozycji(środek n/2)

3.1.2 Usuwanie elementów

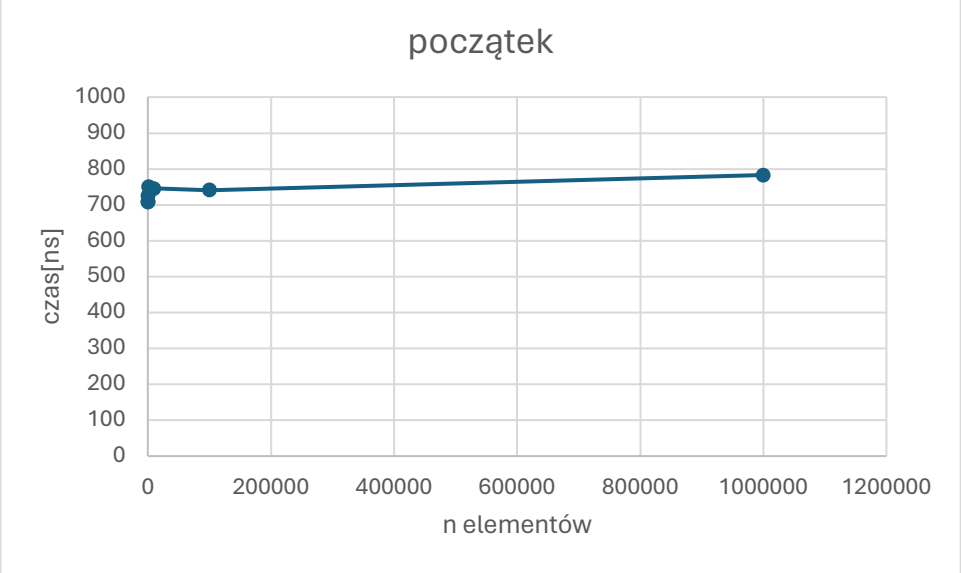
Badanie wydajności usuwania elementów na: początek, koniec i środek(n/2)

Usuwanie elementów					
Koniec		Początek		dowolnie (środek n/2)	
n	czas(ns)	N	czas(ns)	n	czas(ns)
1	792	1	725	1	808
10	900	10	709	10	900
100	1399	100	708	100	911
1000	7166	1000	750	1000	2491
10000	74150	10000	745	10000	16733
100000	612334	100000	741	100000	166800
1000000	3527667	1000000	783	1000000	963016

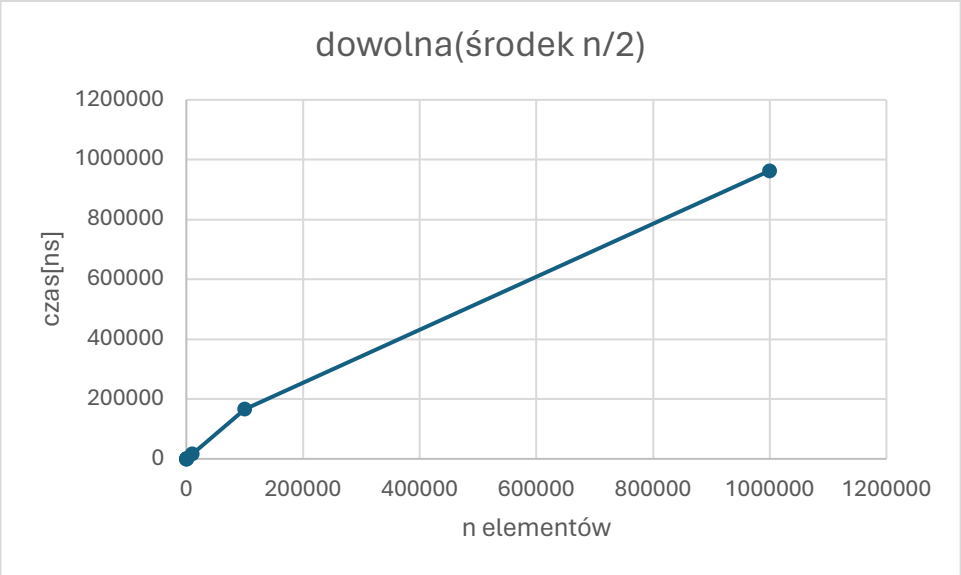
Tabela 2 Wyniki usuwania elementów na wybranych pozycjach



Rysunek 6 Charakterystyka złożoności czasowej dla usuwania na końcu



Rysunek 7 Charakterystyka złożoności czasowej dla usuwania na początku



Rysunek 8 Charakterystyka złożoności czasowej dla usuwania na dowolnej pozycji(środek n/2)

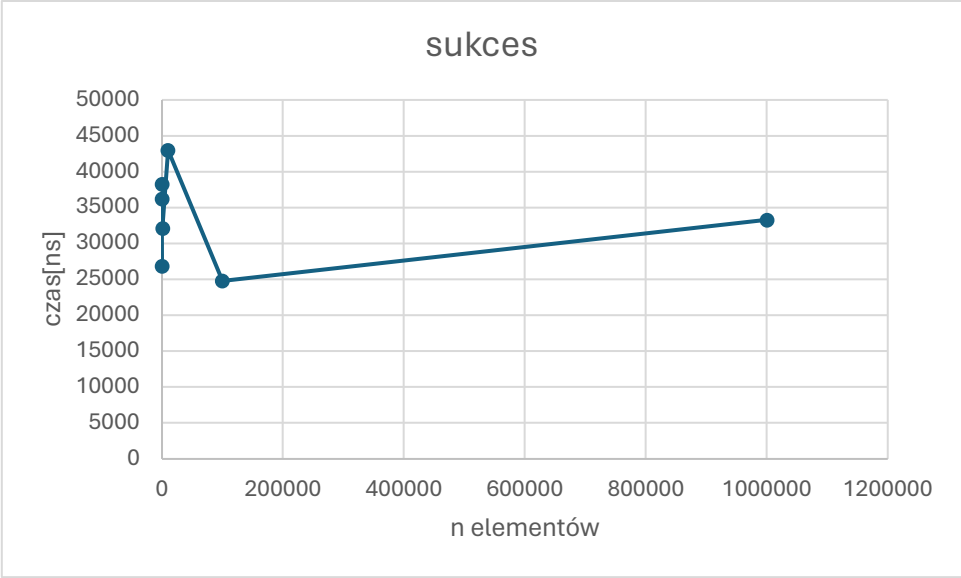
3.1.3 Przeszukiwanie

Badanie przeszukania listy, sprawdzenie czy dana wartość należy do listy, polegało na wypełnieniu n-elementowej listy losowymi wartościami typu int z przedziału od 1 do 100, a następnie przeszukaniu listy w celu znalezienia wybranej wartości.

Badanie przeszukania zakończone sukcesem

przeszukanie(sukces)	
N	czas(ns)
1	26875
10	38250
100	36167
1000	32083
10000	43000
100000	24791
1000000	33292

Tabela 3 Wyniki przeszukania listy zakończone sukcesem

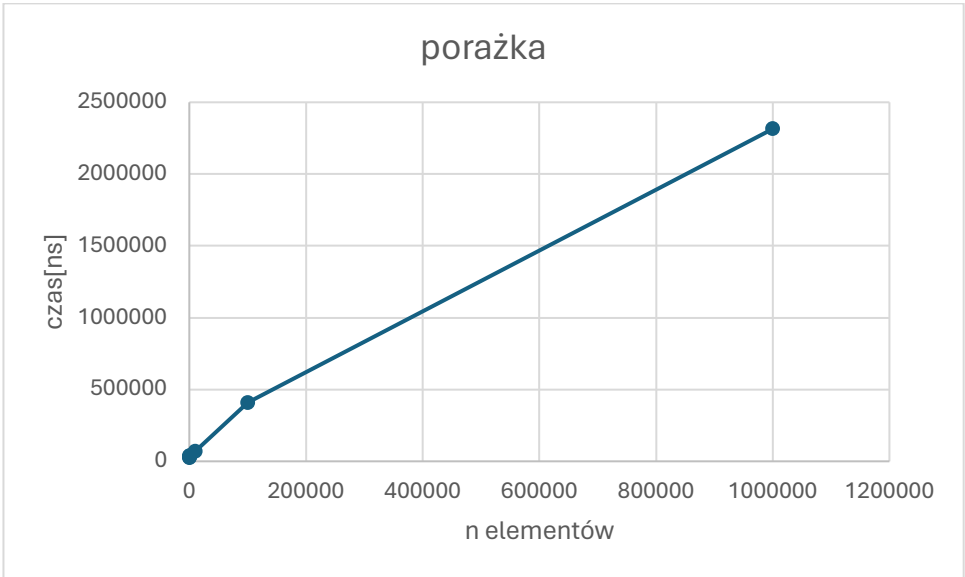


Rysunek 9 Charakterystyka złożoności czasowej dla przeszukania (sukces)

Badanie przeszukania zakończone porażką.

przeszukanie(porażka)	
N	czas(ns)
1	23666
10	29625
100	35625
1000	37334
10000	70335
100000	408375
1000000	2314652

Tabela 4Wyniki przeszukania listy zakończone porażką



Rysunek 10 Charakterystyka złożoności czasowej dla przeszukania (porażka)

3.1.4 Złożoność obliczeniowa

Następnie przeanalizowano złożoność wyników zgodnie z notacja dużego O.

Operacja	Notacja dużego O
Dodawanie na końcu	O(n)
Dodawanie na początku	O(1)
Dodawanie na dowolnej pozycji(n/2)	O(n)
Usuwanie na końcu	O(n)
Usuwanie na początku	O(1)
Usuwanie na dowolnej pozycji(n/2)	O(n)
Przeszukanie(porażka)	O(n)
Przeszukanie(sukces)	-

Tabela 5 Analiza wyników operacji na liście wiązanej zgodnie z notacja dużego O

Wyniki otrzymane po przebadaniu wydajności struktury zostały porównane z wynikami z literatury naukowej.

	Array	Linked List
Cost of accessing elements	O(1)	O(n)
Insert/Remove from beginning	O(n)	O(1)
Insert/Remove from end	O(1)	O(n)
Insert/Remove from middle	O(n)	O(n)

Tabela 6 Wyniki z literatury naukowej

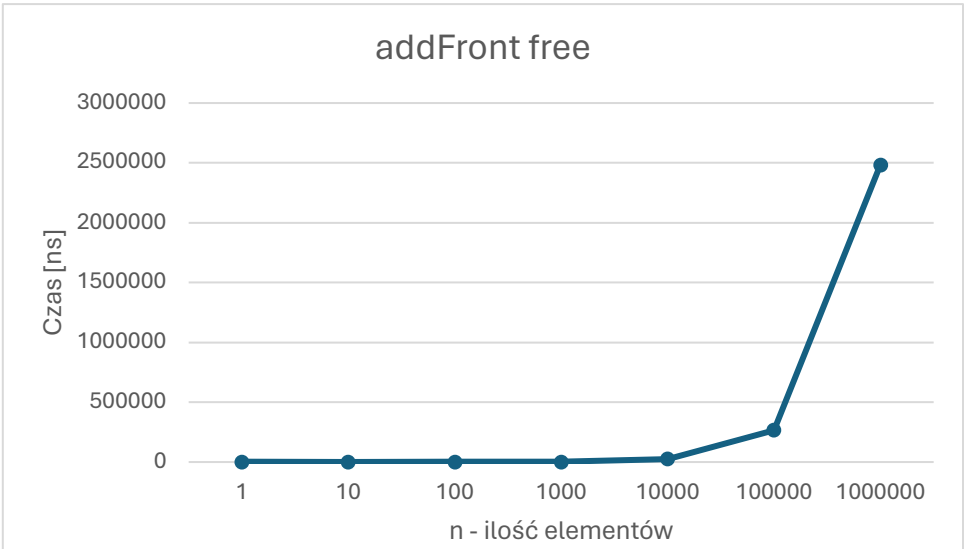
Nasze wyniki potwierdzają oczekiwania teoretyczne i są zgodne z wynikami opisanymi w literaturze.

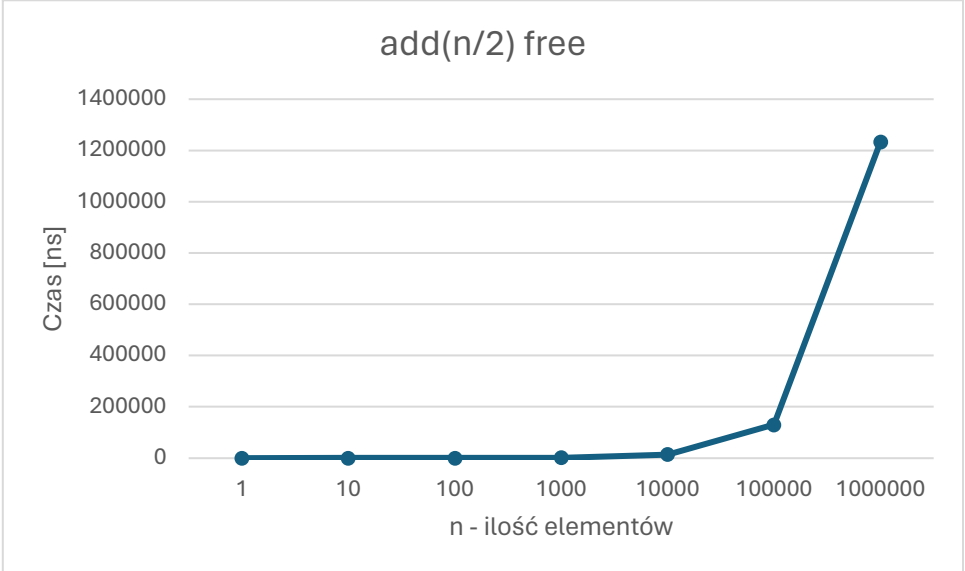
3.2 Tablica dynamiczna

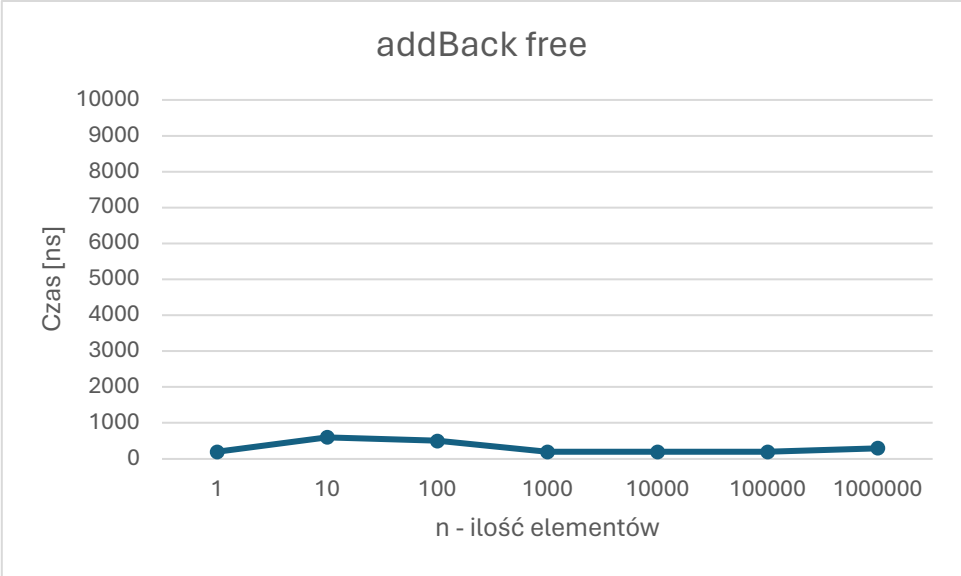
3.2.1 Dodawanie elementów:

Dodawanie elementów w przypadku tablicy z wolnym miejscem:

Dodawanie elementów w przypadku tablicy z wolnym miejscem					
Początek		Połowa		Koniec	
n	t [ns]	n	t [ns]	n	t [ns]
1	500	1	200	1	200
10	400	10	900	10	600
100	800	100	300	100	500
1000	2700	1000	1800	1000	200
10000	26400	10000	14000	10000	200
100000	266400	100000	130800	100000	200
1000000	2484800	1000000	1233500	1000000	300

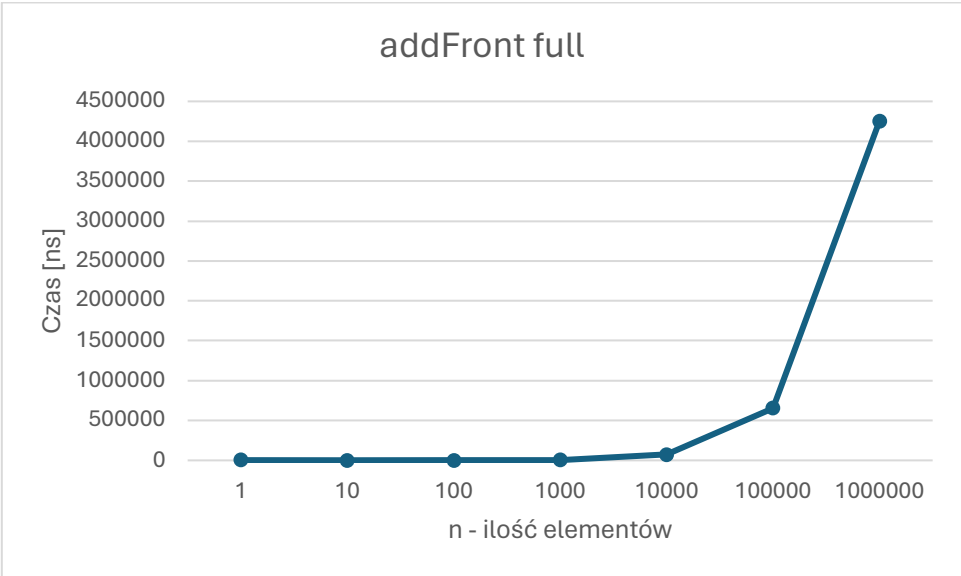


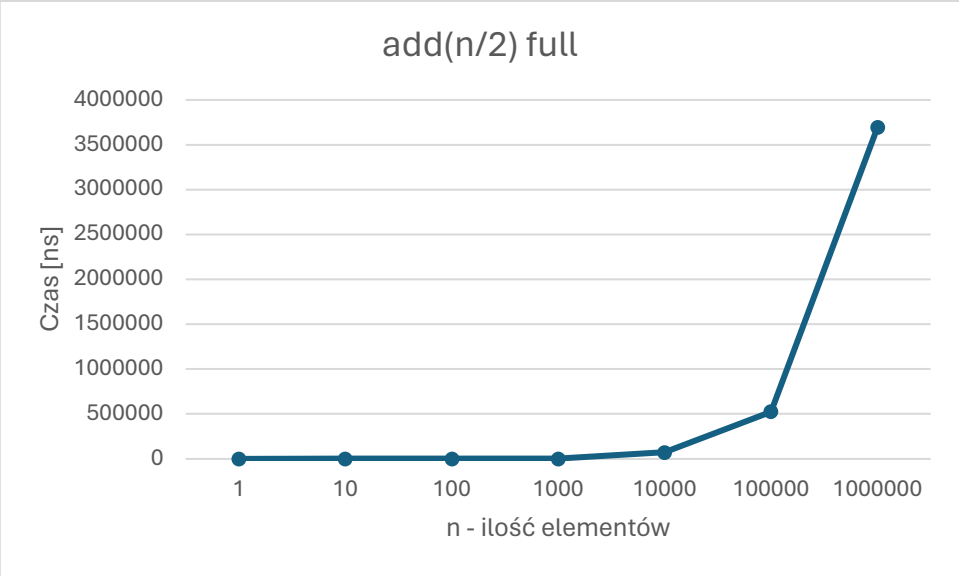


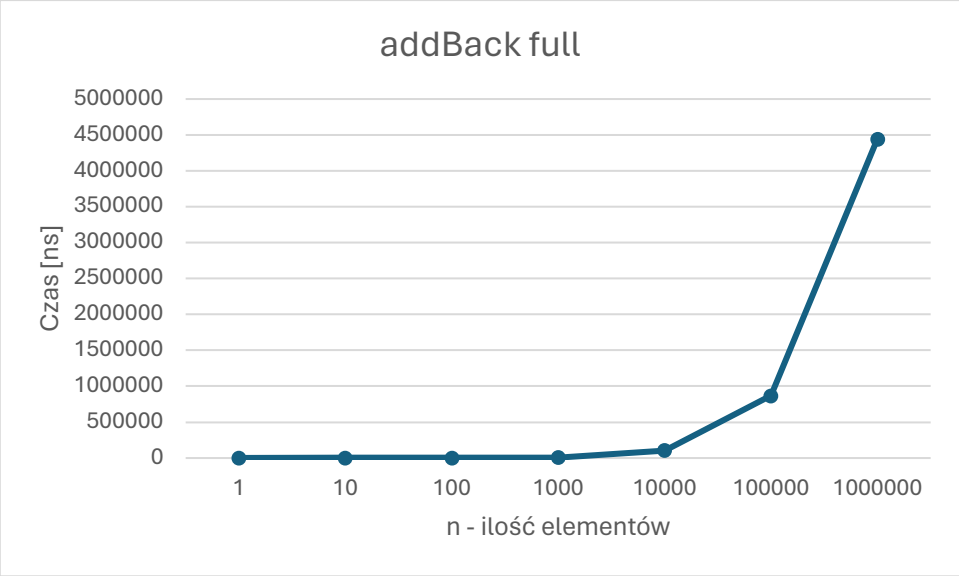


Dodawanie elementów w przypadku pełnej tablicy:

Dodawanie elementów w przypadku pełnej tablicy					
Początek		Połowa		Koniec	
n	t [ns]	n	t [ns]	n	t [ns]
1	3600	1	400	1	1300
10	1800	10	1000	10	2100
100	2000	100	2000	100	2500
1000	5600	1000	3700	1000	8600
10000	72500	10000	73500	10000	104400
100000	655600	100000	527700	100000	866300
1000000	4251600	1000000	3692300	1000000	4445400



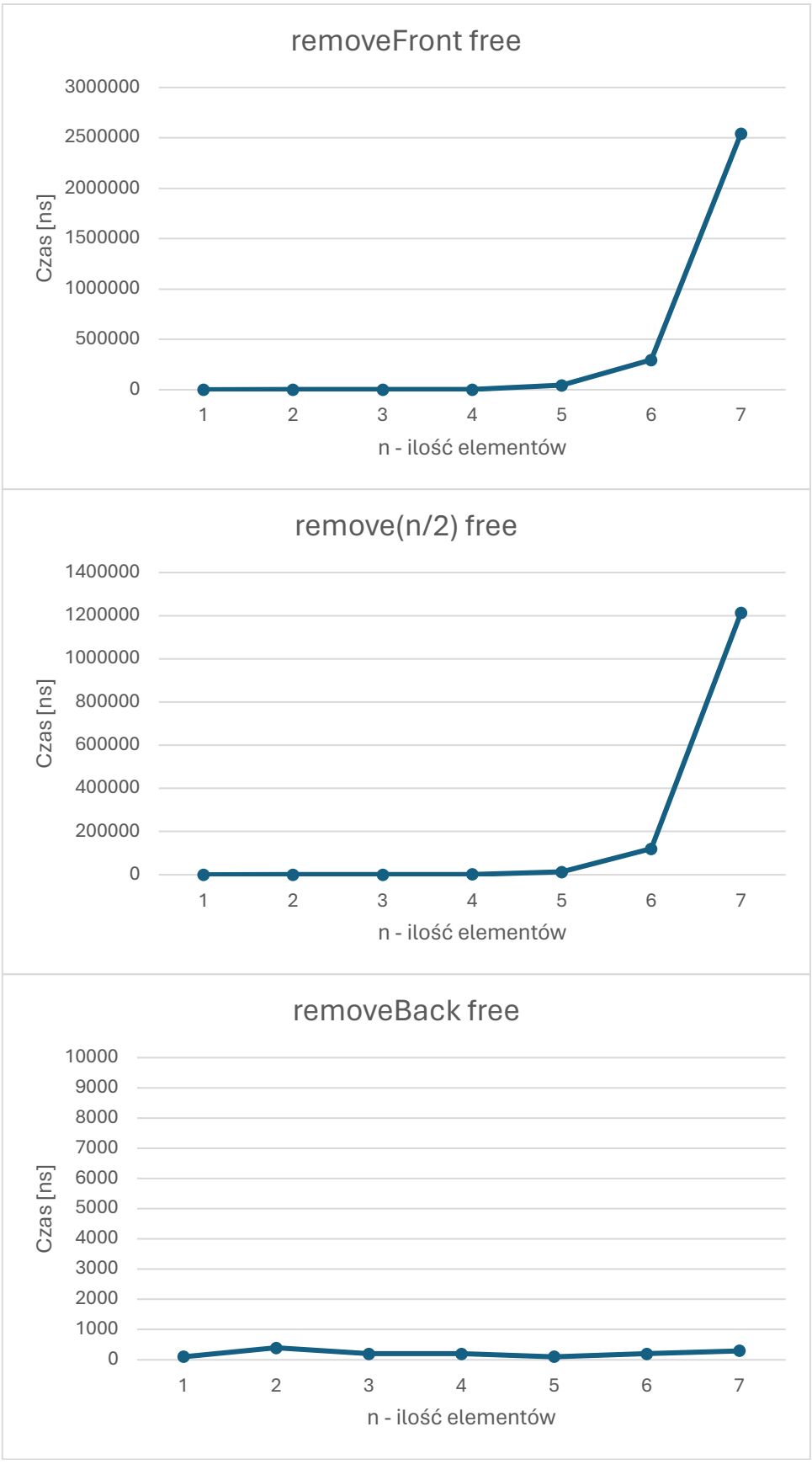




3.2.2 Usuwanie elementów

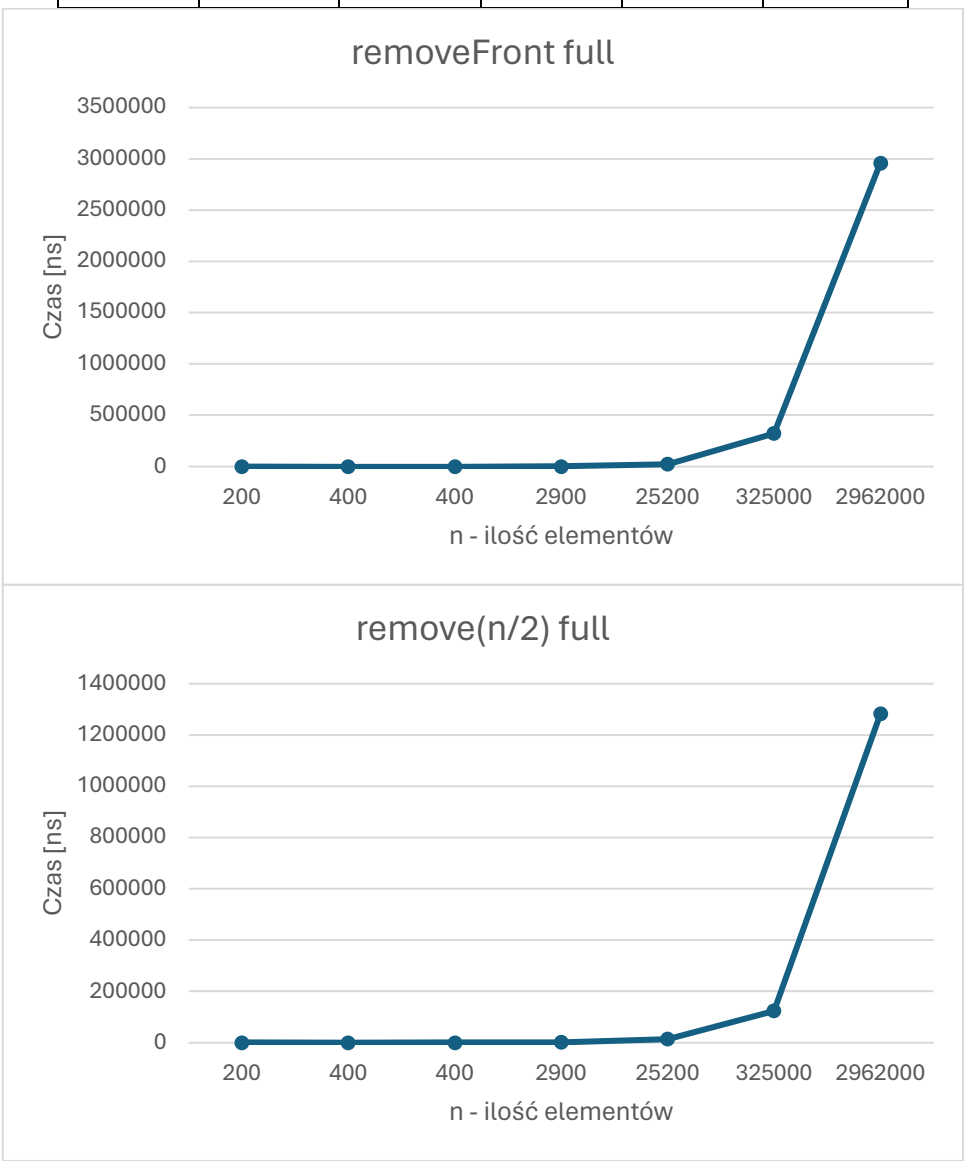
Usuwanie elementów w przypadku tablicy z wolnym miejscem:

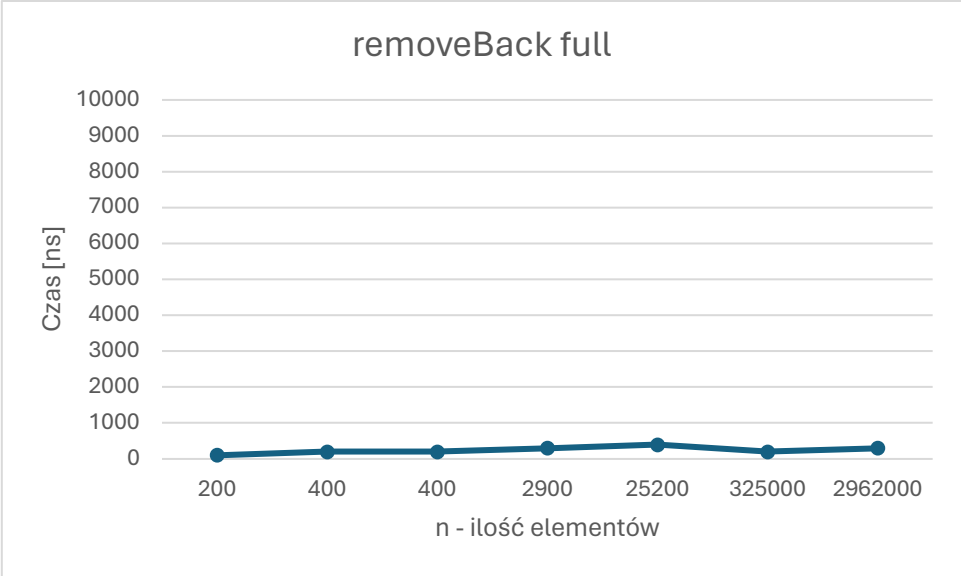
Usuwanie elementów w przypadku tablicy z wolnym miejscem					
Początek		Połowa		Koniec	
n	t [ns]	n	t [ns]	n	t [ns]
1	200	1	100	1	100
10	300	10	500	10	400
100	500	100	400	100	200
1000	2700	1000	1800	1000	200
10000	44700	10000	13300	10000	100
100000	295600	100000	120100	100000	200
1000000	2539700	1000000	1212300	1000000	300


removeFront free
3000000
2500000
2000000
1500000
1000000
500000
0
Czas [ns]
1
2
3
4
5
6
7
n - ilość elementów
remove(n/2) free
1400000
1200000
1000000
800000
600000
400000
200000
0
Czas [ns]
1
2
3
4
5
6
7
n - ilość elementów
removeBack free
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Czas [ns]
1
2
3
4
5
6
7
n - ilość elementów

Usuwanie elementów w przypadku pełnej tablicy:

Usuwanie elementów w przypadku pełnej tablicy					
Początek		Połowa		Koniec	
n	t [ns]	n	t [ns]	n	t [ns]
1	200	1	400	1	100
10	400	10	300	10	200
100	400	100	500	100	200
1000	2900	1000	1500	1000	300
10000	25200	10000	14700	10000	400
100000	325000	100000	124700	100000	200
1000000	2962000	1000000	1282700	1000000	300

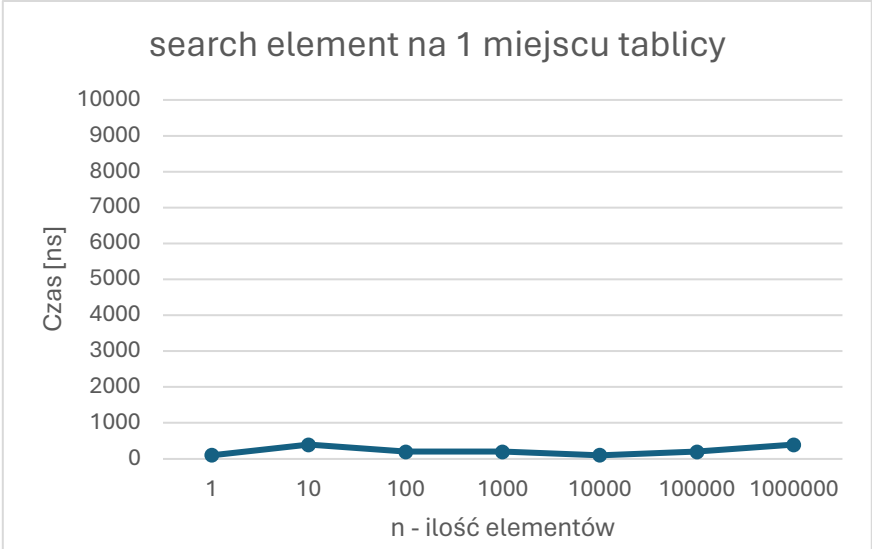


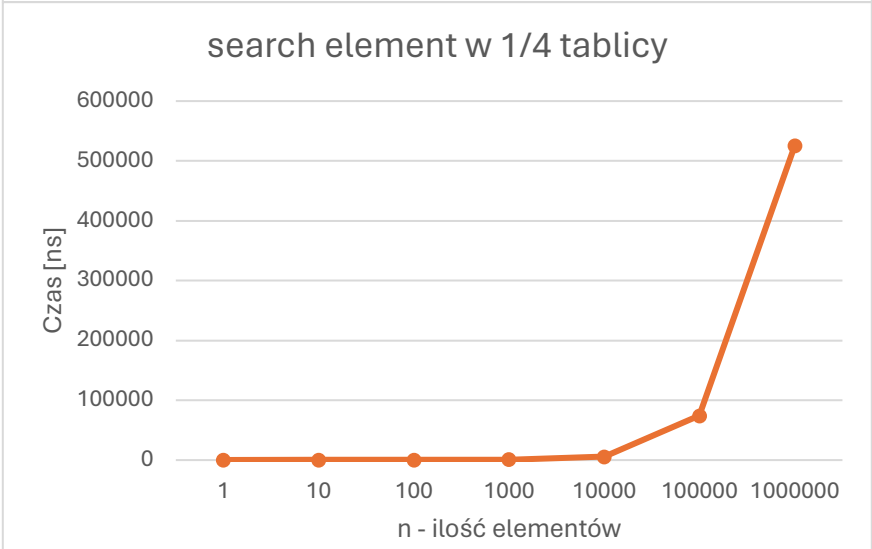


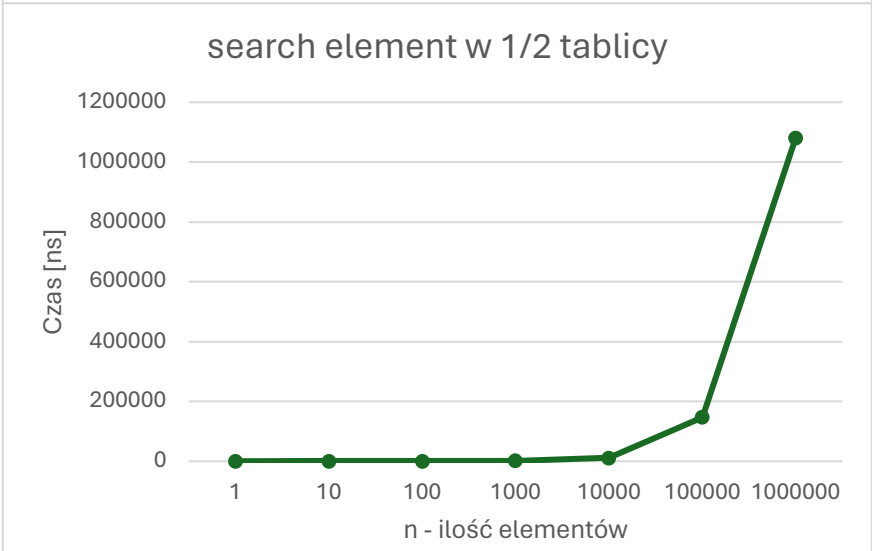
3.2.3 Przeszukiwanie

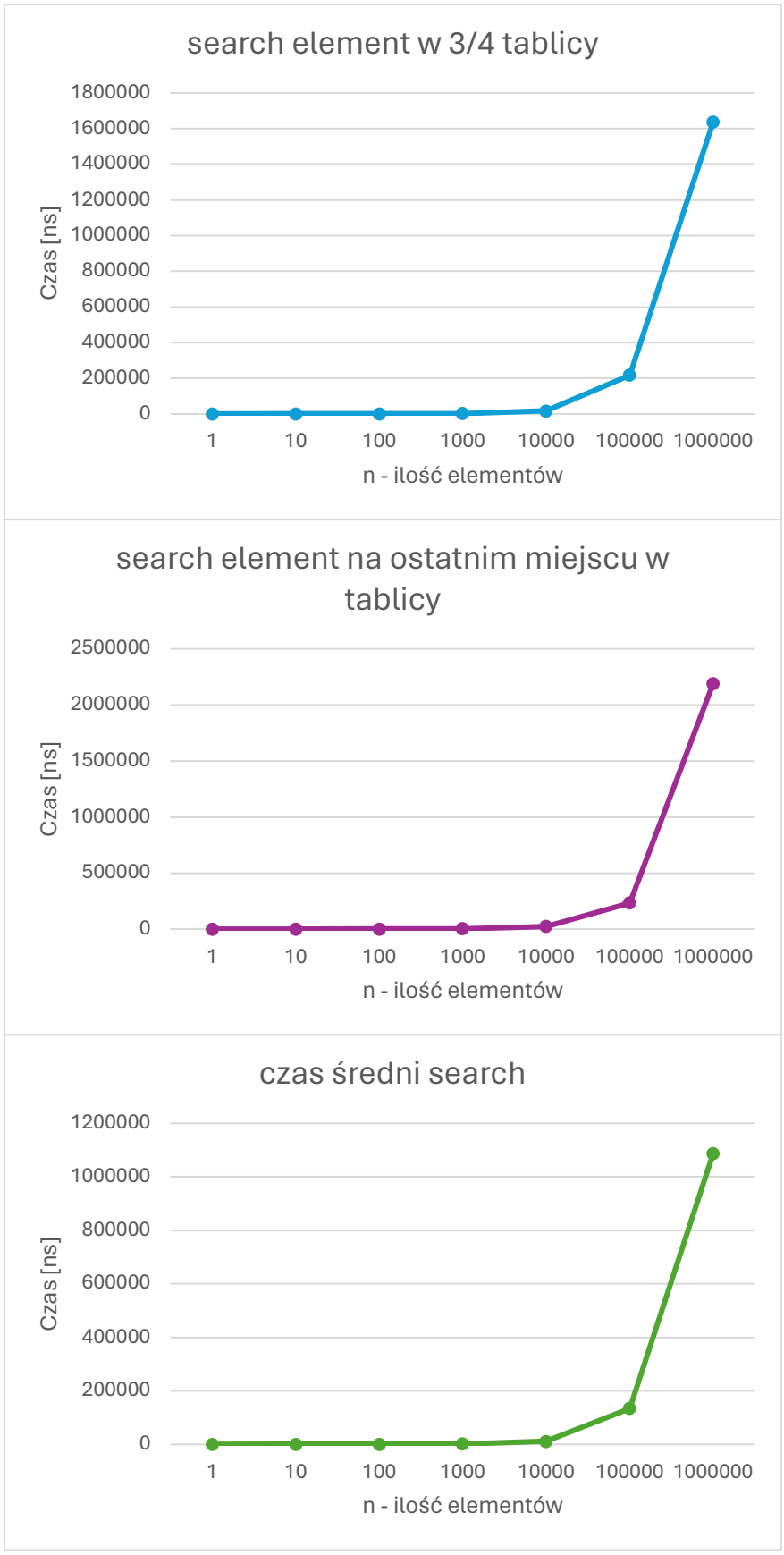
Wyszukiwanie elementów w tablicy dynamicznej					
Element na 1 miejscu		Element w 1/4 tablicy		Element w 1/2 tablicy	
n	t [ns]	n	t [ns]	n	t [ns]
1	100	1	200	1	100
10	400	10	300	10	400
100	200	100	300	100	300
1000	200	1000	800	1000	1600
10000	100	10000	5500	10000	10800
100000	200	100000	74300	100000	147700
1000000	400	1000000	525900	1000000	1081300

Wyszukiwanie elementów w tablicy dynamicznej							
Element w 3/4 tablicy		Element na ostatnim miejscu		Średni czas wszystkich		Nie znaleziono	
n	t [ns]	n	t [ns]	n	t [ns]	n	t [ns]
1	100	1	200	1	140	1	200
10	400	10	200	10	340	10	100
100	500	100	400	100	340	100	500
1000	1900	1000	2700	1000	1440	1000	3000
10000	17700	10000	23500	10000	11520	10000	23600
100000	217600	100000	232500	100000	134460	100000	231900
1000000	1636700	1000000	2190600	1000000	1086980	1000000	2088200


search element na 1 miejscu tablicy
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Czas [ns]
1
10
100
1000
10000
100000
1000000
n - ilość elementów


search element w 1/4 tablicy
600000
500000
400000
300000
200000
100000
0
Czas [ns]
1
10
100
1000
10000
100000
1000000
n - ilość elementów


search element w 1/2 tablicy
1200000
1000000
800000
600000
400000
200000
0
Czas [ns]
1
10
100
1000
10000
100000
1000000
n - ilość elementów


search element w 3/4 tablicy
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
Czas [ns]
1
10
100
1000
10000
100000
1000000
n - ilość elementów
search element na ostatnim miejscu w tablicy
2500000
2000000
1500000
1000000
500000
0
Czas [ns]
1
10
100
1000
10000
100000
1000000
n - ilość elementów
czas średni search
1200000
1000000
800000
600000
400000
200000
0
Czas [ns]
1
10
100
1000
10000
100000
1000000
n - ilość elementów

3.2.4 Złożoność obliczeniowa

Po przebadaniu wyników oraz wykresów dochodzę do następujących wniosków:

Dodawanie na początku wersja optymistyczna	O(n)
Dodawanie w środku wersja optymistyczna	O(n)
Dodawanie na końcu wersja optymistyczna	O(1)
Dodawanie na początku wersja pesymistyczna	O(n)
Dodawanie w środku wersja pesymistyczna	O(n)
Dodawanie na końcu wersja pesymistyczna	O(n)
Usuwanie na początku	O(n)
Usuwanie w środku	O(n)
Usuwanie na końcu	O(1)
Przeszukiwanie z sukcesem średnia	O(n)
Przeszukiwanie bez sukcesu	O(n)

Przy porównaniu do analizy z wykładu Pana Magistra Inżyniera Jarosława Rudego oraz książek załączonych w bibliografii, wyniki wydają się zgadzać.

Klasyczna analiza:

Operacja	Optymistycznie	Średnio	Pesymisycznie
Dodanie na dowolnej pozycji	$O(1)$	$O(n)$	$O(n)$
Dodanie na początku	$O(n)$	$O(n)$	$O(n)$
Dodanie na końcu	$O(1)$	$O(n)$	$O(n)$

	Array	Linked List
Cost of accessing elements	O(1)	O(n)
Insert/Remove from beginning	O(n)	O(1)
Insert/Remove from end	O(1)	O(n)
Insert/Remove from middle	O(n)	O(n)

Wyszukiwanie:

- Optymistycznie: sprawdzany jeden element – czas $O(1)$.
- Średnio: sprawdzamy połowę elementów tj. $\lceil \frac{n}{2} \rceil$ – czas $O(n)$.
- Pesymistycznie: sprawdzamy wszystkie n elementów – czas $O(n)$.

4 Wnioski

4.1 Lista wiązana

Zalety:
• Dynamiczne rozszerzanie się - Lista wiązana może dynamicznie zmieniać swój rozmiar bez konieczności przepisywania wszystkich elementów.
• Szybkie wstawianie i usuwanie na początku listy - Operacje wstawiania i usuwania na początku listy są bardzo szybkie i mają złożoność czasową O(1).
• Szybki dostęp do elementów na początku listy
• Brak konieczności realokacji pamięci - Nie ma potrzeby przepisywania wszystkich elementów podczas rozszerzania listy.

Wady:
• Wolne wstawianie i usuwanie elementów na końcu listy – Operacje na końcu listy są mało wydajne, mają złożoność obliczeniowa O(n)
• Wolny dostęp do elementów - Odczytanie elementu o określonym indeksie wymaga przejścia przez całą listę od początku O(n).
• Większe zużycie pamięci - Każdy element listy wiązanej zawiera dodatkową informację (wskaźnik) do następnego elementu.

4.2 Tablica dynamiczna

Zalety:
• Szybki dostęp do elementów - Odczytanie elementu o określonym indeksie jest bardzo szybkie i ma złożoność czasową O(1).
• Szybkie wstawianie i usuwanie na końcu tablicy: Operacje wstawiania i usuwania na końcu tablicy są bardzo szybkie i mają złożoność czasową O(1).

Wady:
• Konieczność realokacji pamięci - Gdy tablica osiągnie maksymalny rozmiar, może być konieczna realokacja pamięci, co jest kosztowne pod względem czasu.
• Wolniejsze wstawianie i usuwanie na początku - Operacje wstawiania i usuwania są mało wydajne, ponieważ wymagają przesunięcia elementów w pamięci, złożoność O(n).

4.3 Podsumowanie

• Jeśli często dokonujesz operacji wstawiania i usuwania elementów na początku oraz potrzebujesz dostępu do elementów na początku struktury. Lista wiązana może być lepszym wyborem.
• Jeśli potrzebujesz szybkiego dostępu do elementów i operacji na końcu struktury danych, tablica dynamiczna może być bardziej odpowiednia.

• Wybór między listą wiązaną a tablicą dynamiczną zależy od konkretnej sytuacji i wymagań programu.

5 Bibliografia

• Wprowadzenie do algorytmów Cormen Thomas H., Leiserson Charles E., Rivest Ronald L, Clifford Stein ISBN 978-83-01-16911-4
• Data Structures and Algorithms in C++, 2nd Edition Michael T. Goodrich, Roberto Tamassia, David M. Mount ISBN: 978-0-470-38327-8
• http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/files/sd/w3.pdf