INF PWR	STRUKTURY DANYCH I ZŁOŻONOŚĆ OBLICZENIOWA Badanie efektywności operacji na danych w podstawowych strukturach danych.	
Alicja Myśliwiec Numer indeksu: 248867	Wtorek 15.15 – 16.55 TN	Dr inż. Dariusz Banasiak

1. Cel projektu

Celem projektu jest badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w różnych strukturach danych. Należało zaimplementować oraz dokonać pomiaru czasu działania podanych operacji w następujących strukturach:

- tablica,
- lista dwukierunkowa,
- kopiec binarny (typu maksimum – element maksymalny w korzeniu) ,
- drzewo czerwono-czarne

2. Wstęp teoretyczny

Złożoność obliczeniowa to ilość zasobów komputerowych koniecznych do wykonania progamu realizującego algorytm. Jest przedstawiana jako unkcja pewnego paramteru określającego rozmiar rozwiązywanego zadania. W operacji dodawania, usuwania oraz wyszukiwania złożoność obliczeniowa uzależniona jest od rodzaju struktury danych, w których przechowywane są informacje. W dalszej części, do analizy złożoności obliczeniowej struktur będziemy używali **notacji duże - O**.

- **O(1)** – złożoność stała, nie zależy od liczby danych wejściowych
- **O(n)** – złożoność liniowa, jest to przypadek złożoności wielomianowej, gdzie czas rozwiązania problemu jest wprost proporcjonalny do ilości danych
- **O(log(n))** – złożoność logarytmiczna, czas rozwiązania zależy od wyniku logarytmu
- **O(nlog(n))** – złożoność liniowo – logarytmiczna, czas rozwiązania problemu jest wprost proporcjonalny do iloczynu wielkości danych wejściowych i ich logarytmu

Powyżej zostały przedstawione podstawowe złożoności. Wyróżniamy również złożoności kwadratowe, wielomianowe, wykładnicze i typu silnia. Algorytm może mieć różną złożoność obliczeniową określoną w notacji O w zależności od instancji problemu. Są także algorytmy, których złożoność obliczeniowa jest niezależna od instancji problemu. W zależności od wymagań w wyborze algorytmu bierze się pod uwagę złożoność odpowiedniego przypadku.

Tablica dynamiczna

Jest to kontener uporządkowanych danych takiego samego typu, w którym poszczególne elementy dostępne są za pomocą kluczy (indeksu) przyjmujących wartości numeryczne. Ponieważ tablica jest dynamiczna, jej rozmiar zmienia się w trakcie wykonywania programu

Sam dostęp do elementu w tablicy ma złożoność obliczeniową O(1), jednak modyfikacja danego elementu (dodawanie/usuwanie) jest związana z koniecznością ponownego alokowania pamięci, zatem musimy przejść przez całą strukturę. Natomiast wyszukiwanie podanej wartości w tablicy w najgorszym wypadku wymaga przejścia przez wszystkie elementy w niej zawarte, zatem złożoność przedstawia się w następująco:

Operacja	Wyszukanie	Wstawianie			Usuwanie		
		Początek	Koniec	Losowo	Początek	Koniec	Losowo
Złożoność	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)

Lista dwukierunkowa

Jest to struktura danych, która służy do reprezentacji zbiorów dynamicznych, w której elementy ułożone są w porządku liniowym. W liście dwukierunkowej z każdego elementu możliwe jest przejście do jego poprzednika i następnika. Umożliwia to swobodne przemieszczanie się po liście w obie strony. W najgorszym przypadku, aby wyszukać dany element w liście, trzeba przeszukać wszystkie jej elementy, zatem złożoność obliczeniowa wynosi O(n). Sama operacja dodania elementu do listy ma złożoność O(1), ale wcześniej musimy ten element znaleźć, czyli końcowa złożoność dodawania elementu do listy to O(n). Tak samo jest w przypadku usuwania danego elementu. Działania na początku i na końcu listy stanowią pewne odstępstwo, gdyż mamy dostęp do pierwszego i ostatniego elementu w związku z faktem, że ich adresy są bezpośrednio przechowywane w strukturze i złożoność tych działań zawsze będzie taka sama niezależnie od ilości danych, zatem złożoność listy przedstawia się następująco:

Operacja	Wyszukanie	Wstawianie			Usuwanie		
		Początek	Koniec	Losowo	Początek	Koniec	Losowo
Złożoność	O(n)	O(1)	O(1)	O(n)	O(1)	O(1)	O(n)

Kopiec binarny (typu maksimum)

Jest to struktura danych reprezentująca drzewo binarne, którego wszystkie poziomy (oprócz ostatniego) muszą być pełne. Gd ostatni poziom drzewa nie jest pełny, liście ułożone są od lewej do prawej strony drzewa. W kopcu typu maksimum wartość danego węzła niebędącego korzeniem jest zawsze mniejsza niż wartość jego rodzica Wyszukanie elementu w kopcu wymaga (w najgorszym przypadku) przejścia przez wszystkie jego węzły, stąd złożoność O(nlog(n)). Sama operacja dodania nowego elementu na kopiec ma złożoność O(1), ponieważ rozmiar kopca jest przechowywany, a do elementu można odwołać się przez indeksowanie, ale takim działaniem możemy zaburzyć strukturę kopca, co skutkuje koniecznością jego naprawy. Maksymalna liczba napraw kopca jest równa jego wysokości, zatem złożoność wyniesie O(log(n)). Usuwanie danego elementu jest analogiczne do dodawania i jego złożoność również wynosi O(log(n)) ze względu na ilość możliwych napraw po usunięciu, zatem złożoność kopca przedstawia się następująco:

Operacja	Wyszukanie	Wstawianie	Usuwanie
Złożoność	O(nlog(n))	O(log(n))	O(log(n))

Drzewo czerwono – czarne

Jest to rodzaj samoorganizującego się binarnego drzewa poszukiwań, czyli dynamiczna struktura danych będąca drzewem binarnym, w którym lewe poddrzewo każdego węzła zawiera wyłącznie elementy o kluczach mniejszych niż klucz węzła a prawe poddrzewo zawiera wyłącznie elementy o kluczach nie mniejszych niż klucz węzła. Węzły, oprócz klucza, przechowują wskaźniki na swojego lewego i prawego syna oraz na swojego ojca.. W drzewie czerwono-czarnym z każdym węzłem powiązany jest dodatkowy atrybut, kolor, który może być czerwony lub czarny. Oprócz podstawowych własności drzew poszukiwań binarnych, wprowadzone zostały kolejne wymagania, które trzeba spełniać:

1) Każdy węzeł jest czerwony albo czarny.
2) Korzeń jest czarny.
3) Każdy liść jest czarny (Można traktować nil jako liść).
4) Jeśli węzeł jest czerwony, to jego synowie muszą być czarni.
5) Każda ścieżka z ustalonego węzła do każdego z jego potomków będących liśćmi liczy tyle samo czarnych węzłów.

Wymagania te gwarantują, że najdłuższa ścieżka od korzenia do liścia będzie co najwyżej dwukrotnie dłuższa, niż najkrótsza. Zatem łączna długość drugiej ścieżki może wynieść co najwyżej 2n, gdzie n jest długością pierwszej ścieżki zbudowanej wyłącznie z węzłów czarnych. Można udowodnić, że dla n węzłów głębokość drzewa czerwono-czarnego wyniesie najwyżej 2log(n+1), przez co elementarne operacje będą wykonywać się w czasie O(log(n)).

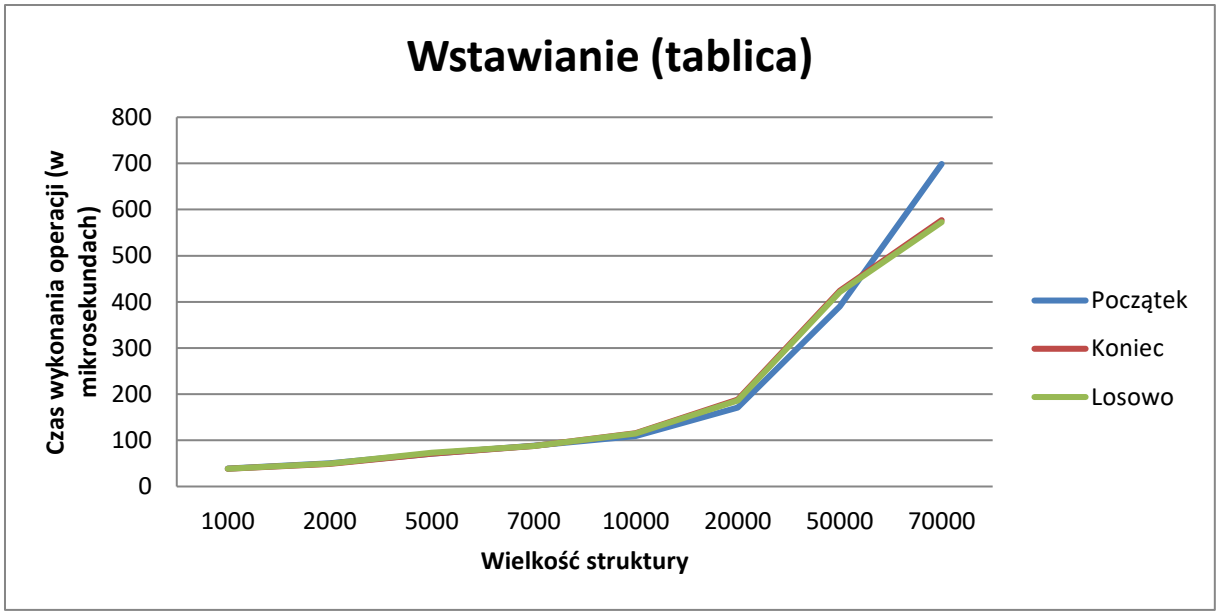
Operacja	Wyszukanie	Wstawianie	Usuwanie
Złożoność	O(log(n))	O(log(n))	O(log(n))

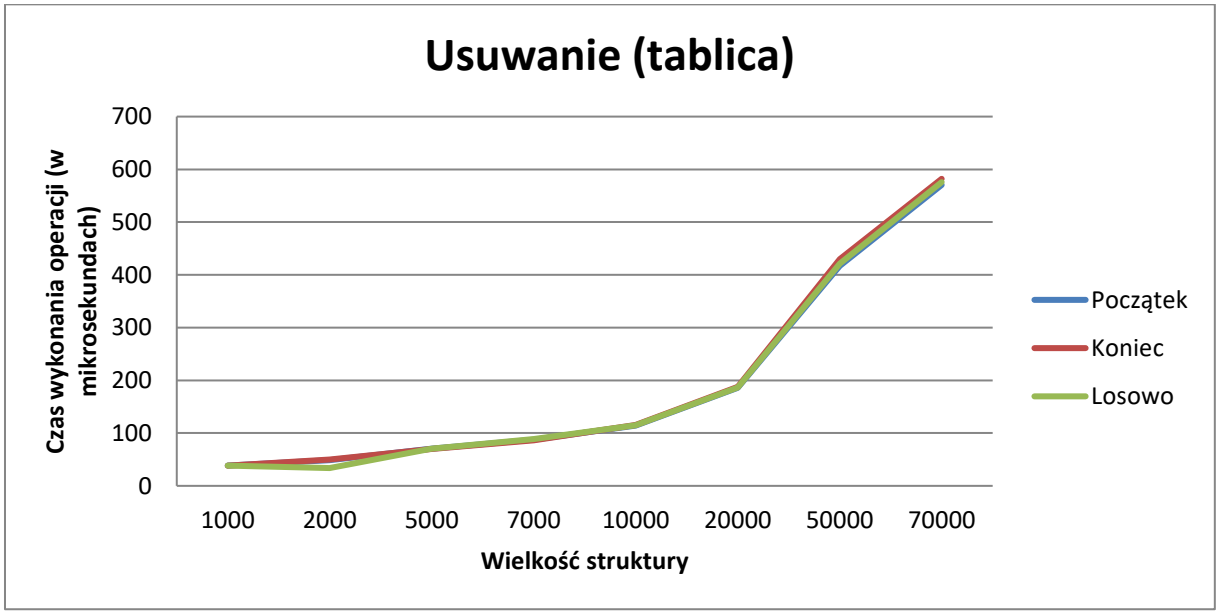
3. Plan projektu

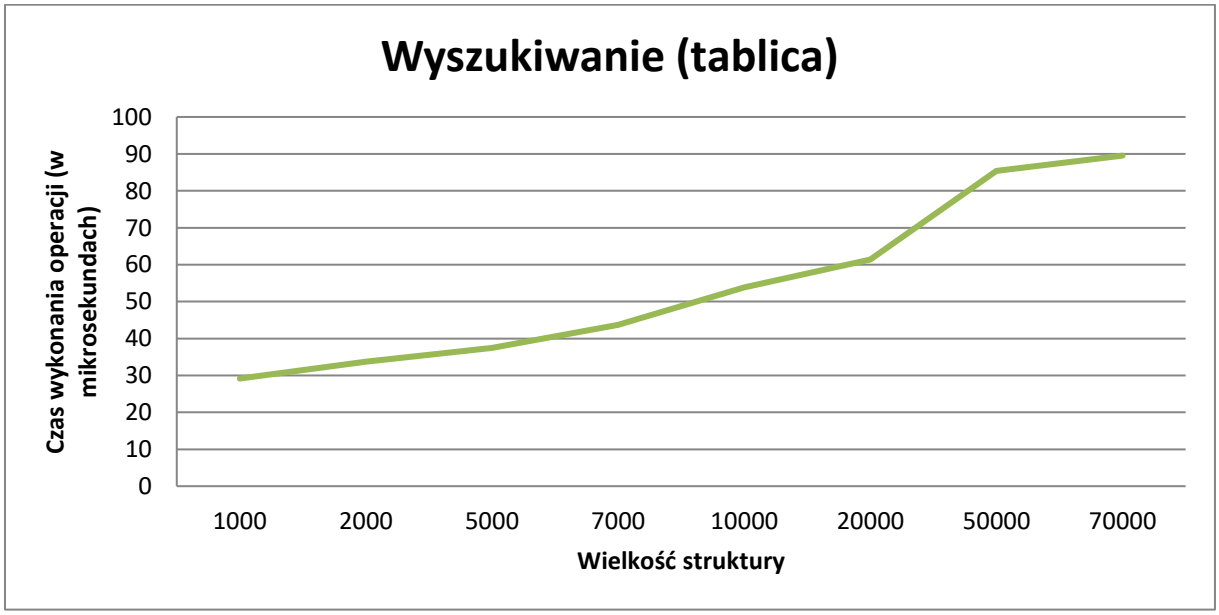
Zadanie polegało na pomiarze czasu wykonywania poszczególnych operacji (wstawianie, usuwanie, wyszukanie) w funkcji rozmiaru danej struktury (liczby elementów w niej przechowywanych). Dla tablicy i listy zostały rozpatrzone dodatkowe operacje dodawania i usuwania z rozróżnieniem na początek, koniec oraz losowe miejsce w strukturze. Ponieważ pojedynczy pomiar może być obarczony znacznym błędem oraz otrzymane wyniki mogą zależeć także od rozkładu danych, to pomiary dla konkretnego rozmiaru struktury zostały wykonane wielokrotnie. Za każdym razem program generował nowy zestaw danych. Łącznie dla każdego z rozmiarów struktur (1000, 2000, 5000, 7000, 10000, 20000, 50000, 70000) przeprowadzono po 100 pomiarów, a wyniki zostały uśrednione i podane w mikrosekundach z dokładnością do części setnych. Do dokładnego pomiaru czasu w systemie Windows w C++ została wykorzystana funkcja QueryPerformanceCounter.

4. Wyniki

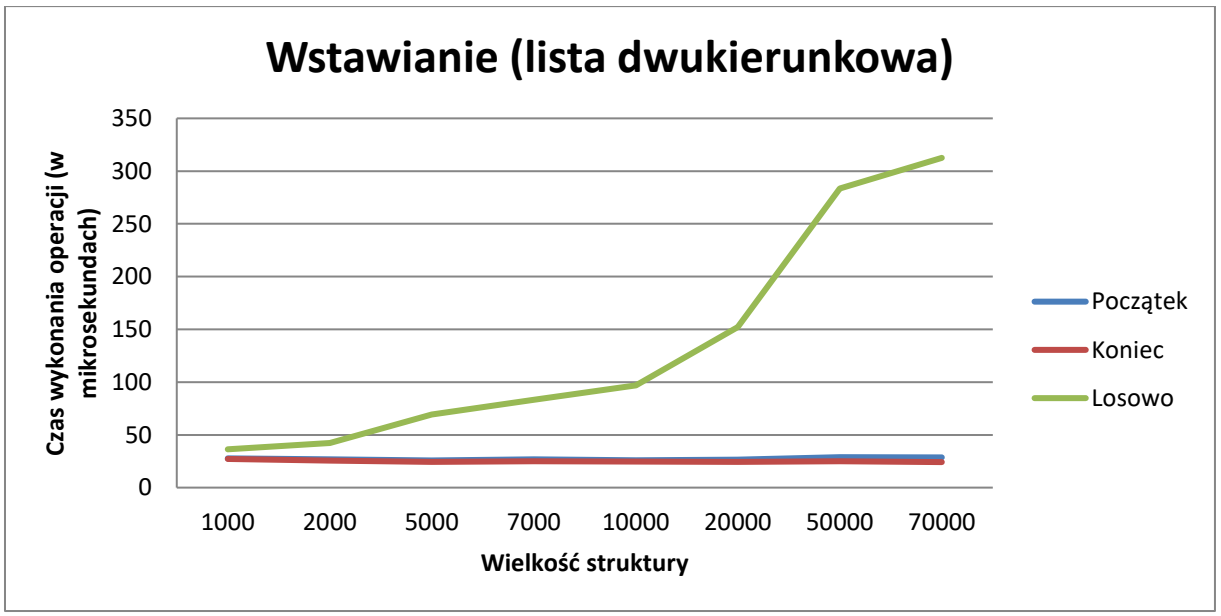
Tablica							
	Średni czas (μs)						
Wielkość struktury	Wstawianie			Usuwanie			Wyszukiwanie
	Początek	Koniec	Losowo	Początek	Koniec	Losowo	
1000	38,89	38,44	38,58	38,47	38,26	38,43	29,16
2000	50,01	48,87	49,74	48,81	49,65	33,74	33,74
5000	71,42	71,07	72,96	70,52	69,71	70,43	37,41
7000	88,77	87,45	88,01	87,24	86,16	88,68	43,68
10000	109,13	115,53	115,15	114,48	115,75	115,06	53,83
20000	171,21	188,52	186,16	185,99	187,59	186,73	61,39
50000	390,45	424,05	421,26	416,99	429,52	421,44	85,39
70000	698,77	577,11	572,71	570,45	582,02	575,76	89,53

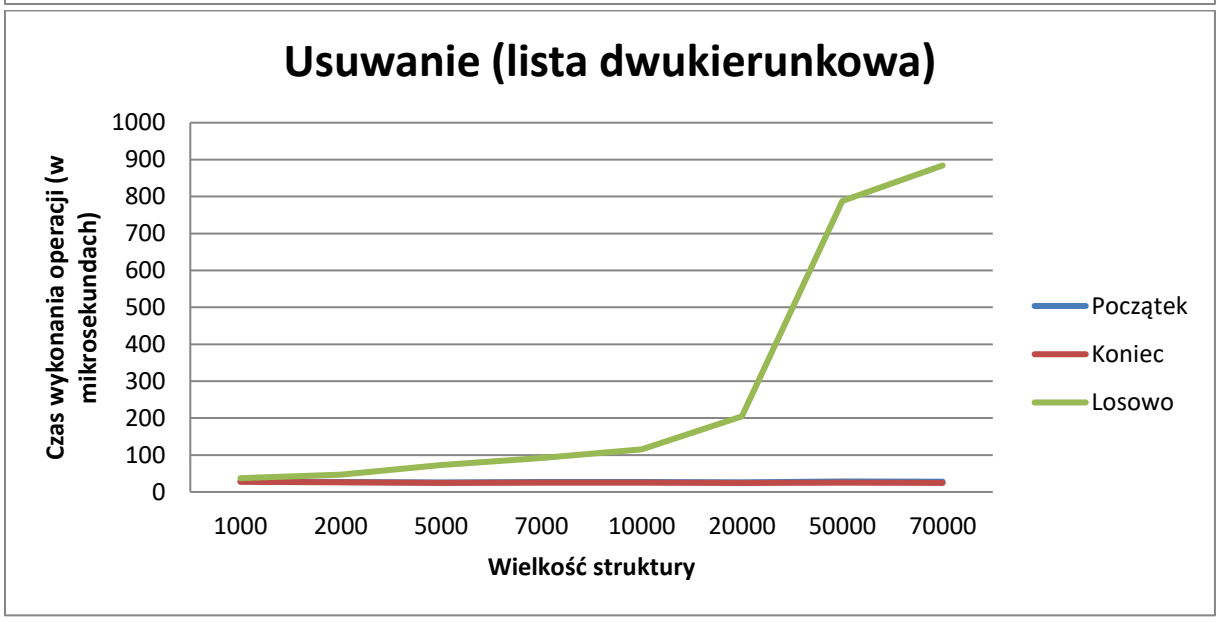

Wstawianie (tablica)
Czas wykonania operacji (w mikrosekundach)
800
700
600
500
400
300
200
100
0
1000
2000
5000
7000
10000
20000
50000
70000
Wielkość struktury
Początek
Koniec
Losowo

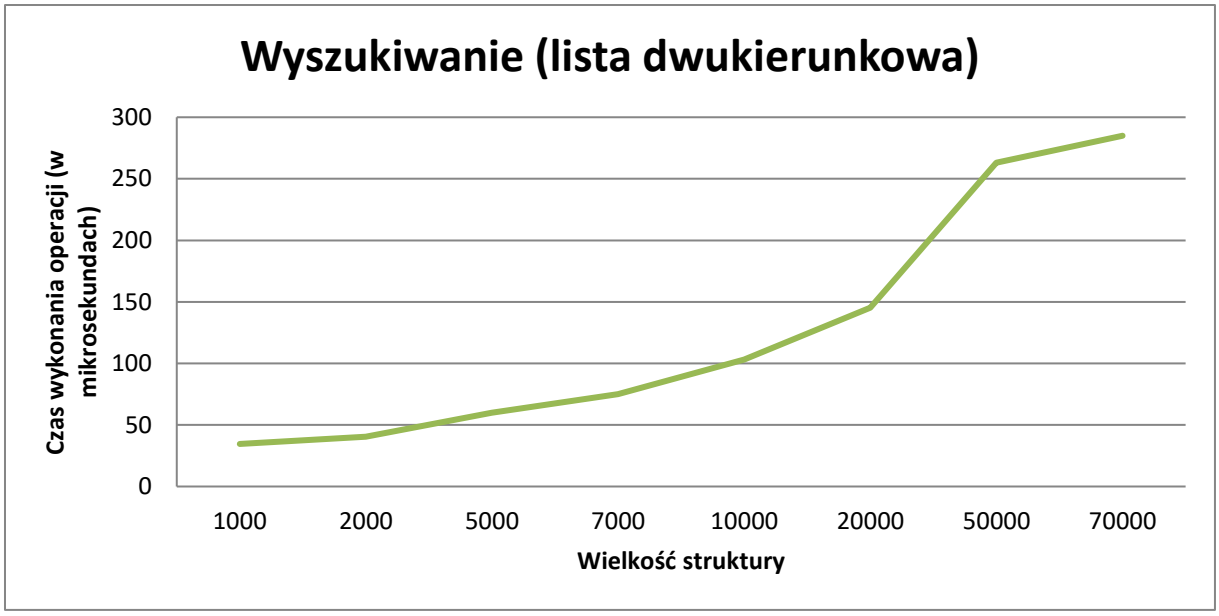

Usuwanie (tablica)
Czas wykonania operacji (w mikrosekundach)
700
600
500
400
300
200
100
0
1000
2000
5000
7000
10000
20000
50000
70000
Wielkość struktury
Początek
Koniec
Losowo


Wyszukiwanie (tablica)
Czas wykonania operacji (w mikrosekundach)
100
90
80
70
60
50
40
30
20
10
0
1000
2000
5000
7000
10000
20000
50000
70000
Wielkość struktury

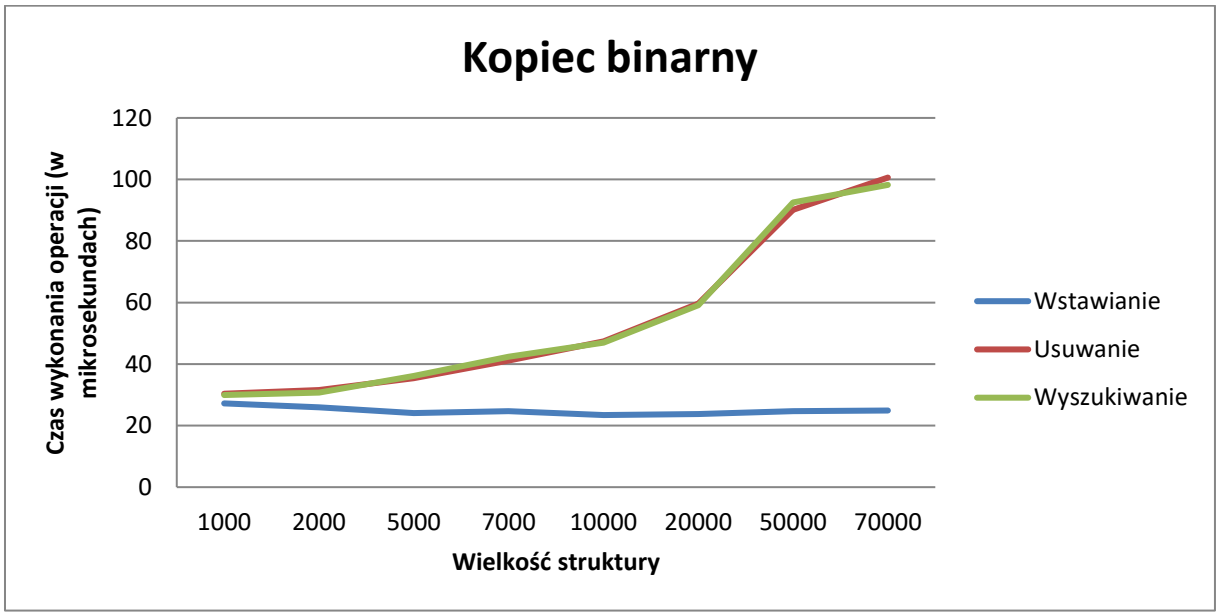
Lista dwukierunkowa							
	Średni czas (µs)						
Wielkość struktury	Wstawianie			Usuwanie			Wyszukiwanie
	Początek	Koniec	Losowo	Początek	Koniec	Losowo	
1000	27,76	27,17	36,33	28,24	27,52	37,36	34,58
2000	26,69	25,65	42,18	27,28	25,78	46,56	40,43
5000	25,74	24,27	69,38	25,81	24,47	72,55	59,98
7000	26,69	25,12	83,36	27,01	25,21	92,07	75,14
10000	25,77	24,54	96,87	27,36	24,89	115,38	103,06
20000	26,46	24,48	151,99	26,28	24,51	204,75	145,23
50000	28,87	24,89	283,52	28,36	25,24	788,27	263,09
70000	28,65	24,18	312,55	27,39	24,47	884,19	284,98



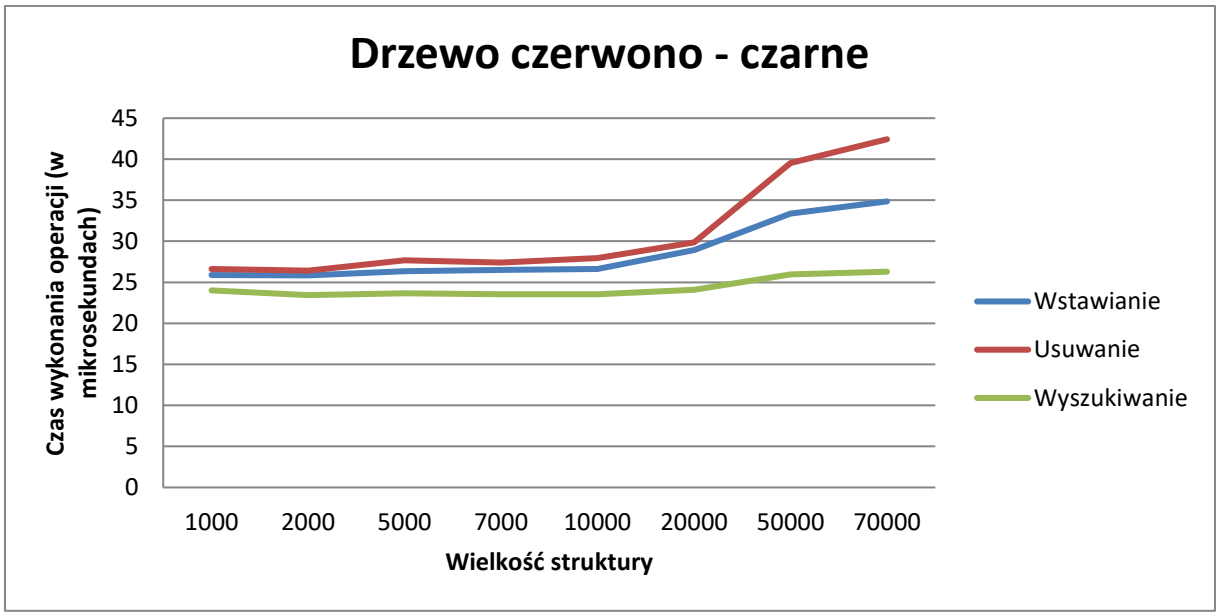




Kopiec binarny			
	Średni czas (µs)		
Wielkość struktury	Wstawianie	Usuwanie	Wyszukiwanie
1000	27,23	30,37	29,95
2000	25,98	31,54	30,71
5000	24,05	35,41	36,19
7000	24,71	41,14	42,37
10000	23,44	47,34	47,01
20000	23,75	59,66	59,11
50000	24,68	90,12	92,52
70000	24,87	100,61	98,24



Drzewo czerwono - czarne			
	Średni czas (µs)		
Wielkość struktury	Wstawianie	Usuwanie	Wyszukiwanie
1000	25,89	26,63	24,01
2000	25,84	26,42	23,44
5000	26,35	27,68	23,67
7000	26,52	27,41	23,56
10000	26,63	27,94	23,54
20000	28,91	29,85	24,08
50000	33,38	39,55	25,95
70000	34,86	42,43	26,29



5. Wnioski

Eksperyment pozwolił nam zaobserwować, jak przedstawiają się wyniki rzeczywistych pomiarów w porównaniu z wcześniejszymi założeniami co do ich złożoności. Powyższe wykresy ukazują, że teorie na temat złożoności są słuszne. Pewne nieścisłości, które pojawiły się w obliczeniach, wynikają prawdopodobne z używania komputera, którego procesor oprócz naszego eksperymentu ma jeszcze inne zadania do wykonania w tym samym czasie. Procesy działające w tle mogły wpływać na chwilowe spadki mocy obliczeniowej i tym samym spowodować wolniejsze wykonywanie operacji na strukturach. Innym powodem mógł być fakt, że program generował losowy zestaw danych, zatem liczby, na których wykonywał operacje, mogły być większe lub mniejsze, co również miało wpływ na pomiar czasu. Najprostsza w zaprojektowaniu oraz zaimplementowaniu była tablica dynamiczna, jednak modyfikacja danych wymaga dużo czasu i jest mało optymalna, szczególnie przy większej ilości danych. Lista dwukierunkowa okazała się bardziej optymalnym rozwiązaniem i tak, jak

się spodziewaliśmy, dostęp do pierwszego i ostatniego elementu był bardzo przydatny, gdyż czas operacji na tych elementach praktycznie nie zmieniał się niezależnie od rozmiaru struktury. Jedynie operacje związane z elementami w losowym miejscu w liście potrzebowały więcej czasu na realizację, gdy elementów było coraz więcej. Przy kopcu czas operacji wstawiania nowego elementu był bardzo podobny dla każdego rozmiaru struktury. Może to wynikać z faktu, że w rzeczywistości przy wstawianiu nowego elementu do kopca, nie potrzebujemy wiele napraw lub nie zajmują one tak dużo czasu. Czas potrzebny do usunięcia elementu był dłuższy, gdyż uwzględniał czas potrzebny na wyszukanie danego elementu. Najbardziej optymalnym rozwiązaniem okazało się drzewo czerwono – czarne, w którym czas potrzebny na wybrane operacje był bardzo podobny dla każdej wielkości struktury, jednak jego implementacja była czasochłonna i wymagająca. Należy również pamiętać, że w wykonywanych zadaniach nie została uwzględniona złożoność pamięciowa, która z pewnością pomogłaby w lepszym zobrazowaniu zalet i wad danych rozwiązań. Główny wniosek, który wyciągnęłam podczas realizacji projektu, jest następujący: Im bardziej wymagająca struktura pod względem implementacji, tym czas potrzebny na wykonywanie podstawowych operacji na jej elementach będzie krótszy.