Struktury Danych i Złożoność Obliczeniowa

Sprawozdanie z projektu 1

Autorka:

Joanna Zoglowek

264452

Temat:

Badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w różnych strukturach danych

Prowadzący kurs:

Dr inż. Antoni Sterna

Termin zajęć:

Wtorek 13:15 TN

Wstęp teoretyczny

Celem projektu było zaimplementowanie podstawowych struktur danych, a następnie pomiar czasu operacji na nich wykonanych. Do operacji należą: dodawanie, usuwanie, wyszukiwanie elementu (pierwszego, ostatniego, dowolnego). Dla kopca wystarczyło usuwanie elementu ze szczytu (korzenia). Elementem wszystkich struktur jest 4 bajtowa liczba całkowita ze znakiem. Testowane struktury:

- Tablica dynamiczna
- Lista dwukierunkowa
- Kopiec binarny
- Drzewo czerwono-czarne

Projekt został napisany i przetestowany w języku C++ bez wykorzystywania gotowych bibliotek takich jak STL, Boost.

1.1 Tablica dynamiczna

Tablica jest strukturą danych zawierającą klucze, z których każdy jest identyfikowalny przez konkretny indeks tablicy. W celu uzyskania dostępu do konkretnego klucza potrzebujemy jego indeksu oraz adresu początkowego tablicy.

Wstawianie elementu

• Początek tablicy

W celu wstawienia nowego elementu na początek tablicy tworzona jest nowa tablica o rozmiarze o 1 większym, w której wszystkie elementy zostają przesunięte o jeden indeks do przodu, a nowy element zostaje dodany na pozycję o indeksie 0.

Złożoność obliczeniowa takiej operacji zależy od wielkości tablicy, ponieważ wszystkie dane muszą zostać jednakowo przesunięte o jedną pozycję, czyli w zapisie za pomocą notacji dużego O będzie to O(n) – złożoność obliczeniowa liniowa.

• Środek tablicy

Również należy realokować całą tablicę, z taką różnicą że mniejszą liczbę danych należy przesunąć. Daje to krótszy czas wykonania programu.

Złożoność obliczeniowa jest taka sama jak w przypadku początku tablicy, a czas wykonywania jest krótszy.

• Koniec tablicy

Nie ma potrzeby przesuwania elementów tablicy, więc czas wykonania się skraca. Nadal potrzebna jest realokacja pamięci.

Złożoność obliczeniowa - pozostaje bez zmian w związku z niezmienną potrzebą realokacji całej tablicy, co zajmuje stosunkowo więcej czasu w zależności od ilości danych. Złożoność O(n).

- **Usuwanie**

• Początek tablicy

Usuwanie jest operacją podobną do wstawiania, tylko że przy usuwaniu z początku tablicy, realokujemy pamięć o mniejszym o jeden rozmiarze i przesuwamy wszystkie elementy o jeden indeks w dół z pominięciem indeksu 0.

Operacja ta ma złożoność czasową liniową ze względu na potrzebę przepisania wszystkich danych. Złożoność O(n).

• Środek tablicy

Różnica taka jak przy wstawianiu – mniej wartości do przesunięcia, co daje krótszy czas wykonywania. Złożoność bez zmian – nadal liniowa - O(n).

• Koniec tablicy

Czas wykonania krótszy, ponieważ nie trzeba przesuwać wartości, jednak czas realokacji pozostaje bez zmian. Złożoność – nadal liniowa – O(n).

▪ **Wyszukiwanie**

• Przeszukiwanie całej tablicy

Każdy kolejny klucz w tablicy porównywany jest z szukanym kluczem do momentu znalezienia szukanej wartości, czyli w najgorszym wypadku trzeba przeszukać całą tablicę.

Złożoność obliczeniowa w tym wypadku jest liniowa – im więcej elementów, tym więcej porównań – złożoność O(n)

1.2 Lista dwukierunkowa

Lista dwukierunkowa jest strukturą danych składającą się z sekwencyjnie połączonych ze sobą elementów. Każdy z elementów zawiera przechowywany klucz oraz dwa pola wskazujące na element poprzedni oraz następny na liście.

Wstawianie

• Początek listy

W przypadku wstawiania na początek listy, nowy element staje się pierwszym elementem na liście, a poprzedni pierwszy element zostaje kolejnym elementem. Operacja ta nie jest czasochłonna ze względu na to, że jedyne co trzeba zrobić to przypisać nowego poprzednika do elementu pierwszego, a do elementu nowego adres elementu poprzednio pierwszego.

Złożoność czasowa operacji jest stała, niezależna od ilości elementów, zawsze musimy wykonać operacje opisane powyżej. Złożoność O(1).

• Środek listy

W przypadku kiedy chcemy wstawić nowy element w środek listy, musimy najpierw odnaleźć element, który obecnie się tam znajduje, co jest bardziej czasochłonne im więcej elementów znajduję się w liście. Następnie tworzymy nowe połączenia między nowym elementem, a jego żądanym następnikiem i poprzednikiem. Złożoność w tym wypadku nie jest już stała, lecz liniowa. Złożoność O(n).

• Koniec listy

Sytuacja podobna jak przy wstawianiu na środek listy z tą różnicą, że należy dotrzeć na koniec listy, co zajmuje jeszcze więcej czasu. Złożoność taka sama jak powyżej – O(n).

Usuwanie

• Początek listy

Przy usuwaniu z początku listy, jedyne operacje jakie musimy wykonać to przemianowanie drugiego elementu listy na element pierwszy, poprzez ustawienie jego pola wskaźnika na poprzednika na adres pusty, oraz usunięcie poprzedniego pierwszego elementu. Operacja ta ma złożoność obliczeniową stałą, liczba elementów listy nie wpływa na szybkość jej wykonywania. Złożoność O(1).

• Środek listy

Tak samo jak w przypadku wstawiania elementu w środek listy tutaj również musimy odnaleźć element środkowy listy, co wraz z wzrostem elementów staje się coraz to bardziej czasochłonne. Po odnalezieniu żądanego elementu, tworzymy nowe powiązania między jego następnikiem i poprzednikiem, a następnie element ten usuwamy. Złożoność obliczeniowa wzrasta wraz ze wzrostem liczby elementów ze względu na potrzebę iteracji przez kolejne elementy listy. Złożoność O(n).

• Koniec listy

Przypadek podobny jak przy usuwaniu ze środka listy, tutaj również należy dotrzeć do elementu ostatniego poprzez iterację przez kolejne elementy listy, następnie element przedostatni staje się ostatnim, a poprzedni ostatni element usuwamy. Złożoność– O(n). ▪ Wyszukiwanie

Przeszukiwanie całej listy

Każdy kolejny klucz, przechowywany przez elementy w liście, porównywany jest z szukanym kluczem do momentu znalezienia szukanej wartości, czyli w najgorszym wypadku należy przeszukać całą tablicę, dopóki wartość klucza szukanego nie znajduje się na liście. Złożoność obliczeniowa w tym wypadku jest liniowa – im więcej elementów, tym więcej porównań – złożoność O(n).

1.3 Kopiec binarny

Kopiec jest strukturą danych przybierającą formę drzewiastą. W zależności od rodzaju kopca, minimalny lub maksymalny, odpowiednio każde dziecko jest większe lub mniejsze od swojego rodzica. W skutek tego, odpowiednio w korzeniu przechowywana jest wartość największa lub najmniejsza. Kopiec jest jednym ze sposobów implementacji kolejki priorytetowej. Wysokość drzewa h = log2(n),

gdzie n to ilość przechowywanych kluczy. Przy implementacji drzewa można wykorzystać tablicę lub listę.

Wstawianie

Operacja wstawiania nie jest rozdzielana na konkretne pozycje, wartość wstawiana jest zawsze na miejsce za ostatnim elementem kopca, a następnie krok po kroku nowa wartość jest przesuwana w górę drzewa, poprzez porównanie nowego elementu z jego rodzicem i ich zamianę miejscami, jeżeli własności kopca nie są zachowane.

W najgorszym przypadku, gdy nowa wartość jest odpowiednio największa lub najmniejsza w kopcu, zależnie od rodzaju kopca, złożoność obliczeniowa operacji wstawiania to O(log2(n)), ponieważ tyle operacji zamiany elementów trzeba wykonać.

Usuwanie

W przypadku usuwania również nie wyróżnia się usuwania z konkretnych pozycji, lecz jedynie usuwanie elementu odpowiednio największego lub najmniejszego, w zależności od rodzaju kopca. W celu usunięcia korzenia, zamienia się wartościami korzeń z elementem ostatnim w drzewie, usuwa się ostatni element, który teraz ma wartość korzenia, a następnie przesuwa się nowy korzeń w dół drzewa, poprzez porównywanie go z jednym z jego dzieci, do momentu kiedy własności kopca będą zachowane. Operacja ta ma złożoność czasową logarytmiczną – O(log2(n)), ponieważ po zamianie korzenia z elementem ostatni, zazwyczaj trzeba przesunąć nową wartość korzenia na sam dół drzewa, czyli wykonać log2(n) operacji zamiany.

Wyszukiwanie

Przeszukiwanie całego kopca

Każdy kolejny klucz przechowywany w kopcu porównywany jest z szukanym kluczem do momentu znalezienia szukanej wartości, czyli w najgorszym wypadku należy przeszukać cały kopiec, do momentu gdy wartość szukanego klucza nie znajduje się w kopcu.

Złożoność obliczenia jest liniowa – im więcej elementów tym więcej porównań – złożoność O(n).

1.4 Drzewo czerwono-czarne

Drzewo czerwono-czarne jest samoorganizującym się drzewem binarnym. W jego liściach nie są przechowywane dane. Puste wskazanie w polu syna węzła może być interpretowane jako liść. W celu zaoszczędzenia pamięci często wybiera się pojedynczy węzeł-strażnika, który pełni rolę wszystkich liści w drzewie. W takim przypadku węzły wewnętrzne w drzewie czerwono-czarnym w polach synów-liści przechowują wskazania do tego węzła strażnika. Puste drzewo zawiera jedynie węzeł strażnika. Drzewo czerwono-czarne gwarantuje, że jego wysokość nie przekroczy dwukrotnej wartości wysokości minimalnej. Aby to uzyskać, poszczególne węzły są kolorowane na czarno albo czerwono, przy zastosowaniu zasad:

- Odnośniki czerwone znajdują się po lewej stronie

- Żaden węzeł nie jest powiązany z dwoma odnośnikami czerwonymi
- Drzewo jest w pełni zbalansowane ze względu na czarne odnośniki – każda ścieżka z korzenia do pustego odnośnika obejmuje te samą liczbę czarnych odnośników

Dodawanie wartości

Aby dodać wartość do drzewa czerwono-czarnego, należy przejść przez kilka etapów.

1.Wstawianie wartości jako liść

Aby dodać wartość do drzewa czerwono-czarnego, należy najpierw wstawić ją jako liść. Nowy liść powinien mieć kolor czerwony, a jego rodzic powinien być czarny. Jeśli nowy liść zostanie wstawiony jako korzeń drzewa, to musi on być czarny.

2.Przywrócenie właściwości drzewa czerwono-czarnego

Wstawienie nowego liścia może spowodować naruszenie jednej z kilku właściwości drzewa czerwono-czarnego. Aby przywrócić właściwości drzewa czerwono-czarnego, należy wykonać następujące operacje:

- Jeśli rodzic nowego liścia jest czarny, to właściwości drzewa są zachowane, a dodanie nowego liścia jest ukończone.
- Jeśli rodzic nowego liścia jest czerwony, to musimy przywrócić właściwości drzewa czerwono-czarnego. Istnieją trzy przypadki, które musimy rozważyć:

1) Przypadek 1: Ojciec nowego liścia jest lewym synem swojego dziadka, a wujek nowego liścia (brat ojca) jest czerwony. W tym przypadku musimy zamienić kolor ojca i wujka na czarny, a dziadka na czerwony. Następnie przechodzimy do kolejnego etapu, którym jest sprawdzenie, czy właściwości drzewa są zachowane dla nowego dziadka.
2) Przypadek 2: Ojciec nowego liścia jest prawym synem swojego dziadka, a wujek nowego liścia (brat ojca) jest czerwony. W tym przypadku musimy wykonać takie same operacje jak w przypadku 1.
3) Przypadek 3: Ojciec nowego liścia jest lewym lub prawym synem swojego dziadka, a wujek nowego liścia (brat ojca) jest czarny. W tym przypadku musimy wykonać rotację wokół dziadka, aby ojciec nowego liścia stał się jego synem. Jeśli ojciec nowego liścia był lewym synem dziadka, to wykonujemy rotację prawostronną wokół dziadka. Jeśli ojciec nowego liścia był prawym synem dziadka, to wykonujemy rotację lewostronną wokół dziadka. Po wykonaniu rotacji musimy jeszcze zamienić kolory dziadka i ojca.

Złożoność obliczeniowa dodawania wartości do drzewa czerwono-czarnego wynosi O(log n), gdzie n to liczba węzłów w drzewie. W najgorszym przypadku, gdy drzewo jest niezrównoważone, tj. gdy jego

wysokość wynosi n, złożoność czasowa wstawiania wartości do drzewa czerwono-czarnego wynosi O(n), ale taki przypadek jest bardzo rzadki.

Dodawanie wartości do drzewa czerwono-czarnego jest operacją o bardzo dobrej złożoności obliczeniowej, co sprawia, że drzewo czerwono-czarne jest często wykorzystywane w implementacji struktur danych, takich jak słowniki, zbiory czy kolejki priorytetowe.

Usuwanie

Usuwanie elementu z drzewa czerwono-czarnego jest bardziej skomplikowane niż dodawanie elementu, ponieważ usuwanie może wpłynąć na zachowanie właściwości drzewa czerwono-czarnego. Istnieją trzy przypadki, w których może pojawić się problem podczas usuwania elementu:

- Usuwany węzeł ma co najwyżej jedno dziecko: w tym przypadku usuwamy węzeł i przenosimy jego jedyne dziecko do miejsca, w którym był usunięty węzeł.
- Usuwany węzeł ma dwoje dzieci: w tym przypadku usuwamy węzeł i zastępujemy go jego następnikiem, czyli najmniejszym węzłem w poddrzewie prawego syna usuniętego węzła.
- Usuwany węzeł jest korzeniem drzewa: w tym przypadku usuwamy węzeł i zastępujemy go nowym korzeniem, którym staje się lewy lub prawy syn usuwanego węzła.

Kroki przy usuwaniu węzła:

Krok 1: Jeśli usuwamy czerwony węzeł, to po prostu go usuwamy, ponieważ jego usunięcie nie wpłynie na pięć właściwości drzewa czerwono-czarnego.

Krok 2: Jeśli usuwamy czarny węzeł, to zastępujemy go węzłem, który spełnia warunki drzewa czerwono-czarnego. Może to wymagać wykonania rotacji lub zamiany kolorów.

Krok 3: Jeśli węzeł zastępujący usunięty węzeł jest czerwony, to po prostu zmieniamy jego kolor na czarny, aby spełnić pięć właściwości drzewa czerwono-czarnego.

Krok 4: Jeśli zastępujący węzeł jest czarny, to wykonujemy operacje naprawcze, aby przywrócić pięć właściwości drzewa czerwono-czarnego. W tym przypadku istnieją dwa podprzypadki:

a) Usuwany węzeł jest lewym synem swojego rodzica: wtedy musimy wykonać rotację w prawo wokół rodzica usuwanego węzła i kontynuować naprawę drzewa.
b) Usuwany węzeł jest prawym synem swojego rodzica: wtedy musimy wykonać rotację w lewo wokół rodzica usuwanego węzła i kontynuować naprawę drzewa

Krok 5: Po zakończeniu naprawy drzewa musimy upewnić się, że korzeń drzewa jest czarny, ponieważ naruszenie tej właściwości może prowadzić do innych problemów.

Złożoność obliczeniowa usuwania elementu z drzewa czerwono-czarnego zależy od wielu czynników, takich jak wysokość drzewa, liczba węzłów w drzewie, a także stopień skomplikowania operacji naprawczych. W pesymistycznym przypadku, kiedy musimy wykonać wiele operacji naprawczych, złożoność obliczeniowa usuwania elementu z drzewa czerwono-czarnego wynosi O(log n), gdzie n to liczba węzłów w drzewie. Jednak w najlepszym przypadku, kiedy nie musimy wykonywać żadnych operacji naprawczych, złożoność obliczeniowa może wynosić O(1), co oznacza, że usuwanie elementu jest bardzo szybkie.

Wyszukiwanie elementu

Aby wyszukać dowolny element w drzewie czerwono-czarnym, musimy przejść przez drzewo, zaczynając od korzenia, aż znajdziemy węzeł, który zawiera poszukiwany element lub dojdziemy do liścia, co oznacza, że szukany element nie istnieje w drzewie.

Procedura wyszukiwania elementu w drzewie czerwono-czarnym wygląda następująco:

1. Rozpocznij od korzenia drzewa i sprawdź, czy jego wartość jest równa poszukiwanemu elementowi. Jeśli tak, to zakończ i zwróć węzeł.
2. Jeśli wartość poszukiwanego elementu jest mniejsza niż wartość bieżącego węzła, kontynuuj poszukiwanie w lewym poddrzewie. Jeśli wartość poszukiwanego elementu jest większa, kontynuuj poszukiwanie w prawym poddrzewie.
3. W trakcie przeszukiwania drzewa, musimy również sprawdzać kolory węzłów, aby zapewnić, że drzewo spełnia pięć właściwości drzewa czerwono-czarnego.
4. Jeśli nie udało się znaleźć poszukiwanego elementu w drzewie, zwróć informację o tym, że szukany element nie istnieje w drzewie.

Złożoność obliczeniowa wyszukiwania elementu w drzewie czerwono-czarnym zależy od wysokości drzewa i wynosi O(log n), gdzie n to liczba węzłów w drzewie.

Złożoność obliczeniowa testowanych struktur danych zgodna z literaturą

OPERACJE NA TABLICY DYNAMICZNEJ	ZŁOŻONOŚĆ
Dodawanie na końcu	O(N)
Dodawanie na początku	O(N)
Dodawanie w dowolnym miejscu	O(N)
Usuwanie na końcu	O(N)
Usuwanie na początku	O(N)
Usuwanie w dowolnym miejscu	O(N)
Wyszukiwanie elementu	O(N)

OPERACJE NA LIŚCIE DWUKIERUNKOWEJ	ZŁOŻONOŚĆ
Dodawanie na końcu	O(1)
Dodawanie na początku	O(1)
Dodawanie w dowolnym miejscu	O(N)
Usuwanie na końcu	O(1)
Usuwanie na początku	O(1)
Usuwanie w dowolnym miejscu	O(N)
Wyszukiwanie elementu	O(N)

OPERACJE NA KOPCU BINARNYM	ZŁOŻONOŚĆ
Dodawanie	O(N)
Usuwanie korzenia	O(N)
Wyszukiwanie elementu	O(N)

OPERACJE NA DRZEWIE CZERWONO - CZARNYM	ZŁOŻONOŚĆ
Dodawanie	O(log(N))
Usuwanie	O(log(N))
Wyszukiwanie elementu	O(log(N))

Opis projektu

Badanie polegało na zmierzeniu czasu wykonania każdej z operacji dla kilku reprezentatywnych rozmiarów danej struktury i wielokrotnym powtórzeniu pomiarów (przynajmniej 10 razy lub więcej, za każdym razem generując nową losową populację), wyników uśrednianiu i zbadaniu zależności czasu wykonywania poszczególnych operacji od rozmiaru danej struktury (liczby przechowywanych elementów).

Czas wykonania operacji mierzony był przy użyciu biblioteki chrono przy pomocy funkcji std::chrono::high_resolution_clock.

Wyniki badań przedstawione zostały w postaci wykresów, które pozwoliły na porównanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w czterech różnych strukturach danych.

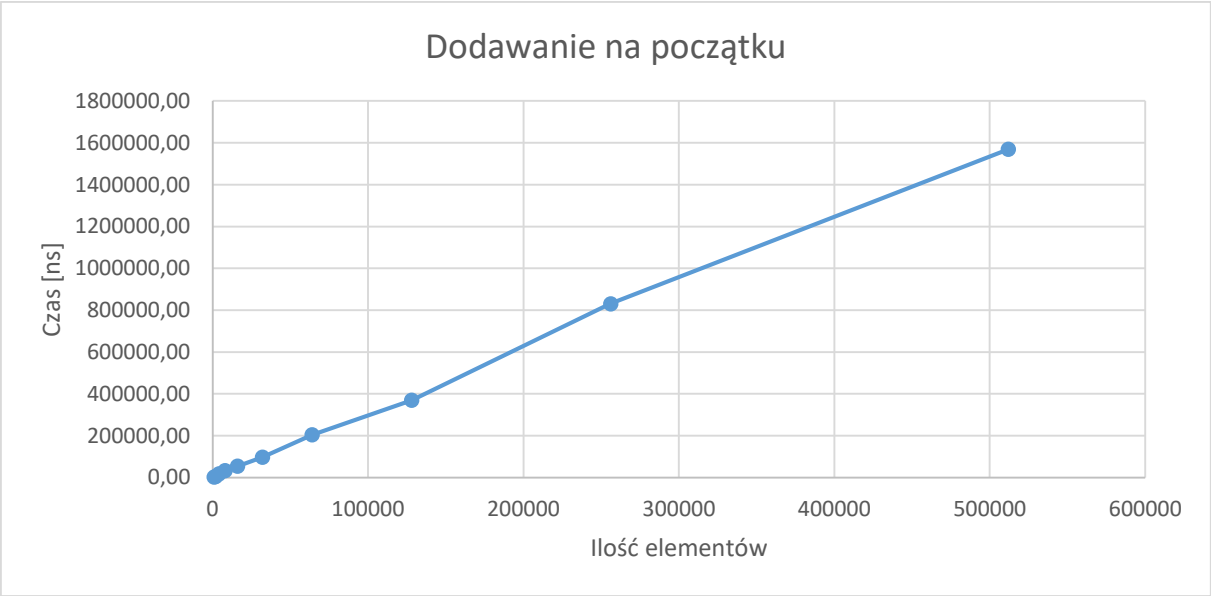
Pomiary wykonywałam 100 razy dla innych losowo wygenerowanych populacji, ponieważ tylko na tyle pozwolił mi komputer. Badanie prowadziłam dla potęgi liczby 2, od 1000 elementów do 512000.

Wyniki testów

1. Tablica dynamiczna

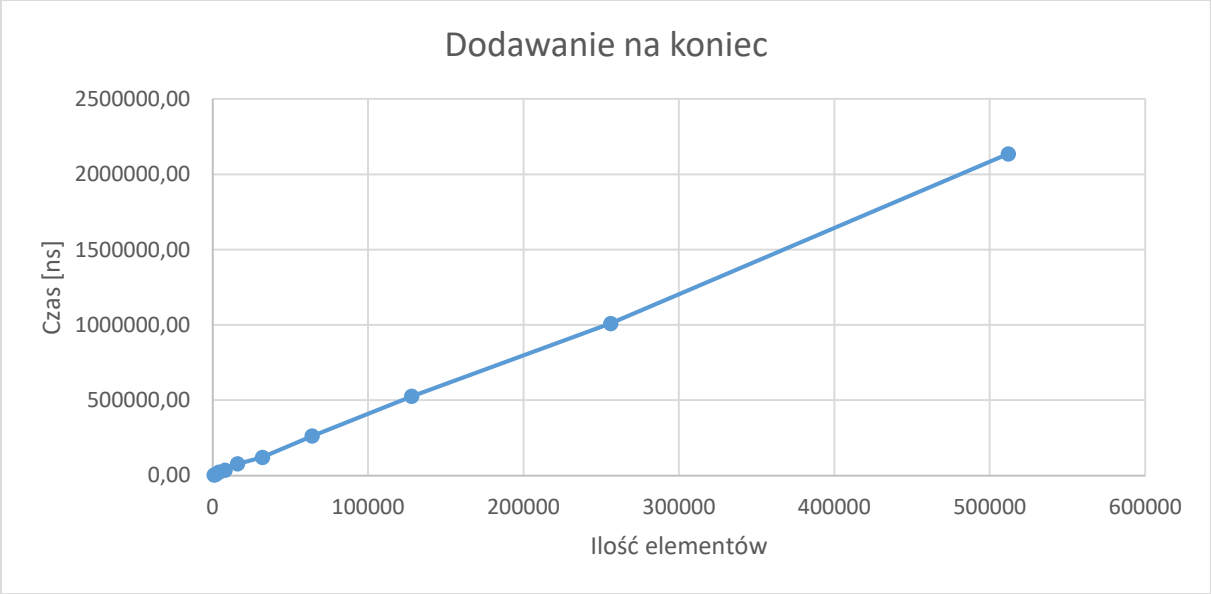
Ilość elementów	Dodawanie na początku [ns]	Dodawanie w losowe wybrane miejsce [ns]	Dodawanie na końcu [ns]	Usuwanie na początku [ns]	Usuwanie w losowym wybranym miejscu [ns]	Usuwanie na końcu [ns]	Wyszukiwanie losowego elementu [ns]
1000	3020,00	3600,00	3300,00	4450,00	6400,00	4250,00	13600,00
2000	6284,00	7750,00	8350,00	16025,00	16300,00	14975,00	19300,00
4000	17476,80	17925,00	21825,00	43862,50	44350,00	43287,50	23900,00
8000	32775,40	44962,50	36162,50	88831,20	88025,00	93893,80	26600,00
16000	54515,10	87981,20	77581,20	369166,00	179462,00	186897,00	35650,00
32000	97063,00	163391,00	120341,00	467733,00	406481,00	365848,00	41975,00
64000	204393,00	253895,00	261720,00	1170570,00	798741,00	744374,00	50288,00
128000	369799,00	496798,00	525260,00	2377280,00	2152120,00	2127990,00	60894,00
256000	831220,00	929049,00	1009580,00	4788190,00	4754660,00	4828890,00	101597,00
512000	1568980,00	1785820,00	2135640,00	9205700,00	9640580,00	9855700,00	131748,00

Wykresy:

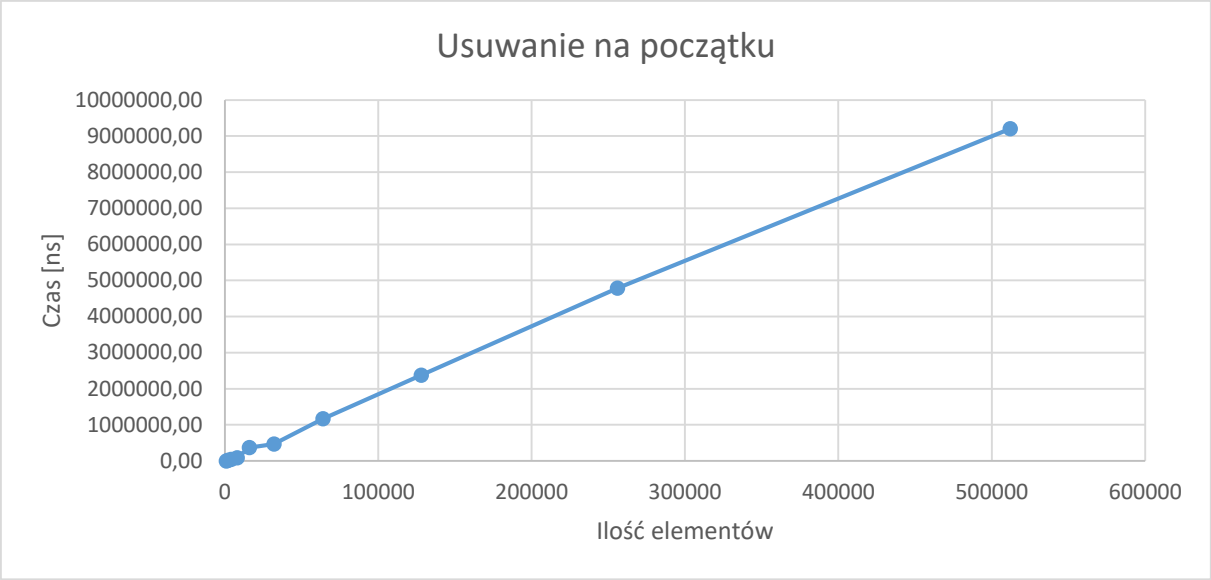


Wyniki przedstawione powyżej na wykresie dla wstawiania elementu na początek zgadzają się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze. Wartość wstawianego elementu ma niezauważalny wpływ na czas wykonania operacji.

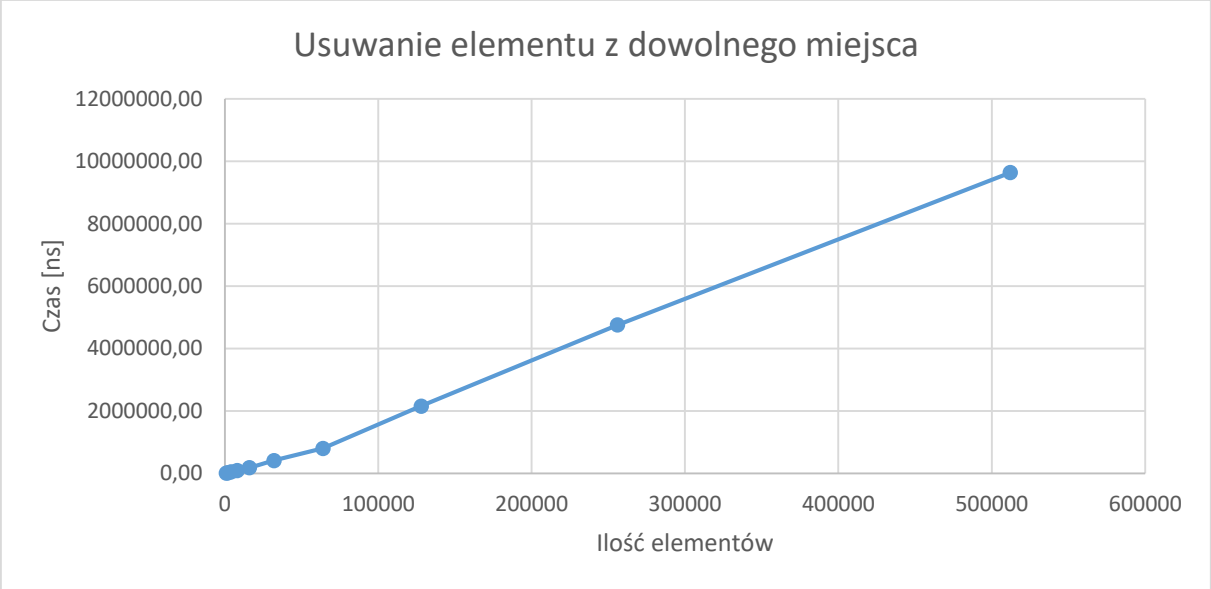
Wyniki przedstawione na wykresie zgadzają się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze. Czas wykonywania nie jest mniejszy niż w przypadku dodawania elementu na początek – brak zgodności z teorią.



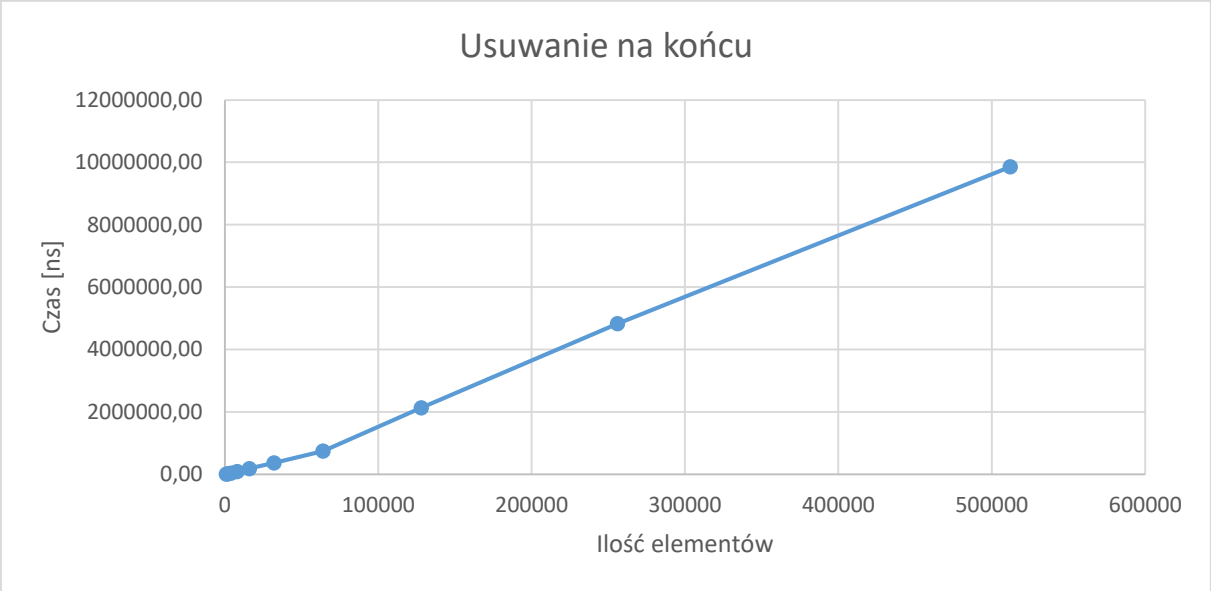
Wyniki przedstawione powyżej – dla wstawiania na koniec tablicy zgadzają się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze. Jednak czas wstawiania na koniec nie jest krótszy od wstawiania na początek oraz wstawiania na środek tablicy.



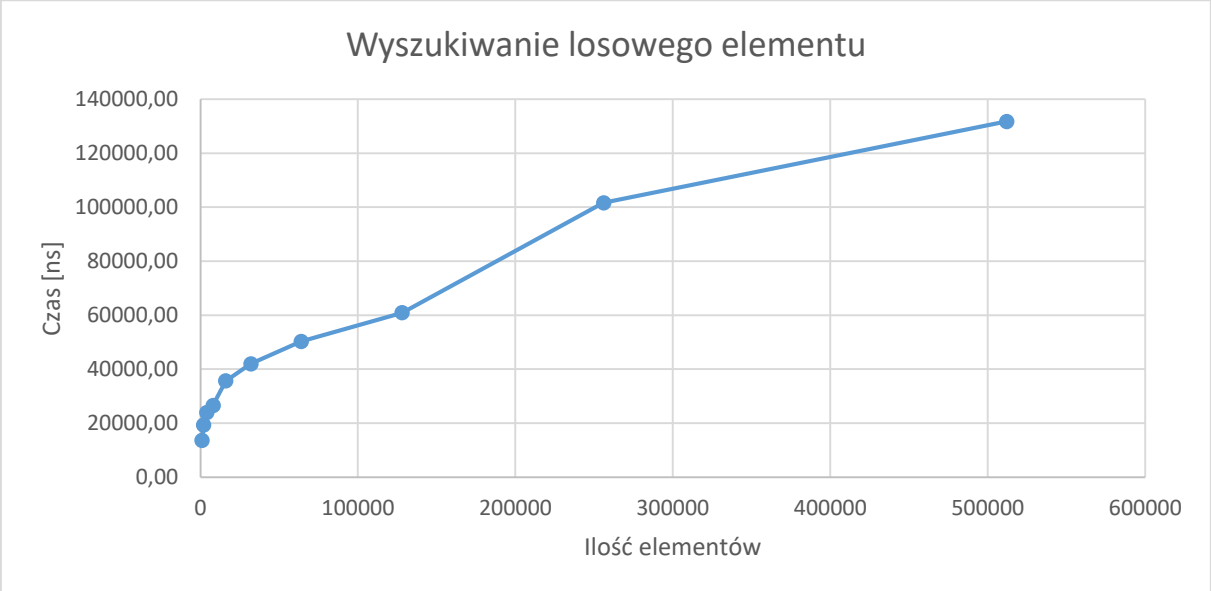
Wyniki przedstawione powyżej dla usuwania z początku tablicy układają się liniowo, co zgadza się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze.



Wyniki przedstawione na wykresie zgadzają się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze. Czas wykonywania jest mniejszy niż w przypadku usuwania elementu z początku, tylko dla połowy punktów pomiarowych – brak zgodności z teorią.



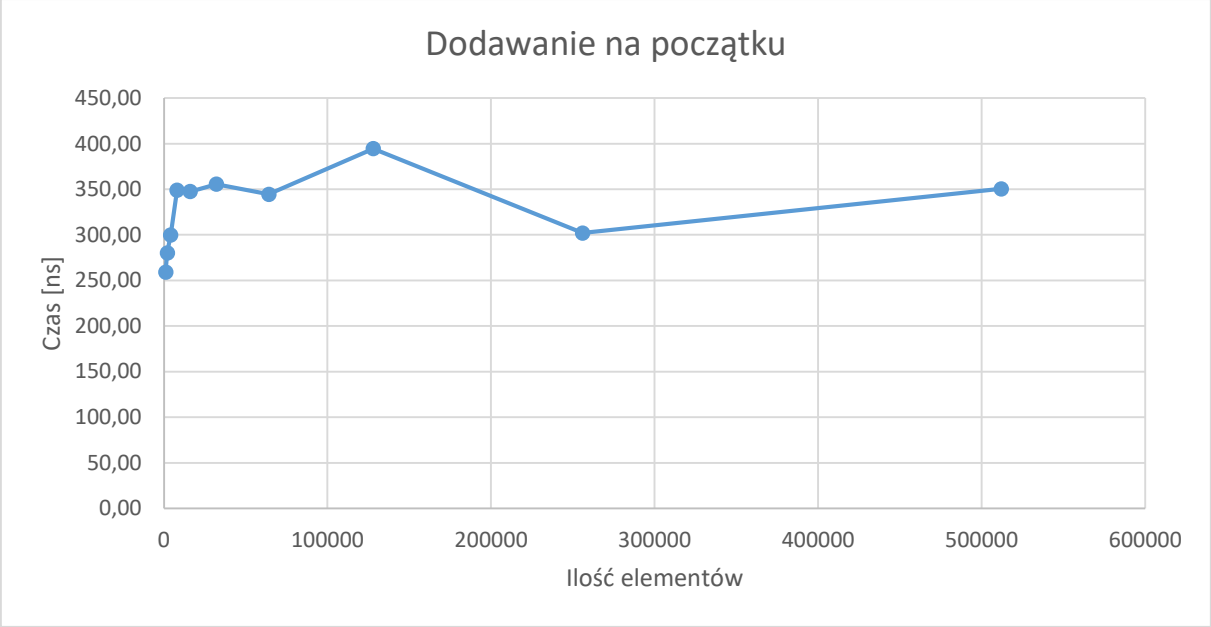
Wyniki przedstawione powyżej dla usuwania z końca tablicy zgadzają się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze. Zgodnie z założeniami w części teoretycznej czas usuwania z końca jest krótszy, dla niektórych wartości od czasu usuwania z początku oraz z środka tablicy. Brak zauważalnego wpływu wartości kluczy na czas wykonania operacji.



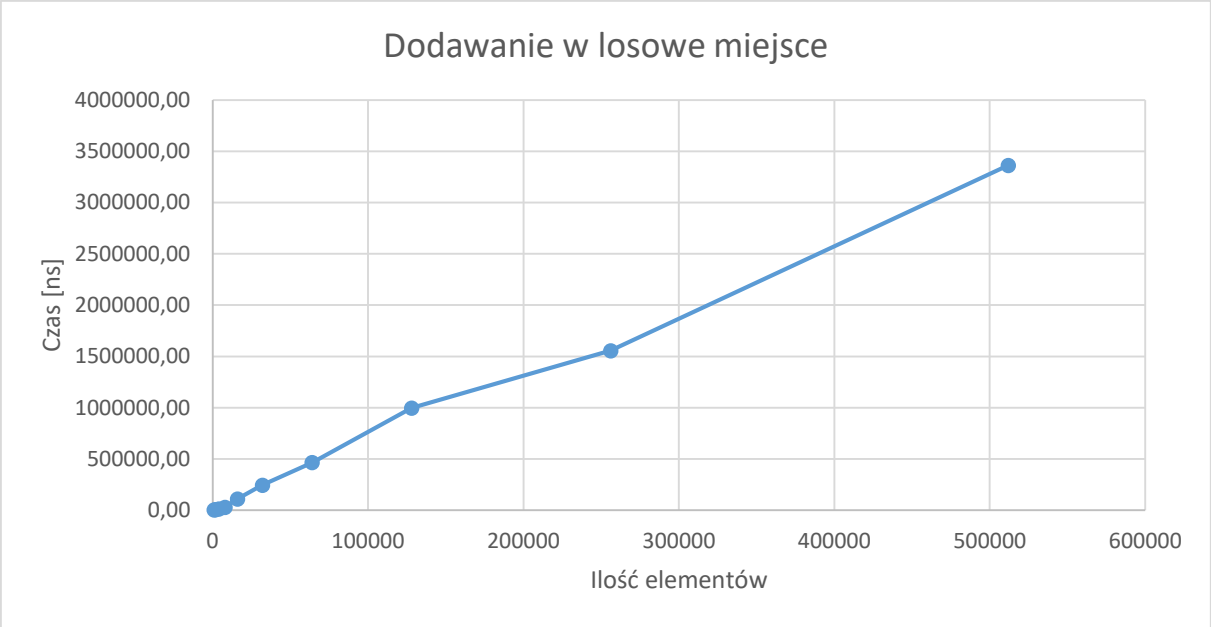
Wyniki przedstawione na wykresie są zbliżone do teorii, jednak nie jest to wykres liniowy.

2. Lista dwukierunkowa

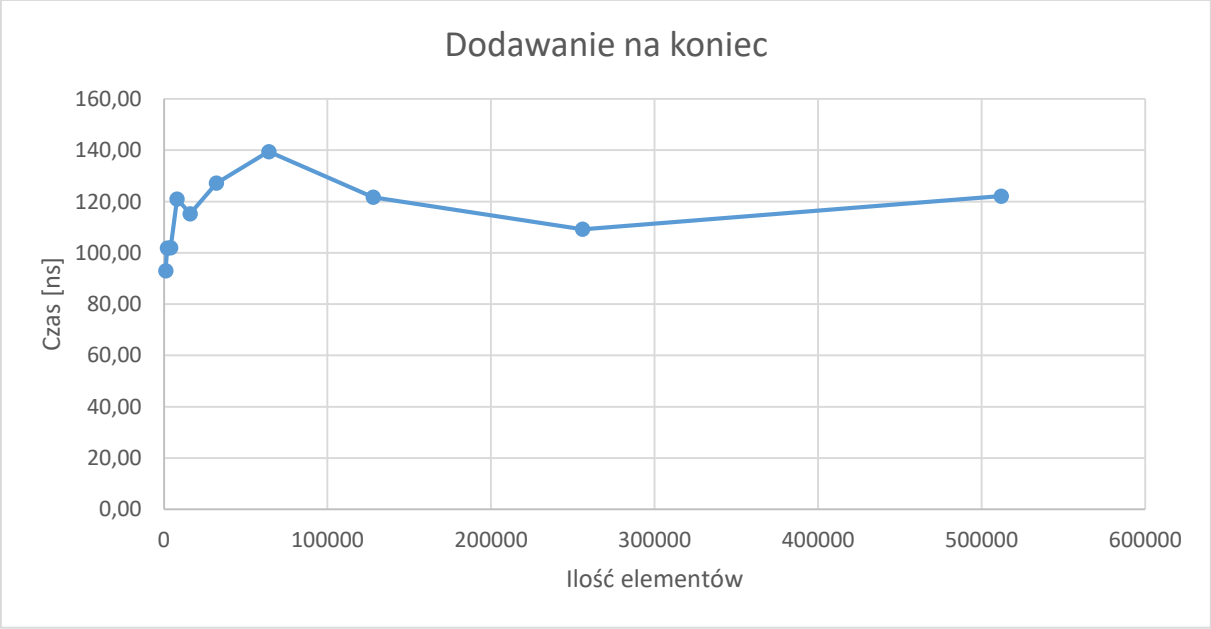
Ilość elementów	Dodawanie na początku [ns]	Dodawanie w losowe wybrane miejsce [ns]	Dodawanie na końcu [ns]	Usuwanie na początku [ns]	Usuwanie w losowym wybranym miejscu [ns]	Usuwanie na końcu [ns]	Wyszukiwanie losowego elementu [ns]
1000	259,00	3232,00	93,00	229,00	6903,00	108,65	782723
2000	280,29	6300,32	101,83	332,29	7681,03	120,09	985280
4000	299,80	14310,00	102,02	362,32	10854,80	131,80	1095870
8000	349,00	29292,10	121,02	323,62	13995,50	131,52	1363590
16000	347,49	110166,00	115,28	271,24	15934,00	86,32	1783280
32000	355,48	244241,00	127,15	293,71	17970,30	101,76	2158000
64000	344,56	464461,00	139,47	323,94	20753,70	125,02	2850500
128000	394,45	996455,00	121,70	337,24	25455,50	90,25	4022690
256000	301,99	1556040,00	109,22	311,37	32234,60	99,90	5077660
512000	350,46	3362940,00	122,09	365,11	44052,30	82,00	7058330



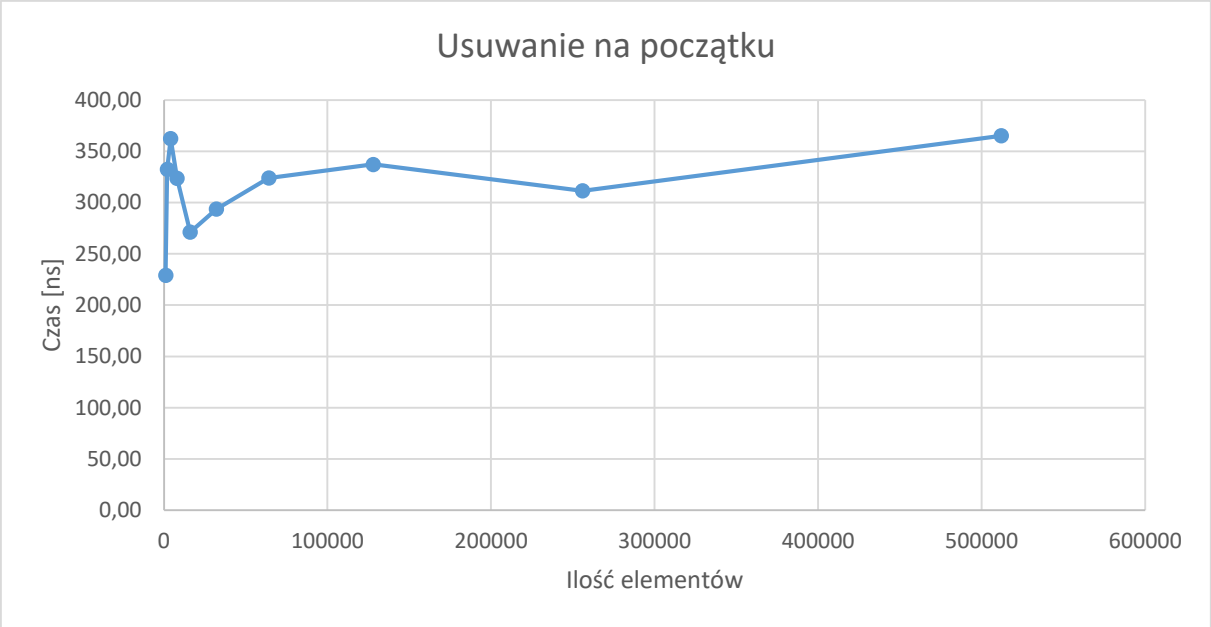
Wykres dodawania elementu na początek do listy dwukierunkowej nie zgadza się z teorią. Powinna być to funkcja prostej. Na błędny wynik mogą wpływać system operacyjny lub inny program, który przerywa operację. Czas zostaje mocno zachwiany.



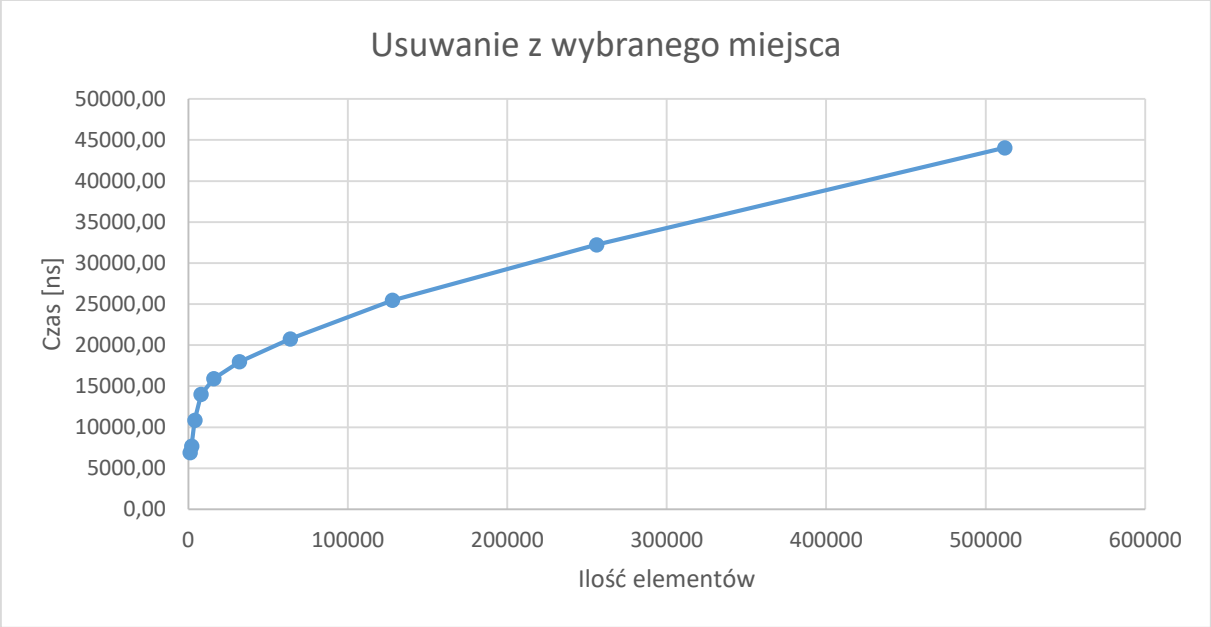
Wykres dodawanie elementu we wskazane miejsce zgadza się z teorią. Dodawanie w dowolne miejsce listy jest procesem, który jest bardziej czasochłonny od wstawiania na początek lub koniec, więc wpływ działania innych programów i systemu operacyjnego nie jest tak zauważalny.



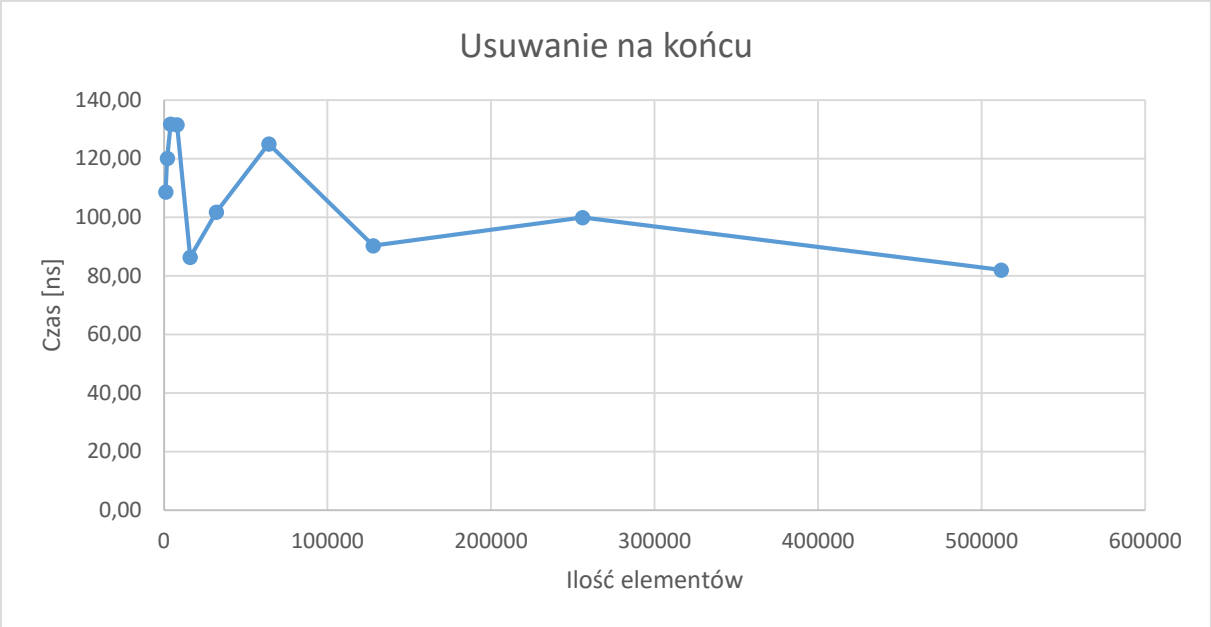
Podobna sytuacja jak przy dodawaniu elementu na początek listy dwukierunkowej. Wykres nie jest zgodny z teorią. Zakłócić operację obliczania czasu mógł system operacyjny i inne programy pracujące w tle.



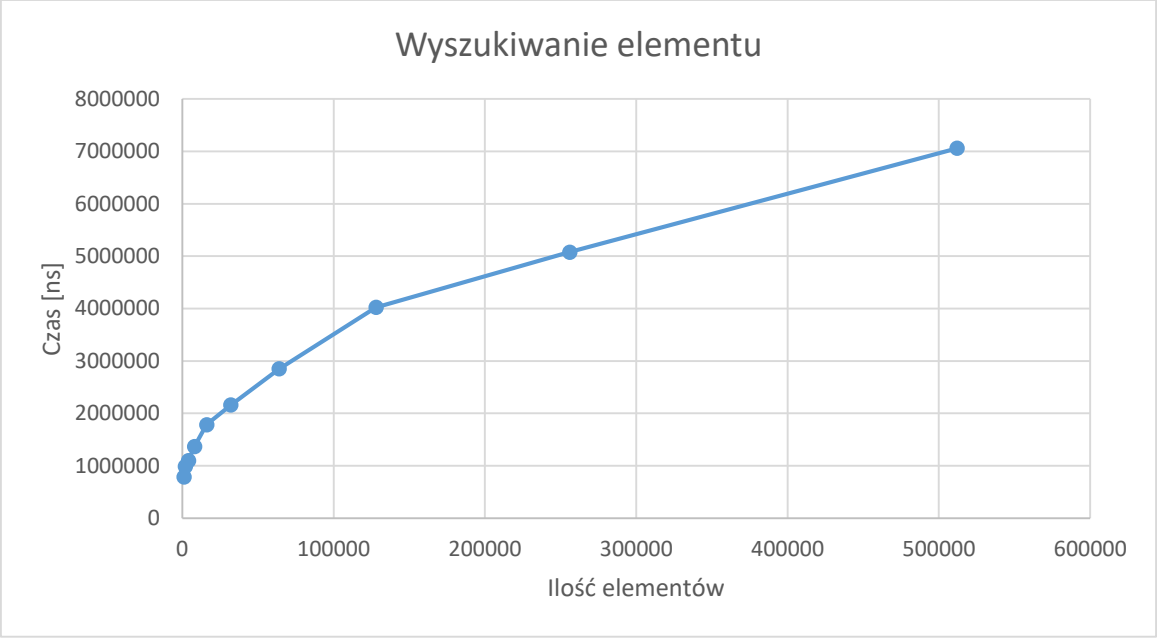
Wyniki przedstawione powyżej dla usuwania z początku listy dwukierunkowej nie zgadzają się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze. Złożoność obliczeniowa powinna być stałą funkcją.



Wyniki przedstawione powyżej dla usuwania z wybranego miejsca listy dwukierunkowej nie zgadzają się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze. Rozbieżność zachodzi w pierwszej części wykresy [1000, 64000].



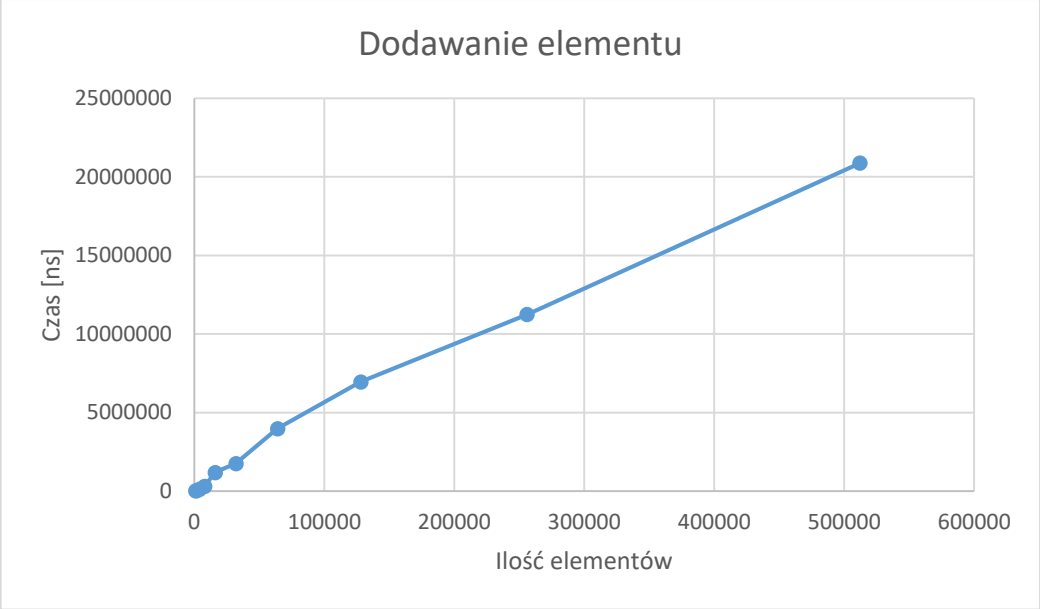
Wyniki przedstawione powyżej dla usuwania z końca listy dwukierunkowej nie zgadzają się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze. Złożoność obliczeniowa powinna być stałą funkcją.



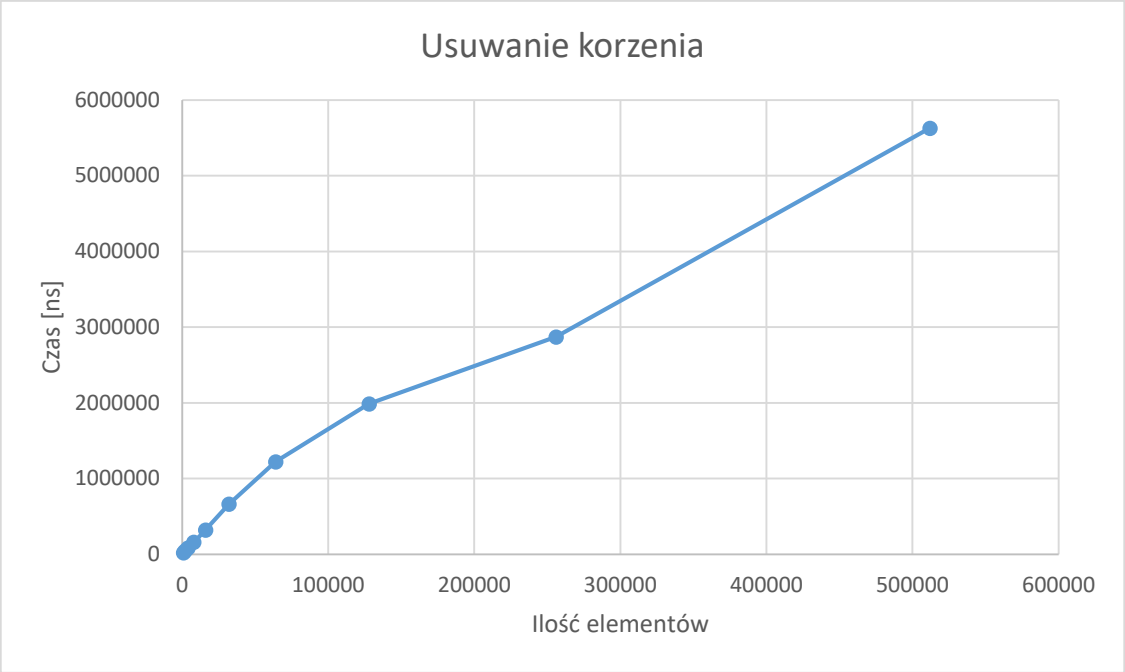
Wyniki przedstawione powyżej dla wyszukiwania dowolnego elementu listy dwukierunkowej zgadzają się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze.

3. Kopiec binarny

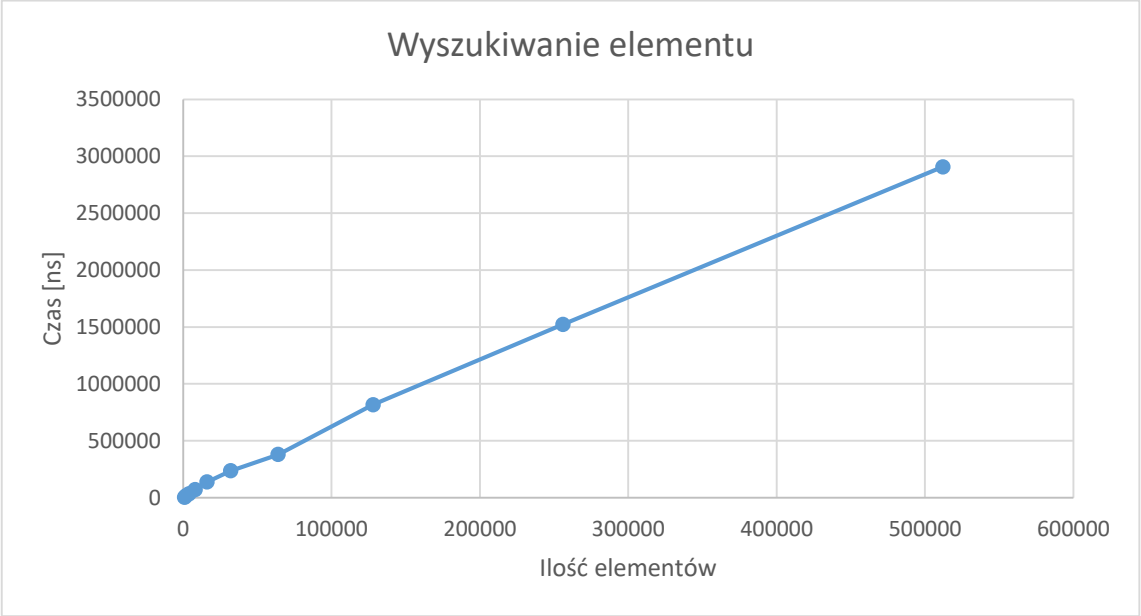
Ilość elementów	Dodawanie elementu	Usuwanie korzenia	Wyszukiwanie elementu
	[ns]	[ns]	[ns]
1000	12250	21150	3550
2000	47725	46225	17725
4000	133412	85327	34113
8000	314906	158223	70856
16000	1192250	321093	136628
32000	1756980	660924	235214
64000	3979440	1220787	379307
128000	6956170	1987663	816204
256000	11233490	2869932	1521750
512000	20872902	5627834	2906735



Wykres dodawania elementu do kopca binarnego zgadza się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze.



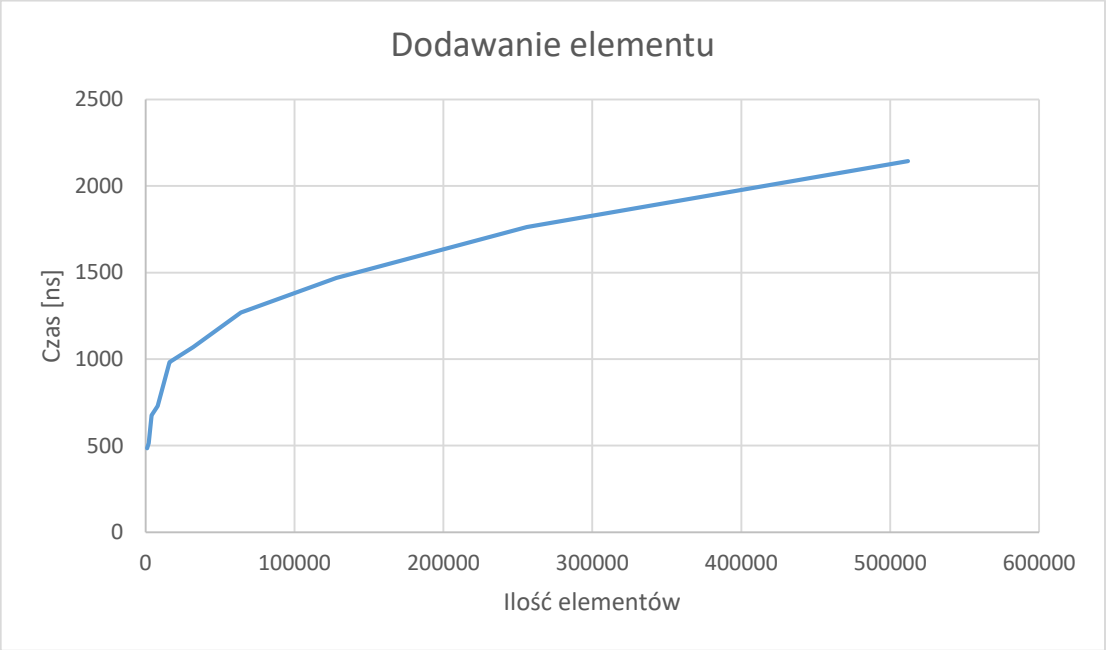
Wykres usuwania elementu z kopca binarnego zgadza się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze.



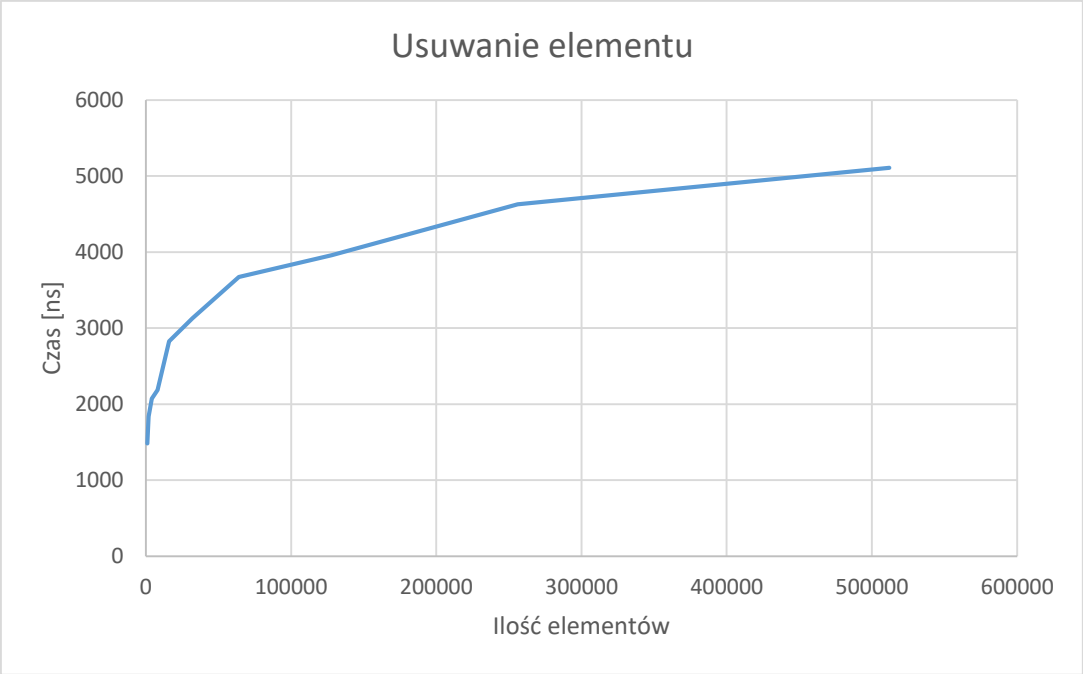
Wykres wyszukiwania dowolnego elementu, który znajduje się w kopcu binarnym zgadza się z teorią, która mówi o liniowej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze.

4. Drzewo czerwono-czarne

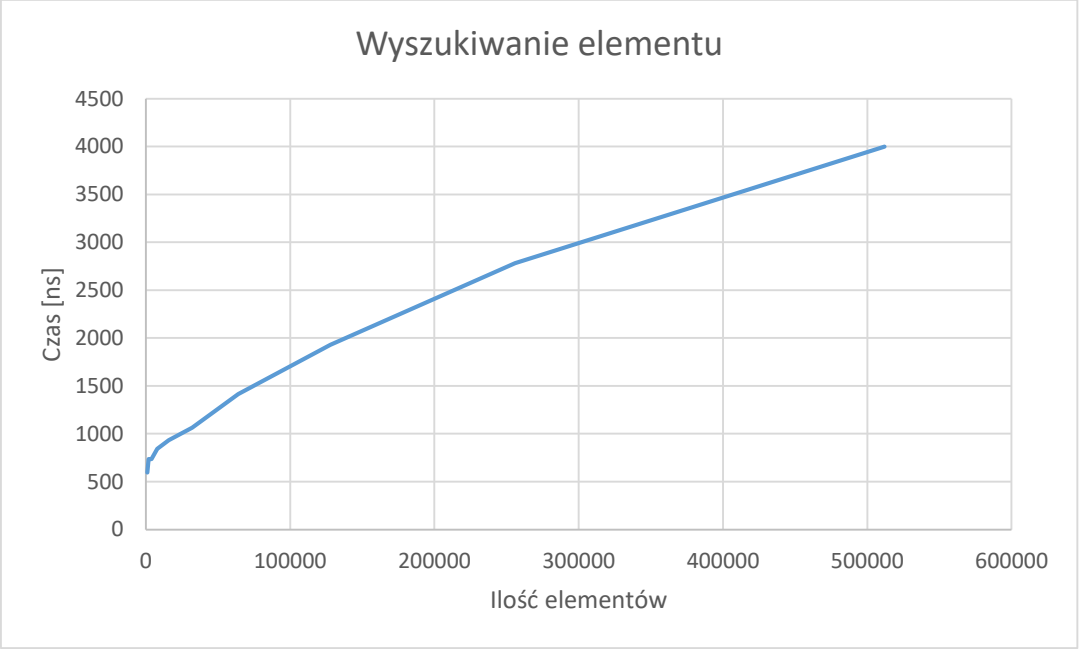
Ilość elementów	Dodawanie elementu	Usuwanie elementu	Wyszukiwanie elementu
	[ns]	[ns]	[ns]
1000	485	1485	595,32
2000	514,25	1839,25	734,766
4000	675,713	2071,96	736,738
8000	728,79	2188,6	841,837
16000	981,44	2824,43	932,092
32000	1069,07	3131,22	1061,605
64000	1268,45	3671,56	1413,08
128000	1468,42	3958,58	1930,654
256000	1763,42	4627,93	2781,533
512000	2143,17	5106,4	3999,077



Wykres dodawania elementu do drzewa czerwono-czarnego zgadza się z teorią, która mówi o logarytmicznej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze.



Wykres usuwania elementu z drzewa czerwono-czarnego zgadza się z teorią, która mówi o logarytmicznej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze.



Wykres wyszukiwania elementu w drzewie czerwono-czarnym lekko odbiega od teorii, która mówi o logarytmicznej zależności czasu wykonywania operacji od liczby elementów znajdujących się w strukturze.

Wnioski ogólne

Wszystkie struktury danych zostały przetestowane pod kątem operacji dodawania, usuwania i wyszukiwania elementów. Najmniej poprawna i zgodna z teoria okazała się tablica dwukierunkowa.
Najlepszą wydajność jednak wykazało drzewo czerwono-czarne, które wykazało się dość dobrymi wynikami podczas wszystkich operacji: wyszukiwania elementów, dodawania i usuwania.
Zależność czasu wykonywania operacji od rozmiaru danych różniła się między poszczególnymi strukturami. Dla tablicy dynamicznej, listy dwukierunkowej i kopca binarnego (przez implementację tablicową) zależność była typowo liniowa, natomiast dla drzewa czerwono-czarnego – logarytmiczna. Zgadzała się jednak ona ze sprawdzoną wcześniej zakładaną złożonością czasowa z niewielkimi odchyleniami.
Użyte w projekcie metody testowania oraz wielokrotne powtórzenie pomiarów z różnymi danymi pozwoliły na uzyskanie wiarygodnych wyników oraz uśrednienie czasu wykonania poszczególnych operacji.