Struktury danych i złożoność obliczeniowa
projekt 1

Porowski Wiktor 252719

April 2021

Spis treści

1 Wstęp

Celem projektu jest zaimplementowanie oraz dokonanie pomiaru czasu działania podstawowych operacji takich jak:
dodawanie, usuwanie, wyszukiwanie elementu itd. w następujących strukturach danych:
1. Tablica dynamiczna,
2. Lista dwukierunkowa,
3. Kopiec binarny,
4. Drzewo poszukiwań binarnych (BST)

2 Plan eksperymentu i założenia

Elementem wszystkich struktur jest 4 bajtowa liczba całkowita ze znakiem, wszystkie struktury danych są alokowane dynamicznie
i zajmują minimalną ilość miejsca.
Ponieważ wyniki niektórych operacji zależą od wartości elementów, pomiary dla konkretnego rozmiaru wykonywane są 10 razy,
za każdym razem generując nową losową populację, wynik jest
uśredniony.
Pomiary są wykonane dla 5 reprezentatywnych rozmiarów tak, aby odzwierciedlały one typ zależności (liniowy, logarytmiczny).

3 Oczekiwane złożoności czasowe wykonywania operacji

TABLICA DYNAMICZNA:
-wstawienie na koniec: O(n)

-wstawienie na początek: 0(n)
-wstawienie w dowolne miejsce: O(n)
-usunięcie ostatniego elementu: O(n)
-usunięcie pierwszego elementu: O(n)
-usunięcie elementu na dowolnej pozycji: O(n)
-dostęp do elementu poprzez index: O(1)
-wyszukanie elementu o zadanej wartości: O(n)

LISTA DWUKIERUNKOWA:
-wstawienie na koniec: O(1)
-wstawienie na początek: O(1)
-wstawienie w dowolne miejsce: O(n)
-usunięcie ostatniego elementu: O(1)
-usunięcie pierwszego elementu: O(1)
-usunięcie elementu na dowolnej pozycji: O(n)
-dostęp do elementu poprzez index: O(n)
-wyszukanie elementu o zadanej wartości: O(n)

KOPIEC (TYPU MAX)
-dostęp do elementu o maksymalnej wartości: O(1)
-pobranie i usunięcie elementu o maksymalnej wartości: O(log n)
-wstawienie elemetu o zadanej wartości: O(log n)
-znalezienie elementu o zadanej wartości: O(n*log n)

DRZEWO BST
-wstawianie elementu: O(h)
-usunięcie elementu: O(h)
-znaezienie elementu: O(h)
-znalezienie minimalnego elementu w drzewie: O(h)
-znalezienie maksymalnego elementu w drzewie: O(h)
-znalezienie następnika elementu: O(h)

O(h) - wyjaśnienie:
Większość operacji wykonywanych na drzewie BST ma złożoność czasową zależną od wysokośći drzewa.
Wysokość takiego drzewa w zależności od ilości n elementów w tym drzewie może być w niekorzystnym przypadku rzędu n.
(jedna gałąź - tzw drzewo zdegenerowane czyli w zasadzie drzewo przyjmuje postać listy).
Jednak w przypadku drzewa zrównoważonwgo "równomiernie rozgałęzionego" wysokość drzewa jest $log_2 n$.
Można udowodnić, że dla drzewa BST zbudowanego z losowych elementów średnio wysokość tego drzewa jest rzędu $log_2 n$.
(drzewo BST będie miało wysokość równą n gdy podamy dane w porządku).
Tak więc mówiąc o złożoności czasowej O(h) mówimy o złożoności w średnim przypadku O(log n)
w niekorzystnym przypadku O(n).

4 Zestawienie wyników pomiarów

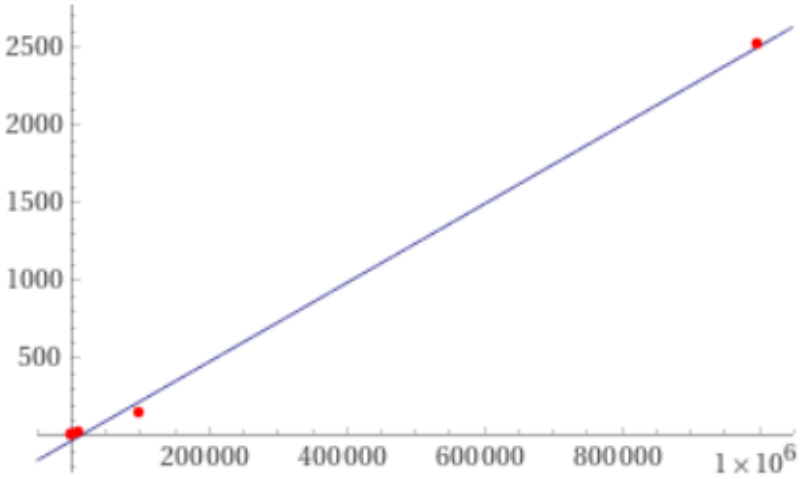
TABLICA DYNAMICZNA:

```
SIZE OF STRUCTURE =100
Time [us] = 1.62 -put first
Time [us] = 0.71 -put last
Time [us] = 1.5 -put by index
Time [us] = 0.63 -remove last
Time [us] = 0.86 -remove first
Time [us] = 0.78 -remove by index
Time [us] = 0.04 -get by index
Time [us] = 0.12 -find element by value

SIZE OF STRUCTURE =1000
Time [us] = 2.66 -put first
Time [us] = 1.96 -put last
Time [us] = 2.7 -put by index
Time [us] = 2.5 -remove last
Time [us] = 2.21 -remove first
Time [us] = 2.29 -remove by index
Time [us] = 0.11 -get by index
Time [us] = 0.91 -find element by value

SIZE OF STRUCTURE =10000
Time [us] = 20.19 -put first
Time [us] = 12.4 -put last
Time [us] = 14.15 -put by index
Time [us] = 17.91 -remove last
Time [us] = 13.9 -remove first
Time [us] = 14.62 -remove by index
Time [us] = 0.04 -get by index
Time [us] = 8.26 -find element by value

SIZE OF STRUCTURE =100000
Time [us] = 140.63 -put first
Time [us] = 121.14 -put last
Time [us] = 115.67 -put by index
Time [us] = 147.42 -remove last
Time [us] = 211.71 -remove first
Time [us] = 167.61 -remove by index
Time [us] = 0.08 -get by index
Time [us] = 76.17 -find element by value

SIZE OF STRUCTURE =1000000
Time [us] = 2515.57 -put first
Time [us] = 2202.52 -put last
Time [us] = 2360.36 -put by index
Time [us] = 2598.41 -remove last
Time [us] = 2465.98 -remove first
Time [us] = 2482.12 -remove by index
Time [us] = 0.06 -get by index
Time [us] = 501.14 -find element by value
```

Rysunek 1: Tablica dynamiczna czasy wykonywania operacji

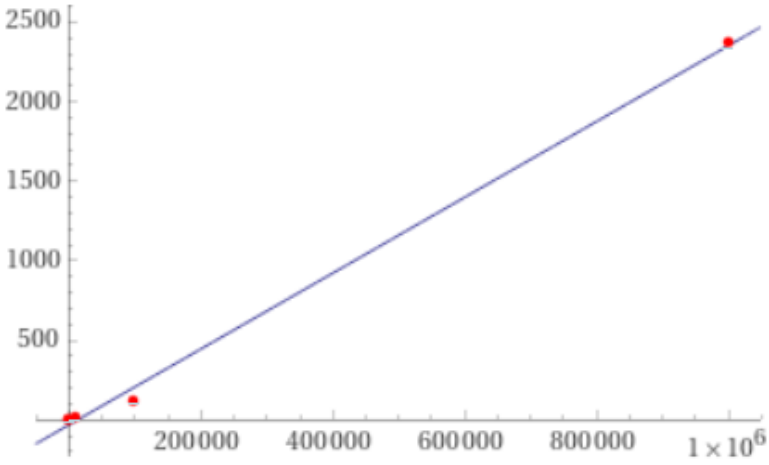
fit		
	data	{{100, 1.62}, {1000, 2.66}, {10 000, 20.19}, {100 000, 140.63}, {1 000 000, 2515.57}}
	model	linear function

Rysunek 2: put first operation



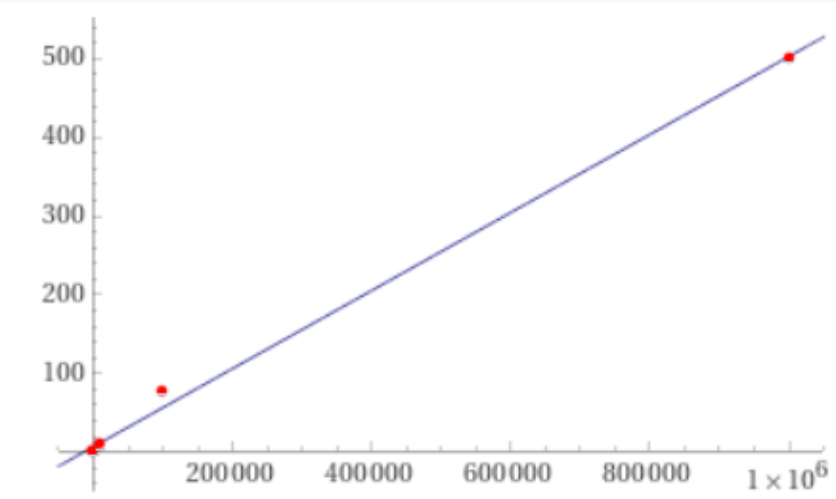
fit		
	data	{{100, 1.5}, {1000, 2.7}, {10 000, 14.15}, {100 000, 115.67}, {1 000 000, 2360.36}}
	model	linear function

Rysunek 3: put by index operation



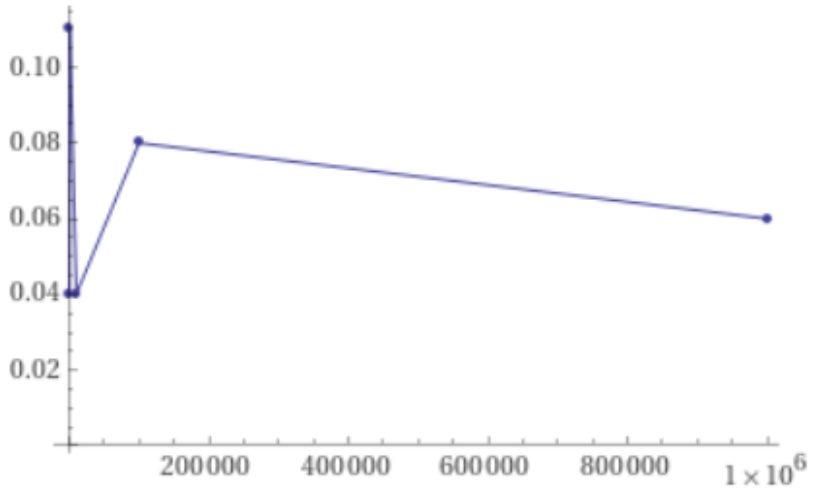
fit	data	{{100, 0.12}, {1000, 0.91}, {10000, 8.26}, {100000, 76.17}, {1000000, 501.14}}
	model	linear function

Rysunek 4: find operation



plot	{{100, 0.04}, {1000, 0.11}, {10000, 0.04}, {100000, 0.08}, {1000000, 0.06}}

Rysunek 5: get operation



Uwagi:
Pomiary odzwierciedlają przewidywania.
Operacja find ma charakter liniowy jednak z mniejszą stała od innych operacji o liniowej złożoności czasowej ponieważ niepotrzebne są operacje związane z realokacją.

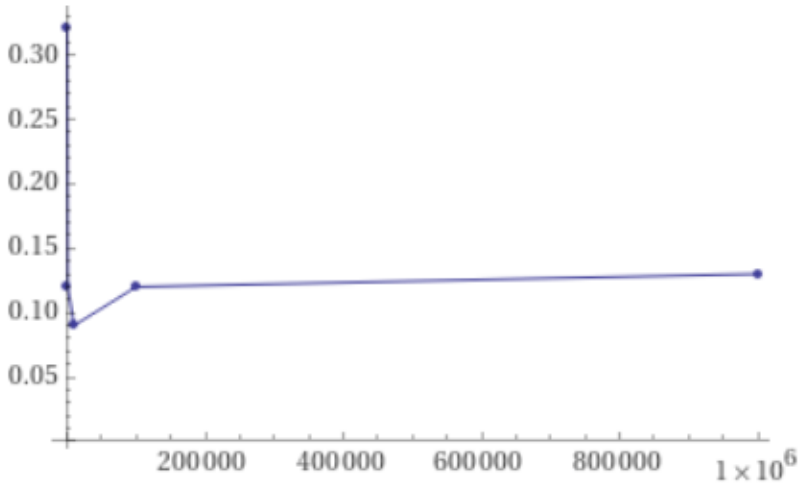
LISTA DWUKIERUNKOWA:

```
SIZE OF STRUCTURE =100
Time [us] = 0.32 -put first
Time [us] = 0.1 -put last
Time [us] = 0.78 -put by index
Time [us] = 0.14 -remove last
Time [us] = 0.09 -remove first
Time [us] = 0.26 -remove by index
Time [us] = 0.25 -get by index
Time [us] = 1.19 -find element by value

SIZE OF STRUCTURE =1000
Time [us] = 0.12 -put first
Time [us] = 0.1 -put last
Time [us] = 15.55 -put by index
Time [us] = 0.19 -remove last
Time [us] = 0.1 -remove first
Time [us] = 4.38 -remove by index
Time [us] = 39.4 -get by index
Time [us] = 33.34 -find element by value

SIZE OF STRUCTURE =10000
Time [us] = 0.09 -put first
Time [us] = 0.14 -put last
Time [us] = 124.53 -put by index
Time [us] = 0.33 -remove last
Time [us] = 0.09 -remove first
Time [us] = 53.85 -remove by index
Time [us] = 59.53 -get by index
Time [us] = 185.61 -find element by value

SIZE OF STRUCTURE =100000
Time [us] = 0.12 -put first
Time [us] = 0.12 -put last
Time [us] = 1222.9 -put by index
Time [us] = 0.63 -remove last
Time [us] = 0.13 -remove first
Time [us] = 795.67 -remove by index
Time [us] = 653.03 -get by index
Time [us] = 2214.84 -find element by value

SIZE OF STRUCTURE =1000000
Time [us] = 0.13 -put first
Time [us] = 0.1 -put last
Time [us] = 15806 -put by index
Time [us] = 0.78 -remove last
Time [us] = 0.11 -remove first
Time [us] = 8028.56 -remove by index
Time [us] = 8773.34 -get by index
Time [us] = 7642.44 -find element by value
```

Rysunek 6: Lista dwukierunkowa czasy wykonywania operacji

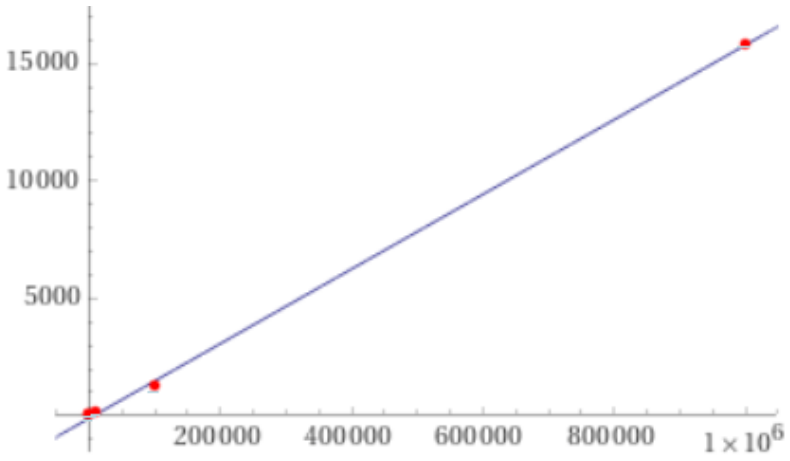
plot	{{100, 0.32}, {1000, 0.12}, {10000, 0.09}, {100000, 0.12}, {1000000, 0.13}}

Rysunek 7: put first operation



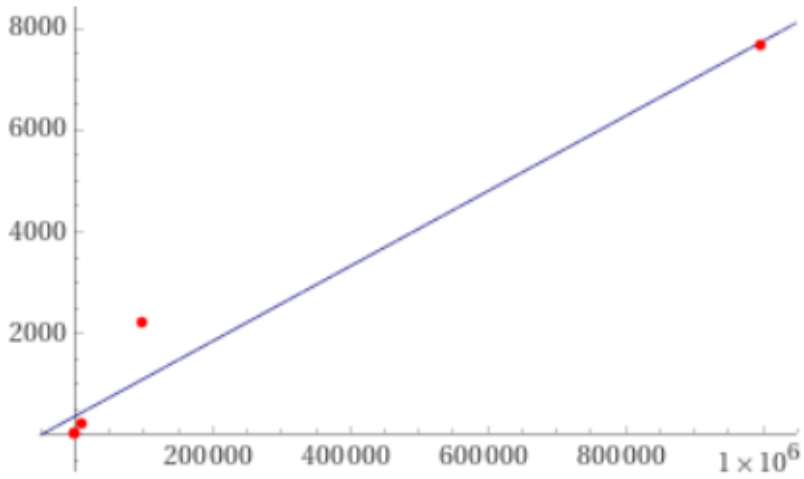
fit	data	{{100, 0.78}, {1000, 15.55}, {10000, 124.53}, {100000, 1222.9}, {1000000, 15806}}
	model	linear function

Rysunek 8: put by index operation



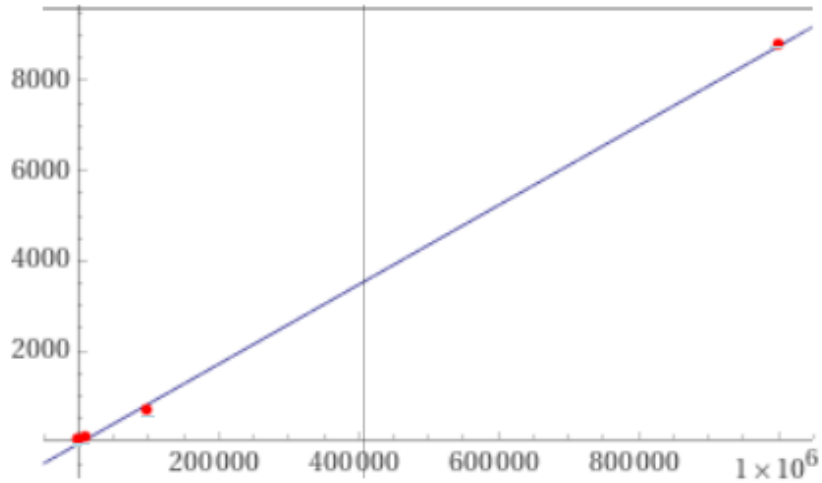
fit	data	{{100, 1.19}, {1000, 33.34}, {10000, 185.61}, {100000, 2214.84}, {1000000, 7642.44}}
	model	linear function

Rysunek 9: find operation



fit		
	data	{{100, 0.25}, {1000, 39.4}, {10000, 59.53}, {100000, 653.03}, {1000000, 8773.34}}
	model	linear function

Rysunek 10: get operation



Uwagi:
Pomiary odzwierciedlają przewidywania.
W operacji find prawdopodobieństwo znalezienia liczby z losowanego zakresu jest większe dla dużej listy z tąd może się brać nieco mniejszy od oczekiwanego rezultat pomiaru dla listy o rozmiarze 1000000.

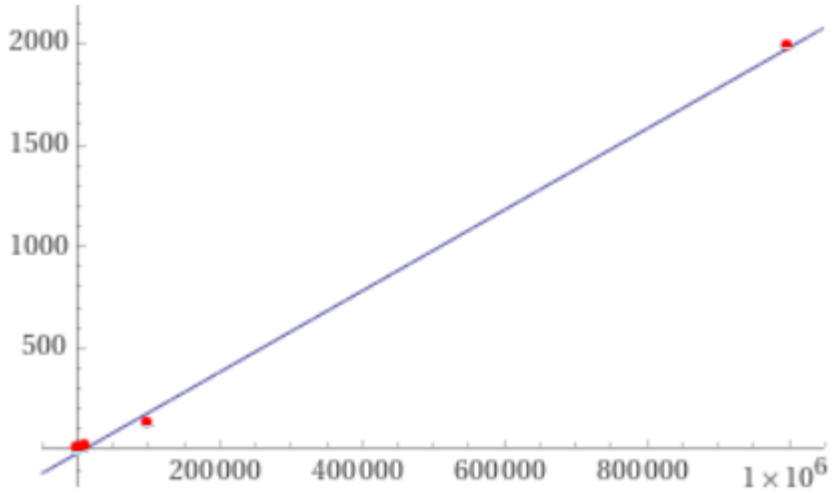
KOPIEC (TYPU MAX):

```
SIZE OF STRUCTURE =100
Time [us] = 1.36 -put
Time [us] = 0.83 -remove first
Time [us] = 0.03 -get max
Time [us] = 0.25 -find element by value

SIZE OF STRUCTURE =1000
Time [us] = 3.08 -put
Time [us] = 1.7 -remove first
Time [us] = 0.03 -get max
Time [us] = 1.08 -find element by value

SIZE OF STRUCTURE =10000
Time [us] = 12.52 -put
Time [us] = 9.79 -remove first
Time [us] = 0.07 -get max
Time [us] = 5.66 -find element by value

SIZE OF STRUCTURE =100000
Time [us] = 129.13 -put
Time [us] = 114.94 -remove first
Time [us] = 0.06 -get max
Time [us] = 51.09 -find element by value

SIZE OF STRUCTURE =1000000
Time [us] = 1983.61 -put
Time [us] = 1930.32 -remove first
Time [us] = 0.07 -get max
Time [us] = 549.05 -find element by value
```

Rysunek 11: Kopiec czasy wykonywania operacji

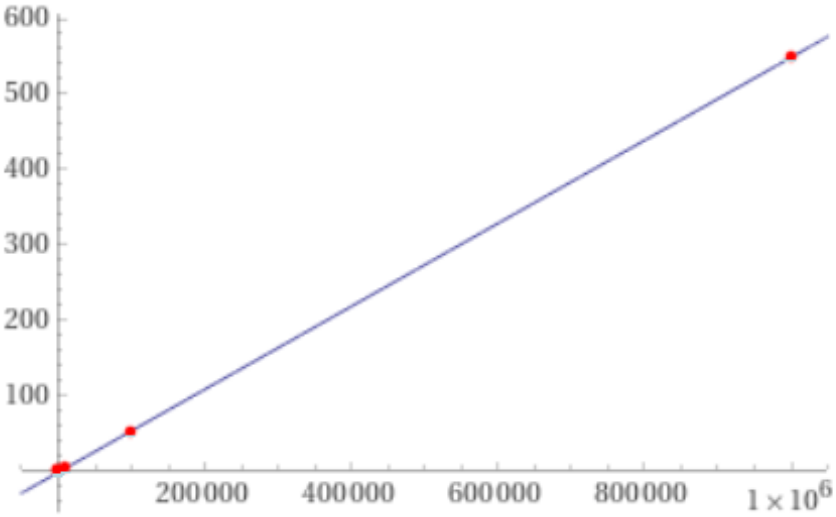
fit		
	data	{{100, 1.36}, {1000, 3.08}, {10000, 12.52}, {100000, 129.13}, {1000000, 1983.61}}
	model	linear function

Rysunek 12: put operation



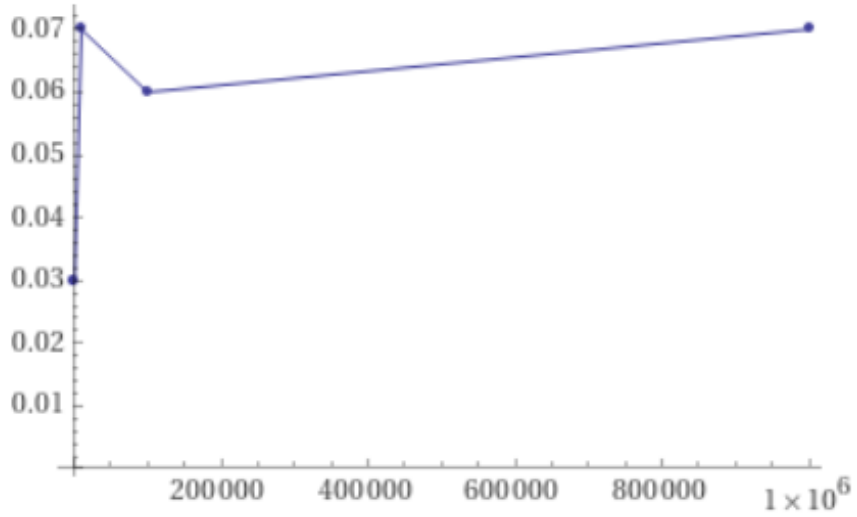
fit		
	data	{{100, 0.25}, {1000, 1.08}, {10000, 5.66}, {100000, 51.09}, {1000000, 549.05}}
	model	linear function

Rysunek 13: find operation



plot	{{100, 0.03}, {1000, 0.03}, {10000, 0.07}, {100000, 0.06}, {1000000, 0.07}}

Rysunek 14: get max operation



Uwagi:
Jak widać w praktyce operacje:
-pobranie i usunięcie elementu o maksymalnej wartości: O(log n)
-wstawienie elemetu o zadanej wartości: O(log n)
okazały się mięć charakter liniowy ze względu na implementacje kopca w tym wypadku z wykorzystaniem tablicy dynamicznej. Usuwanie oraz pobieranie elementu wiąże się z realokacją (jeżeli chcemy aby struktura zajmowała minimalny rozmiar) która jest operacją liniową tak więc nasza złożoność czasowa O(n + log n) czyli O(n).
Natomiast opisana w literaturze operacja:
-znalezienie elementu o zadanej wartości: O(n*log n)
W typ wypadku również ze względu na tablicową implementacje daję możliwość znalezienia elementu poprzez przejście po tablicy O(n).
Domniemane O(n*log n) wynika z tąd że traktując kopiec jako struktura z dostępem tylko do korzenia potrzebne by było wstawianie potencjalnie wszystkich elementów do korzenia -(n) i przywracanie własności kopca O(log n). Tak więc O(n*log n).

DRZEWO BST:

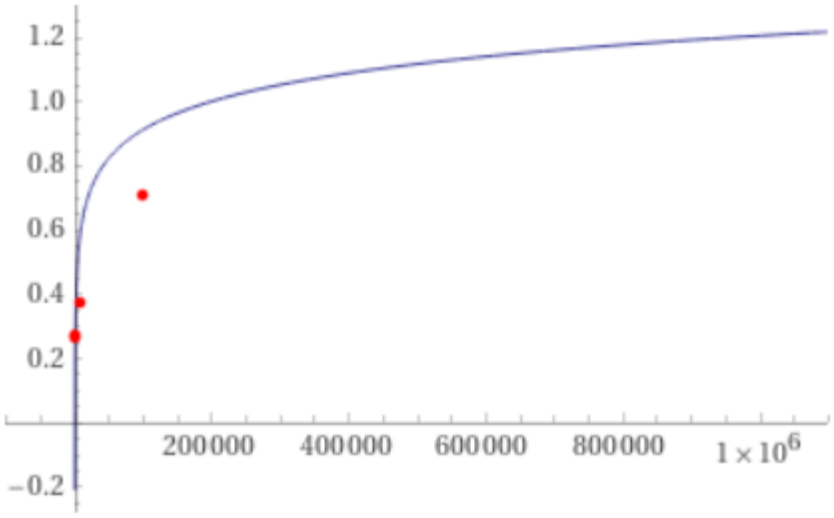
```
SIZE OF STRUCTURE =100
Time [us] = 0.26 -insert
Time [us] = 0.14 -find by value
Time [us] = 0.11 -remove when we have pointer to element
Time [us] = 0.38 -remove with finding element by value
Time [us] = 0.13 -find min in tree
Time [us] = 0.1 -find max in tree
Time [us] = 0.07 -finding succesor when we have pointer

SIZE OF STRUCTURE =1000
Time [us] = 0.27 -insert
Time [us] = 0.26 -find by value
Time [us] = 0.11 -remove when we have pointer to element
Time [us] = 0.31 -remove with finding element by value
Time [us] = 0.11 -find min in tree
Time [us] = 0.1 -find max in tree
Time [us] = 0.07 -finding succesor when we have pointer

SIZE OF STRUCTURE =10000
Time [us] = 0.37 -insert
Time [us] = 0.26 -find by value
Time [us] = 0.11 -remove when we have pointer to element
Time [us] = 0.43 -remove with finding element by value
Time [us] = 0.14 -find min in tree
Time [us] = 0.17 -find max in tree
Time [us] = 0.07 -finding succesor when we have pointer

SIZE OF STRUCTURE =100000
Time [us] = 0.71 -insert
Time [us] = 0.77 -find by value
Time [us] = 0.37 -remove when we have pointer to element
Time [us] = 0.91 -remove with finding element by value
Time [us] = 0.45 -find min in tree
Time [us] = 0.38 -find max in tree
Time [us] = 0.1 -finding succesor when we have pointer

SIZE OF STRUCTURE =1000000
Time [us] = 1.5 -insert
Time [us] = 1.2 -find by value
Time [us] = 0.46 -remove when we have pointer to element
Time [us] = 1.59 -remove with finding element by value
Time [us] = 0.78 -find min in tree
Time [us] = 0.43 -find max in tree
Time [us] = 0.54 -finding succesor when we have pointer
```

Rysunek 15: Drzewo BST czasy wykonywania operacji

fit

data	{{100, 0.26}, {1000, 0.27}, {10000, 0.37}, {100000, 0.71}, {1000000, 1.5}}
model	logarithmic

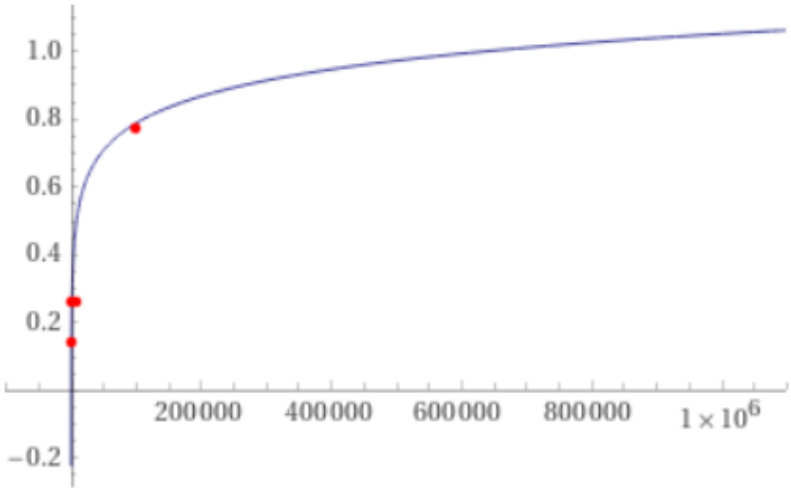
Rysunek 16: put operation



fit

data	{{100, 0.14}, {1000, 0.26}, {10000, 0.26}, {100000, 0.77}, {1000000, 1.2}}
model	logarithmic

Rysunek 17: find operation



Uwagi:
Widać że drzewo w średnim wypadku faktycznie okazuje się mieć wysokość log n ponieważ operacje wykonywane na tym drzewie mają złożoność czasową O(log n).

5 Wnioski

Lista oraz Tablica dynamiczna dają podobne możliwości,
jednak gdy zależy nam na częstym dostępie do elementów tablica dynamiczna
może okazać się lepszym wyborem.
Natomiast gdy często dodajemy/usuwamy elementy (i nie zależy nam na miejscu w liście w którym będą - domyślnie dodanie na koniec O(1)) i stosunkowo rzadko potrzebujemy dostęp do pojedyńczego elementu wtedy zdecydowanie lepsza będzie lista.

Kopiec świetnie sprawdza się jako kolejka priorytetowa.
Daje nam dostęp do największego elementu w czasie O(1)! (Dla listy i tablicy jest to O(n) - trzeba znaleźć max).
Gdy chcemy w kopcu usunąć największy element potrzebujemy O(log n) co również jest bardzo dobrym wynikiem.(W przypadku naszej implementacji O(n).)
Kopiec będzie bardzo dobrym wyborem gdy często potrzebny jest nam element o maksymalnym kluczu z elementów zbioru. (albo min dla kopca typu min)

Drzewo BST zapewnia nam wykonywanie operacji "słownikowych" w bardzo
korzystnym czasie. Średnia złożoność czasowa operacji: wstawiania, wyszukania, usunięcia. O(log n). Można więc wykorzystać tą strukture jako matematyczy zbiór.
Niebezpieczna jest sytuacja w której drzewo się zdegeneruje. Dlatego potrzebny jest algorytm równowarzący drzewo.
Gdy chcemy mieć zapewnione zawsze zrównoważone drzewo powinniśmy pomyśleć nad drzewem czerwono-czarnym.