

# Struktury Danych i Złożoność Obliczeniowa - Projekt: Sprawozdanie

Wydział Elektroniki	Kierunek: Informatyka Techniczna
Grupa zajęciowa Wt 15:15	Semestr: 2020/2021 Lato
Prowadzący:	Dr inż. Dariusz Banasiak

Autor
Byczko Maciej

## Wstęp

Zadanie projektowym było napisanie programu i zmierzenie czasu wykonywania działań na:

- Tablicy dynamicznej
- Liście dwukierunkowej
- Kopcu binarnym typu MAX
- drzewie czerwono-czarnym

Działania wykonywane na powyższych strukturach były następujące:

- Dodawanie
- Usuwanie
- Znajdowanie

## Założenia

- 4 bajtowa liczba całkowita ze znakiem (int w C++)
- wszystkie struktury danych powinny być alokowane dynamicznie
- należy zmierzyć czasy wykonywania poszczególnych operacji w funkcji rozmiaru danej struktury
- językami programowania są języki kompilowane do kodu natywnego (C, Objective C, C++, Rust, GO)
- nie wolno korzystać z gotowych bibliotek np. STL, Boost lub innych – wszystkie algorytmy i struktury muszą być zaimplementowane przez studenta
- realizacja zadania powinna być wykonana w formie jednego programu
- kod źródłowy powinien być komentowany
- program musi skompilowany do wersji exe

## Dodatkowe funkcje

- utworzenie struktury na podstawie danych zapisanych w pliku tekstowym. Pierwsza liczba określa rozmiar struktury, następnie należy wprowadzić odpowiednią liczbę danych np. każda liczba w osobnej linii
- wyświetlenie struktury na ekranie (w przypadku drzew zaproponować odpowiednią formę, która uwzględnia relacje między elementami tej struktury)
- możliwość wykonania wszystkich przewidzianych operacji na danej strukturze (wybór operacji najlepiej zrealizować w formie menu)

## Złożoności obliczeniowe

### Ogólne informacje

Złożoność obliczeniowa jest nam potrzebna aby określić ilość zasobów potrzebnych do rozwiązania problemu obliczeniowego. Rozważanymi zasobami są głównie:

- Czas (Czasowa złożoność obliczeniowa)- ilość czasu potrzebna do wykonania algorytmu
- Pamięć (Pamięciowa złożoność obliczeniowa)- ilość pamięci wykorzystanej w celu realizacji algorytmu

### Złożoność tablicy dynamicznej

Funkcja	Średnia	Pesymistyczna
Dodawanie	$O(n)$	$O(n)$
Usuwanie	$O(n)$	$O(n)$
Znajdowanie	$O(n)$	$O(n)$

### Złożoność listy dwukierunkowej

Funkcja	Średnia	Pesymistyczna
Dodawanie	$O(-)$	$O(n)$
Usuwanie	$O(-)$	$O(n)$
Znajdowanie	$O(n)$	$O(n)$

### Złożoność kopca binarnego

Funkcja	Średnia	Pesymistyczna
Dodawanie	$O(1)$	$O(1)$
Usuwanie	$O(1)$	$O(1)$
Znajdowanie	$O(n)$	$O(n)$

### Złożoność drzewa czerwono-czarnego

Funkcja	Średnia	Pesymistyczna
Dodawanie	$O(\log(n))$	$O(\log(n))$
Usuwanie	$O(\log(n))$	$O(\log(n))$
Znajdowanie	$O(\log(n))$	$O(\log(n))$

## Plan eksperymentu

### Informacje ogólne

- Pomiar czasu podczas dodawania elementów od 1000 do 45000
- Funkcja mierząca czas: `std::chrono::high_resolution_clock`
- Sposób generacji struktur:
  - Tworzenie struktury z losowymi danymi o podanym rozmiarze
  - Wykonanie operacji mierzonej
  - Zapisanie wyniku do pliku
  - Powtórzanie operacji (wartość zadana przez użytkownika)

### Pomiary czasowe

Pomiar czasowe były mierzone w **nanosekundach** ( $1[ns] = 1 * 10^9[s]$ ) za pomocą następującej funkcji:

```
template<typename T>
double Timer(T i) {
    auto start = chrono::high_resolution_clock::now(); // Start the counter
    i(); // our function
    auto end = chrono::high_resolution_clock::now(); // Get value after executing function
    auto duration = end - start; // get time difference
    auto elapsed_time = chrono::duration_cast<chrono::nanoseconds> (duration).count(); // calculate time
    return elapsed_time; // Return executing time in nanoseconds
}
```

## Wyniki wykonanych eksperymentów

### Pomiary tablicy dynamicznej

#### Wyniki pomiarów tablicy

Liczba danych	Dodawanie na początek	Dodawanie na koniec	Dodawanie gdziekolwiek	Usuwanie początek	Usuwanie koniec	Usuwanie gdziekolwiek	Szukanie
L.p. j	[ns]	[ns]	[ns]	[ns]	[ns]	[ns]	[ns]
1 5000	14200	13400	208300	10200	92700	175800	6200
2 10000	24800	21000	219700	18300	103700	185800	12100
3 15000	33400	28100	231200	27000	109600	196600	18900
4 20000	41100	37200	240300	34700	118300	203400	26600
5 25000	46900	44700	251800	43000	126300	215500	35300
6 30000	55100	52000	266200	51700	135300	266100	45200
7 35000	62200	60200	395640	59600	142800	360100	58000
8 40000	72800	68600	413800	67500	150900	376100	76000
9 45000	82000	76400	428800	75800	189700	389200	117400

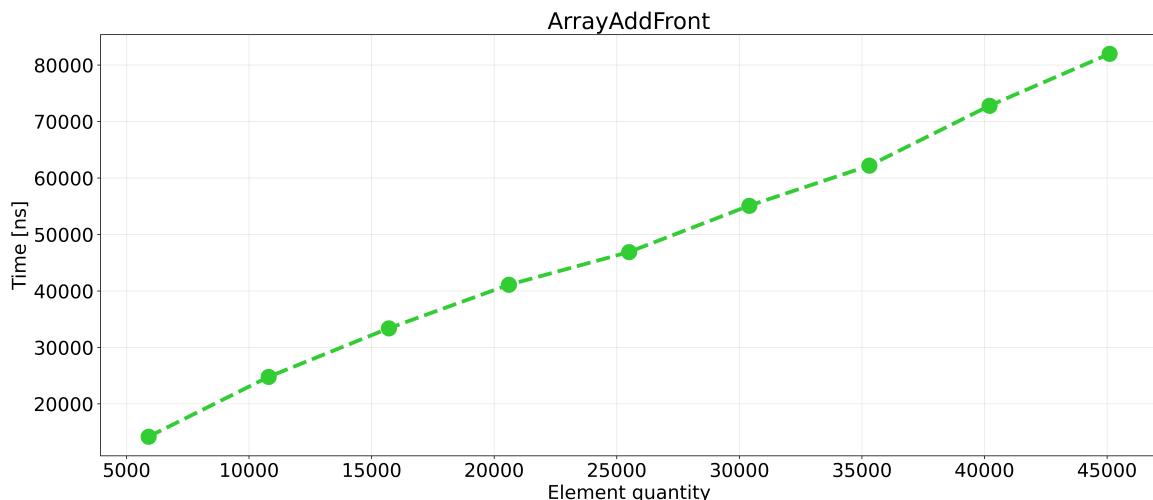


Figure 1: arrayAddFront

Wykres dodawania elementu z przodu tablicy

Wykres dodawania elementu z tyłu tablicy

Wykres dodawania elementu w środku tablicy

Wykres usuwania elementu z przodu tablicy

Wykres usuwania elementu z tyłu tablicy

Wykres usuwania elementu w środku tablicy

Wykres znajdowania elementu w tablicy

**Wnioski na temat tablicy** Czas wykonywania operacji wzrastał wraz z ilością elementów w tablicy, więc czas potrzebny na dodawanie, usuwanie bądź znajdowanie elementu zależy od wielkości tej struktury. Wykresy są w większości funkcją liniową.

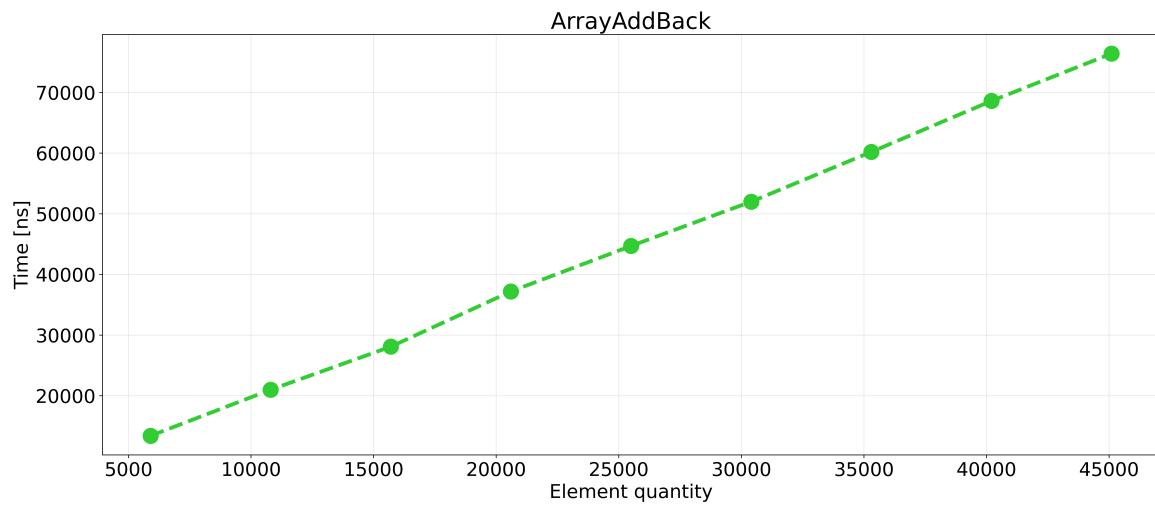


Figure 2: arrayAddBack

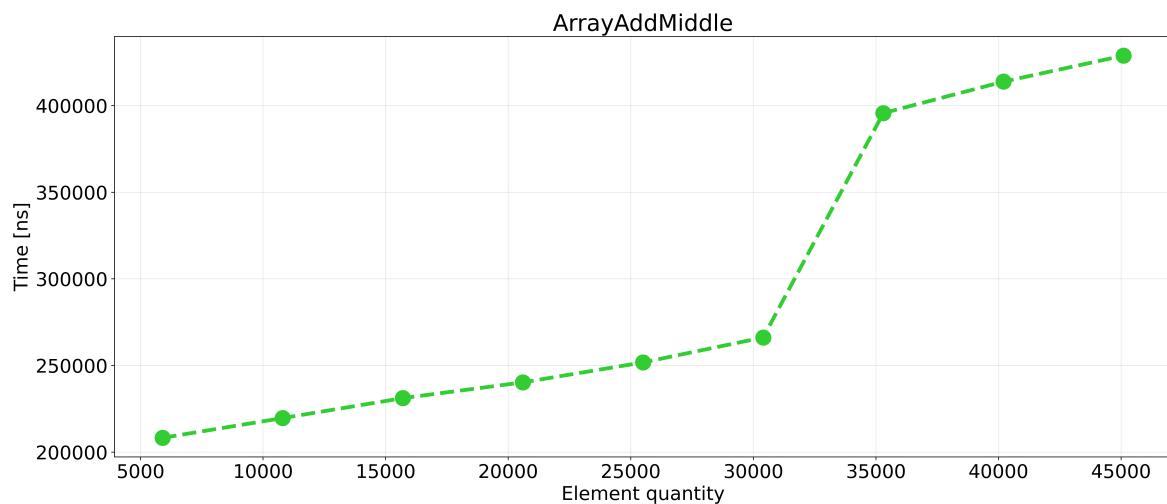


Figure 3: arrayAddAnywhere

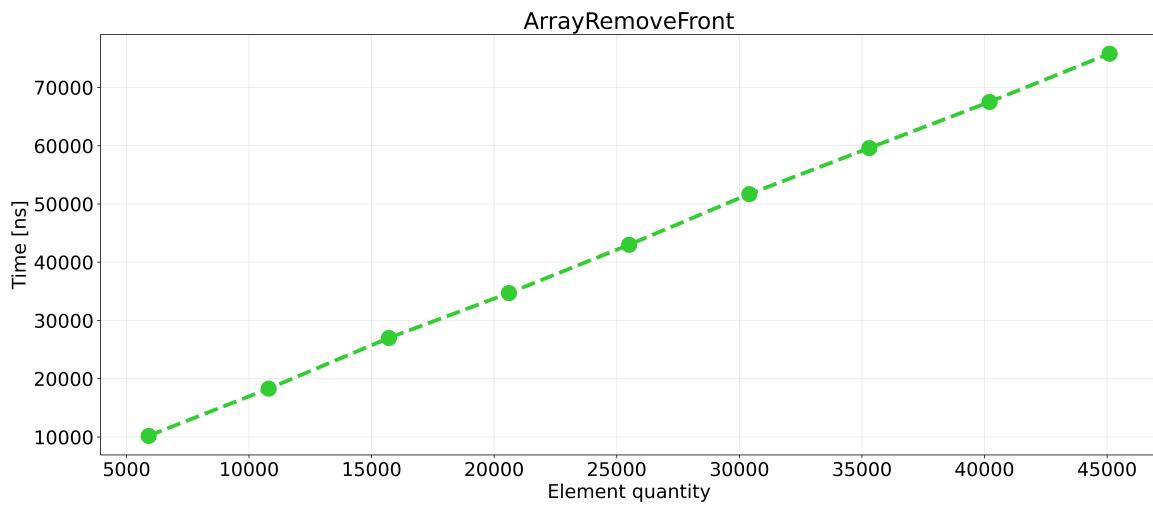


Figure 4: arrayRemoveFront

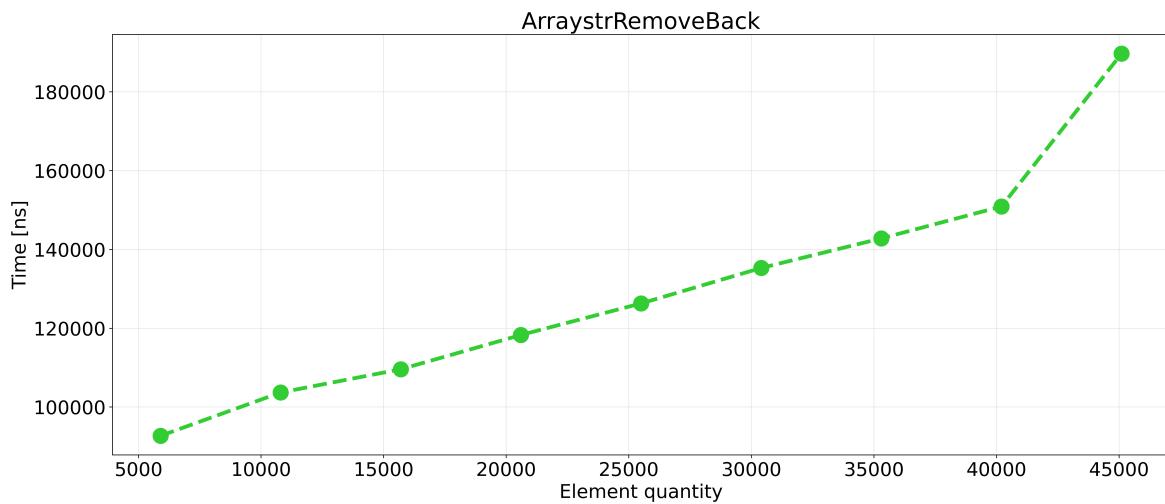


Figure 5: arrayRemoveBack

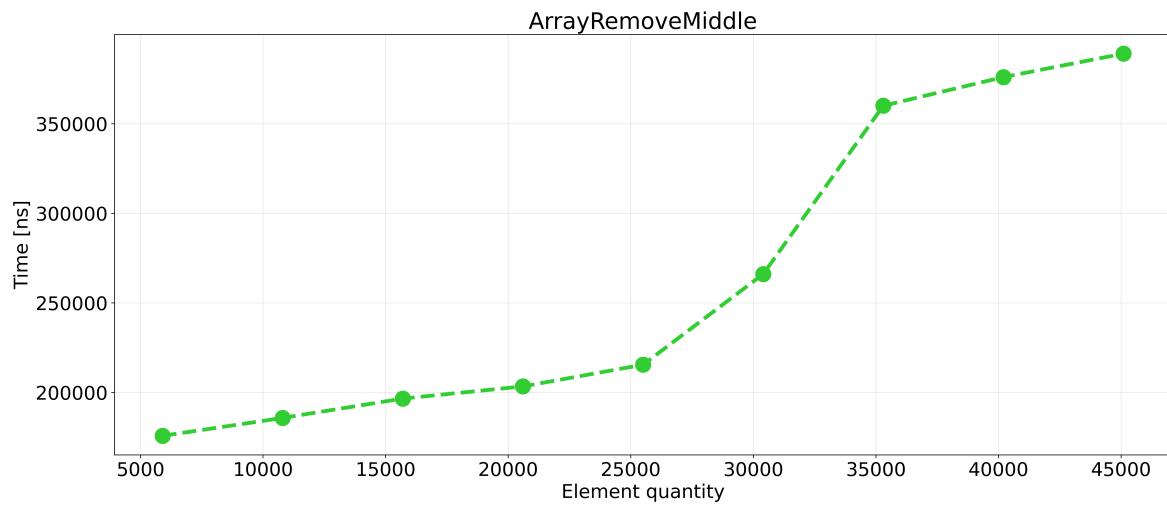


Figure 6: arrayRemoveAnywhere

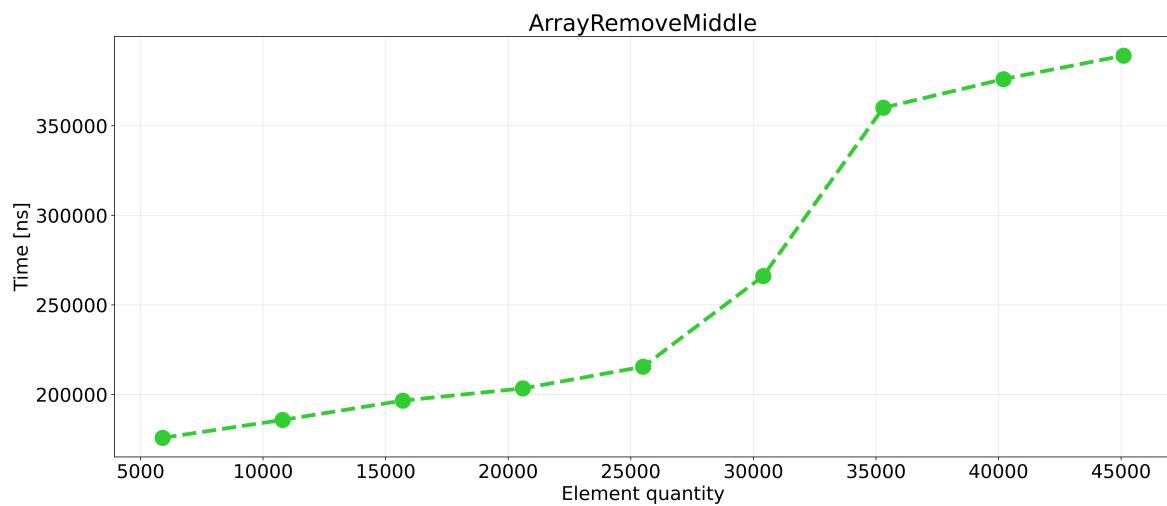


Figure 7: arrayFind

## Pomiary listy dwukierunkowej

### Wyniki pomiarów listy

Liczba danych	Dodawanie na początek	Dodawanie na koniec	Dodawanie gdziekolwiek	Usuwanie początek	Usuwanie koniec	Usuwanie gdziekolwiek	Szukanie
L.p. j	[ns]	[ns]	[ns]	[ns]	[ns]	[ns]	[ns]
1 5000	300	21100	19500	100	400	19400	200
2 10000	300	43940	36900	100	400	36000	3800
3 15000	400	65660	52900	100	400	52800	3900
4 20000	400	88800	70800	100	400	69400	20480
5 25000	400	112700	86500	100	400	85800	55700
6 30000	400	141100	100500	100	400	02000	99200
7 35000	400	156400	117000	100	400	18900	156300
8 40000	400	180700	133100	100	400	35900	216000
9 45000	400	198900	149800	100	500	55900	277400

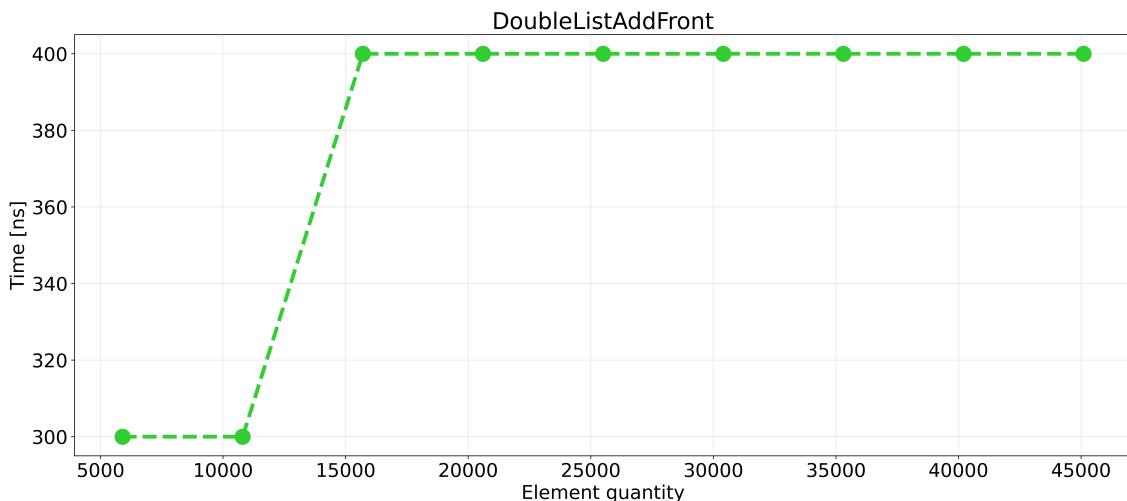


Figure 8: listAddFront

Wykres dodawania elementu z przodu listy

Wykres dodawania elementu z tyłu listy

Wykres dodawania elementu w środku listy

Wykres usuwania elementu z przodu listy

Wykres usuwania elementu z tyłu listy

Wykres usuwania elementu w środku listy

Wykres znajdowania elementu w liście

Wnioski na temat listy coś tu będzie.

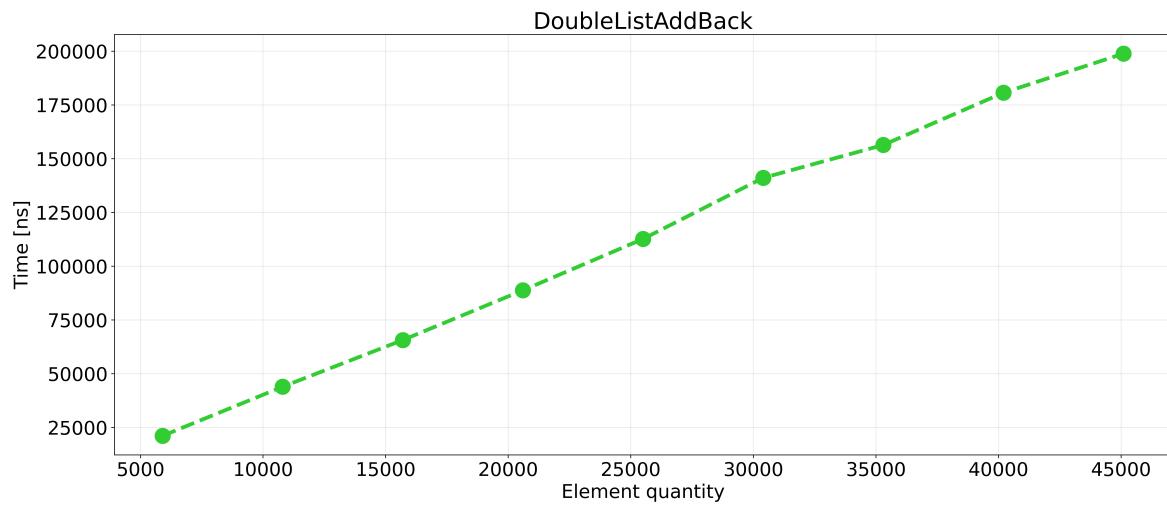


Figure 9: listAddBack

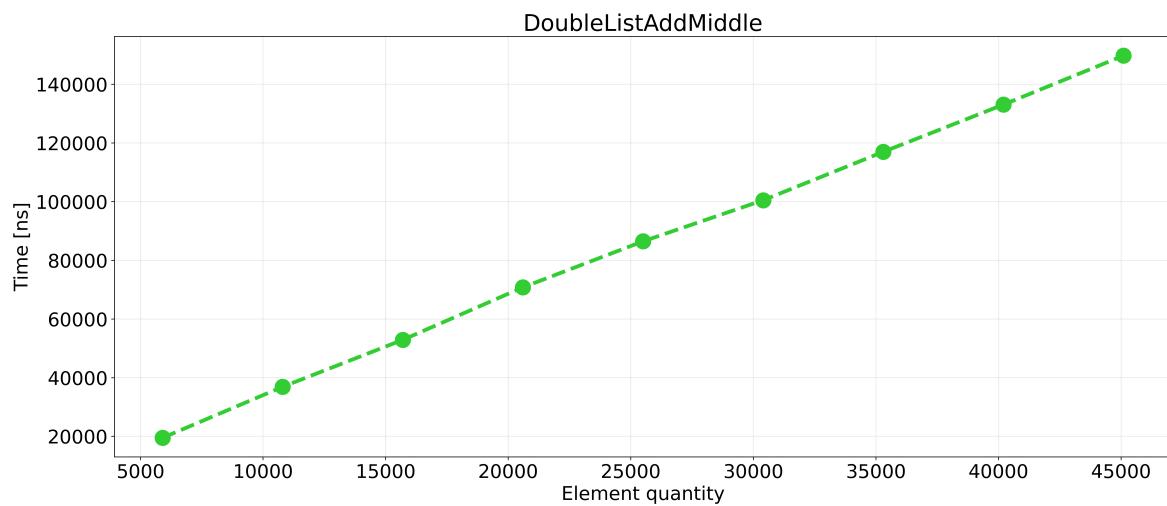


Figure 10: listAddAnywhere

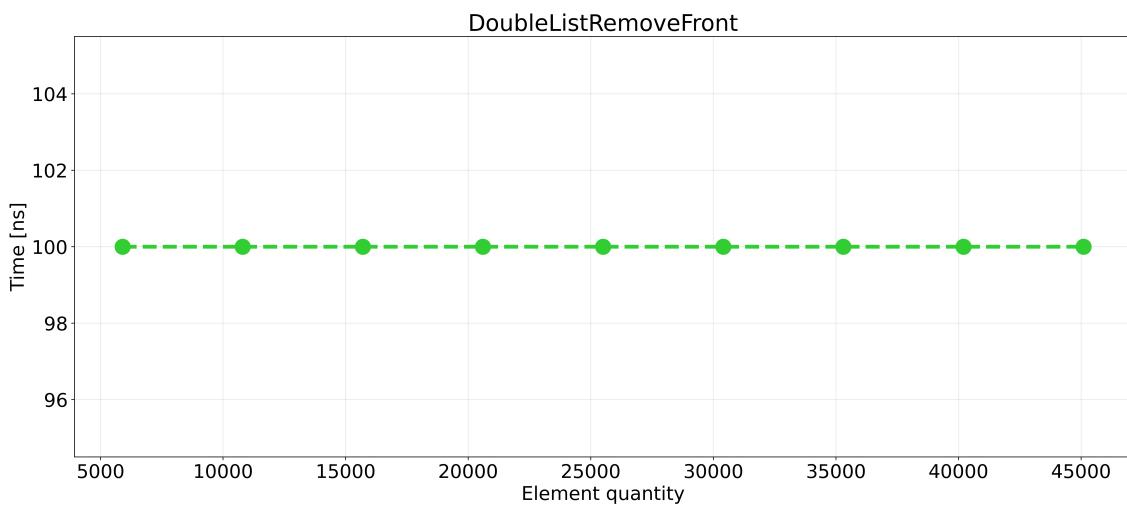


Figure 11: listRemoveFront

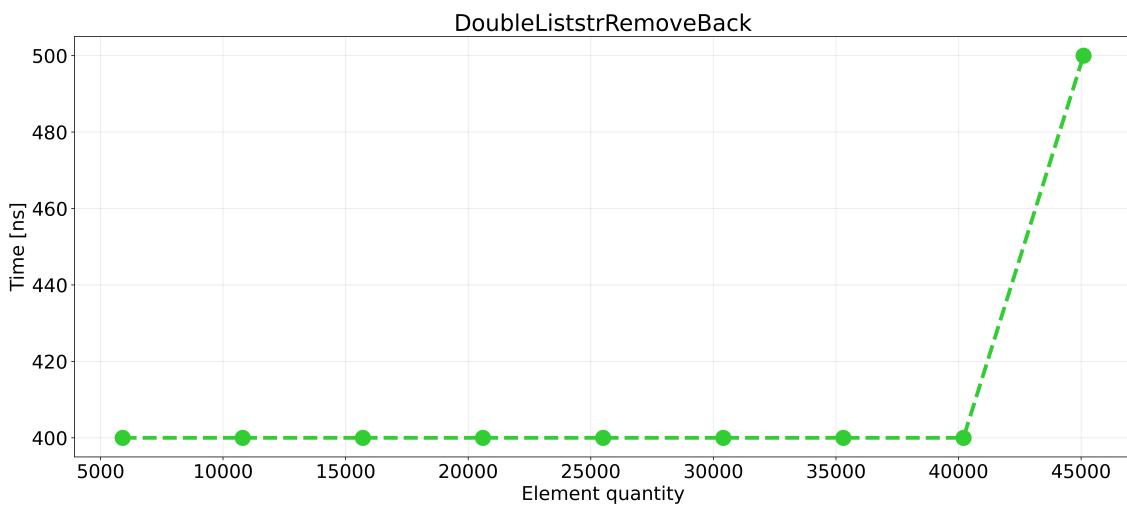


Figure 12: listRemoveBack

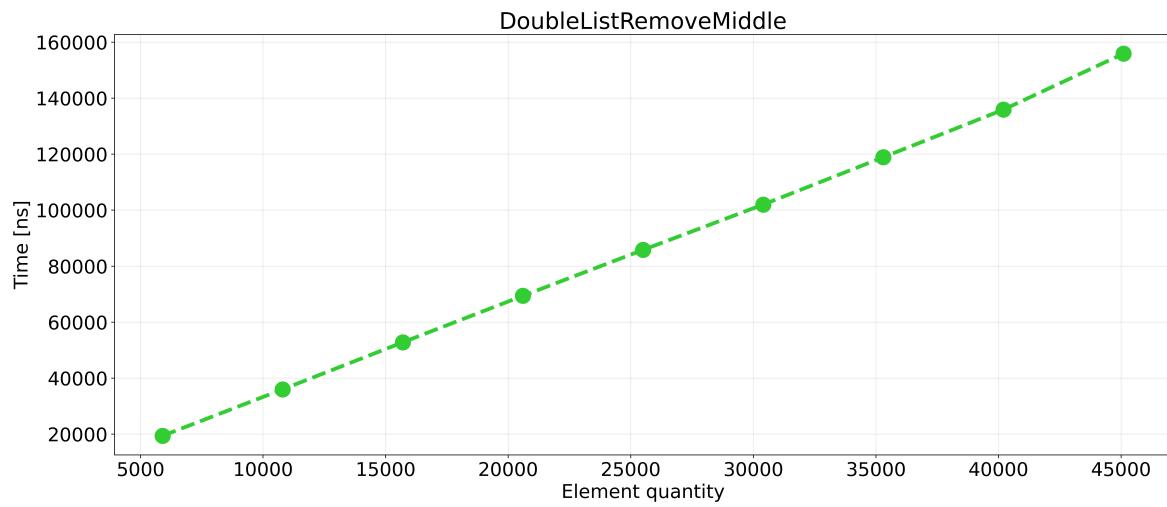


Figure 13: listRemoveAnywhere

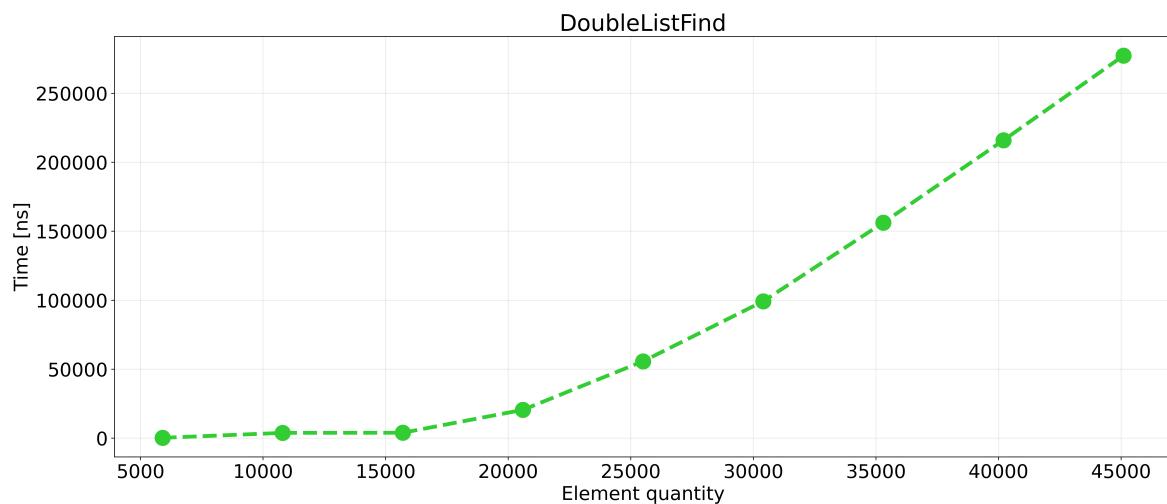


Figure 14: listFind

Pomiary kopca binarnego

Wyniki pomiarów kopca

L.p.	j	Liczba danych	Dodawanie	Usuwanie	Szukanie
		[ns]	[ns]	[ns]	[ns]
1	5000	11600	44800	7900	
2	10000	20400	49100	14000	
3	15000	27500	55000	20300	
4	20000	35200	61400	27300	
5	25000	42900	67600	36000	
6	30000	51700	75400	43000	
7	35000	59200	90300	48200	
8	40000	67700	105100	54800	
9	45000	76600	125400	63200	

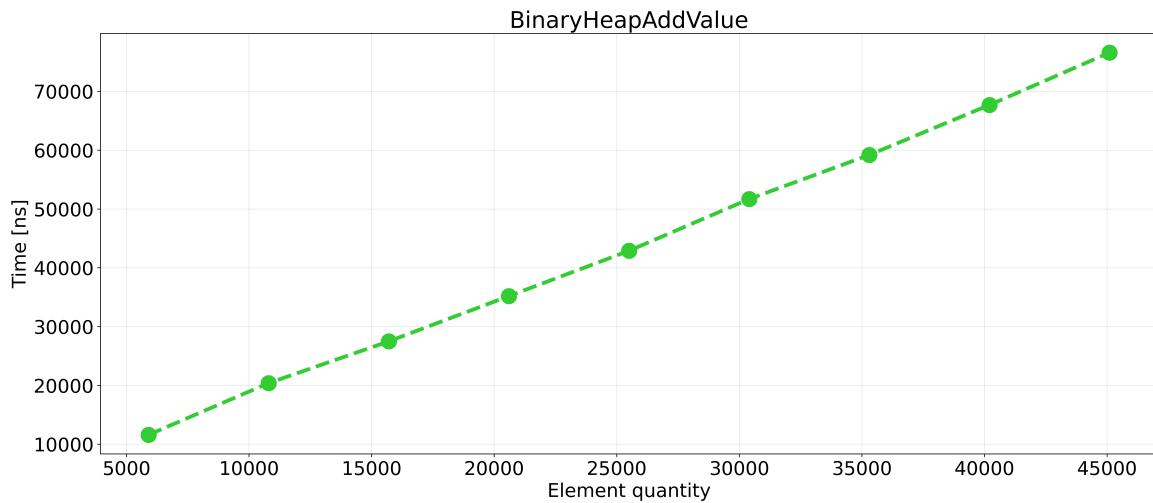


Figure 15: heapAdd

### Dodawanie elementu w kopcu

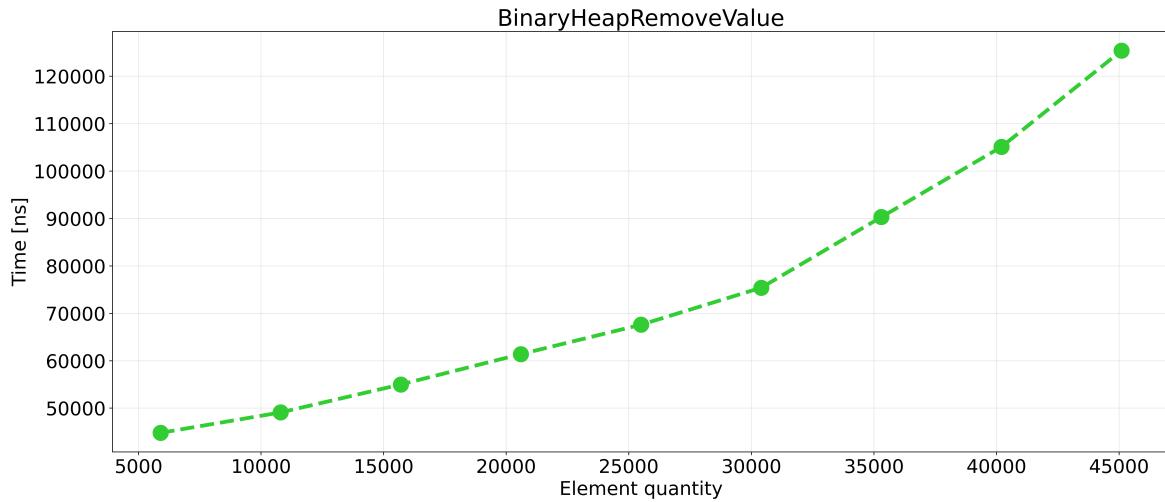


Figure 16: heapRemove

### Usuwanie elementu w kopcu

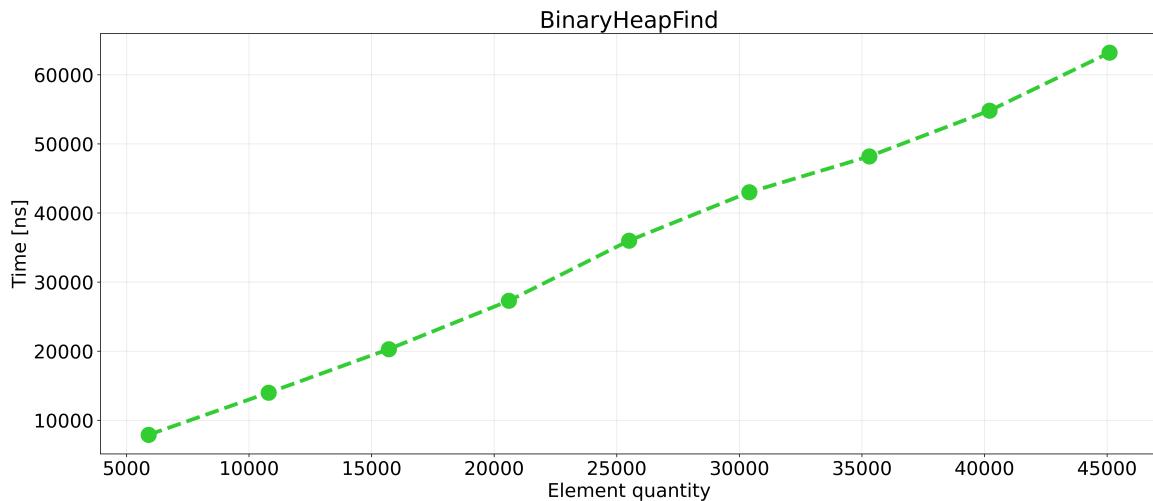


Figure 17: heapFind

### Znajdowanie elementu w kopcu

#### Pomiary drzewa czerwono-czarnego

#### Wyniki pomiarów drzewa

L.p.	j	Liczba danych	Dodawanie	Usuwanie	Szukanie
		[ns]	[ns]	[ns]	[ns]
1	5000	500	400	300	
2	10000	600	400	400	
3	15000	600	400	400	
4	20000	600	500	400	
5	25000	600	500	400	
6	30000	700	600	400	
7	35000	700	600	400	
8	40000	700	700	500	
9	45000	800	700	500	

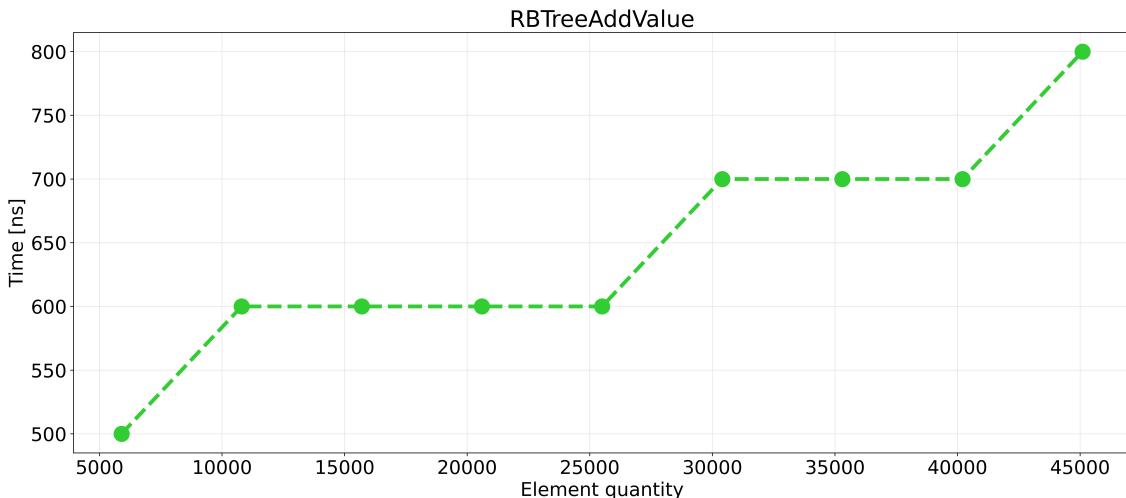


Figure 18: treeAdd

	Liczba danych	Dodawanie	Usuwanie	Szukanie
4	20000	600	800	400
5	25000	700	900	500
6	30000	700	900	500
7	35000	700	1000	500
8	40000	800	1100	600
9	45000	800	1200	700

**Dodawanie elementu w drzewie AVL**

**Usuwanie elementu w drzewie AVL**

**Znajdowanie elementu w drzewie AVL**

**Wnioski na temat drzewa AVL** Coś tu będzie.

**Wnioski końcowe**

Tutaj musi coś być.

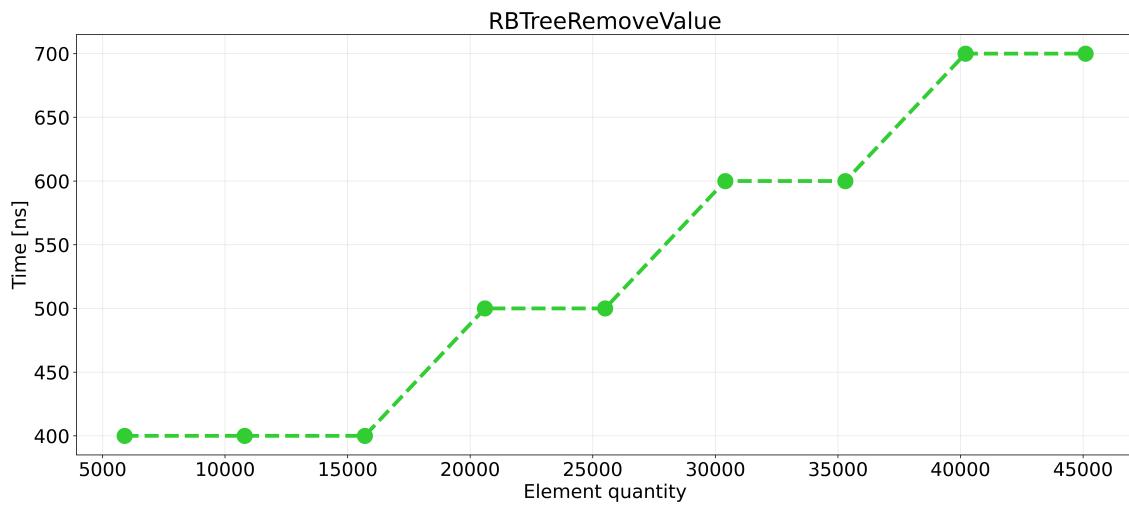


Figure 19: treeRemove

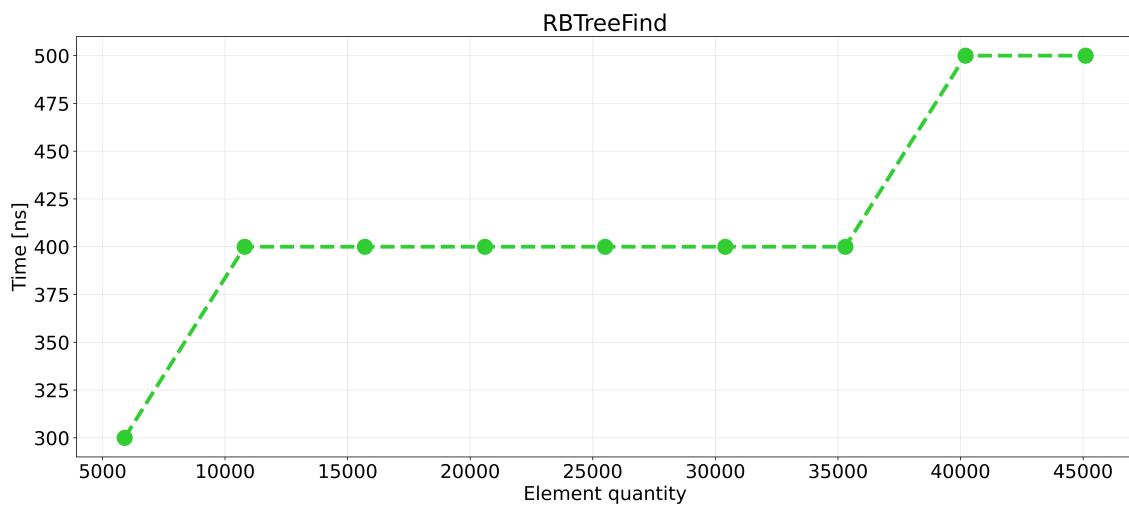


Figure 20: treeFind

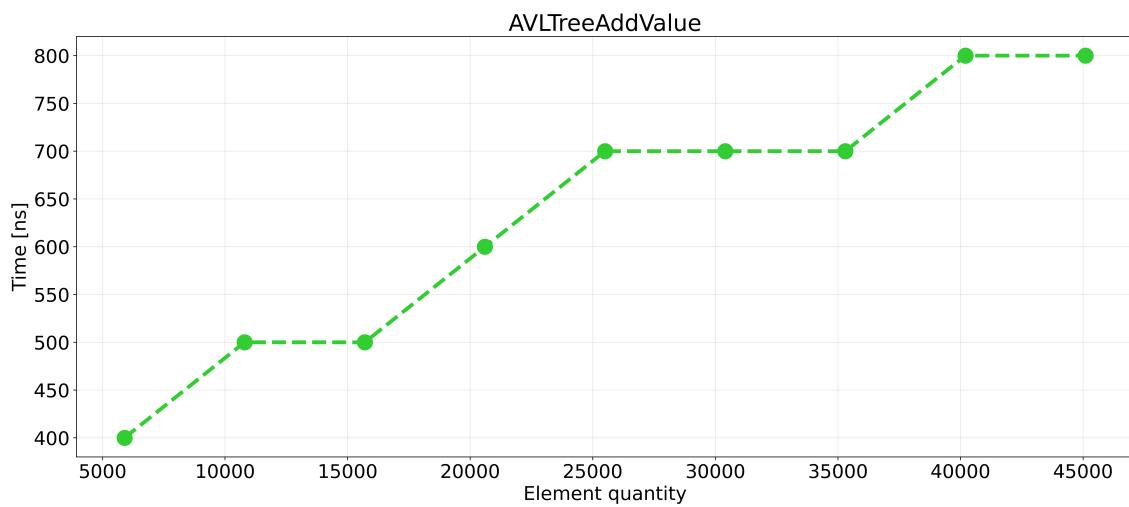


Figure 21: AVLTreeAdd

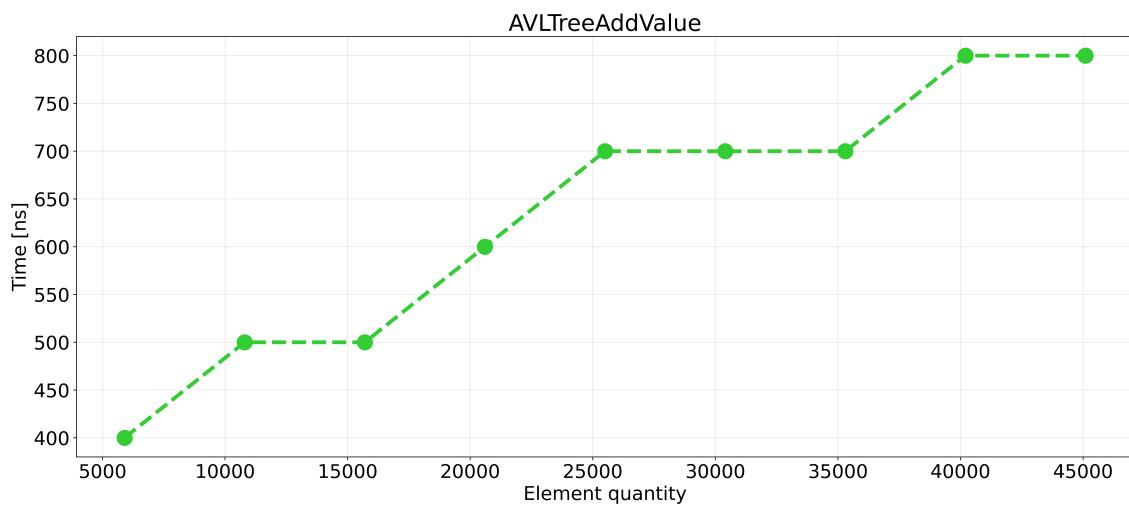


Figure 22: AVLTreeRemove

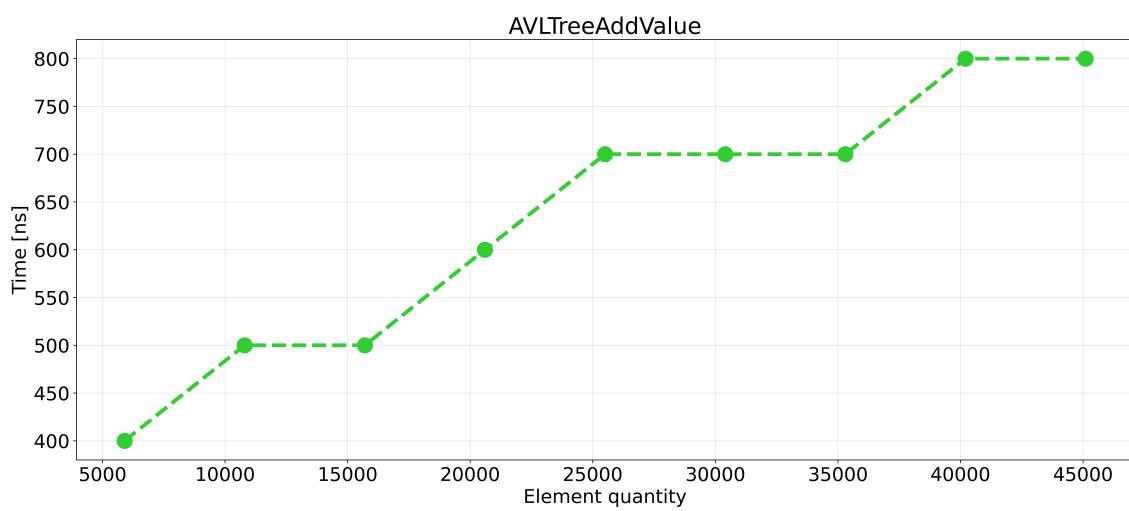


Figure 23: AVLTreeFind