Sprawozdanie z zajęć laboratoryjnych PAMSI

Sortowanie

1. Cel zadania

Zaimplementowanie oraz zbadanie wydajności 3 wybranych algorytmów sortowania (Przynajmniej dwa o złożoności nie większej niż O(n log n)).

2. Sortowanie przez scalanie

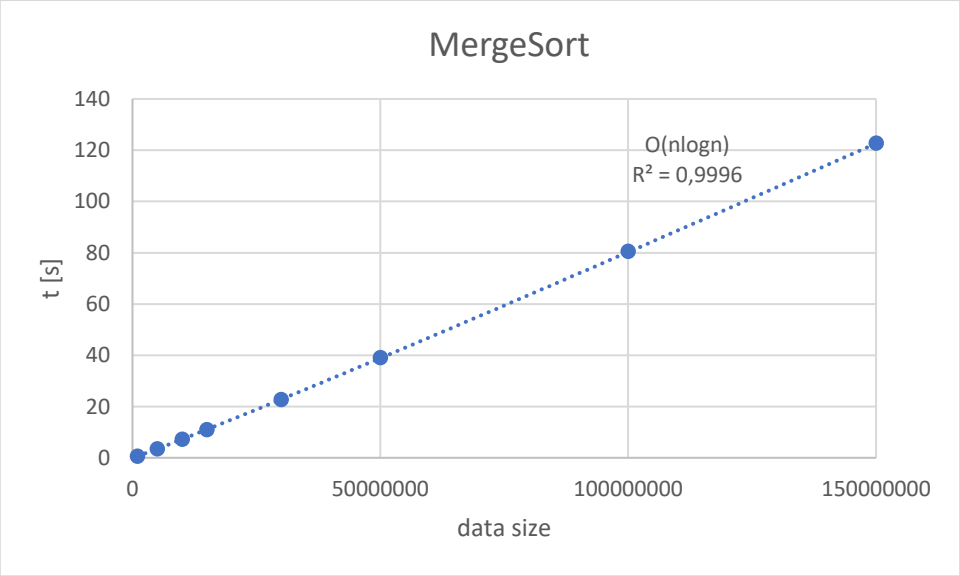
Sortowanie to stosuję zasadę „dziel i zwyciężaj” oraz ma złożoność czasową O (n log n).

A) Wyniki działania programu „Sorting_performance”

```
Merge sort for data size: 1000000
Sorting time: 0.655486

Merge sort for data size: 5000000
Sorting time: 3.48788

Merge sort for data size: 10000000
Sorting time: 7.2298

Merge sort for data size: 15000000
Sorting time: 10.9788

Merge sort for data size: 30000000
Sorting time: 22.688

Merge sort for data size: 50000000
Sorting time: 39.0611

Merge sort for data size: 100000000
Sorting time: 80.584

Merge sort for data size: 150000000
Sorting time: 122.818
```

Rysunek 1 - Sorting_performance - MergeSort

B) Wykres czasu wykonaniu algorytmu w zależności od liczby danych do posortowania



Rysunek 2 – Wykres czasu MergeSort

3. Sortowanie szybkie

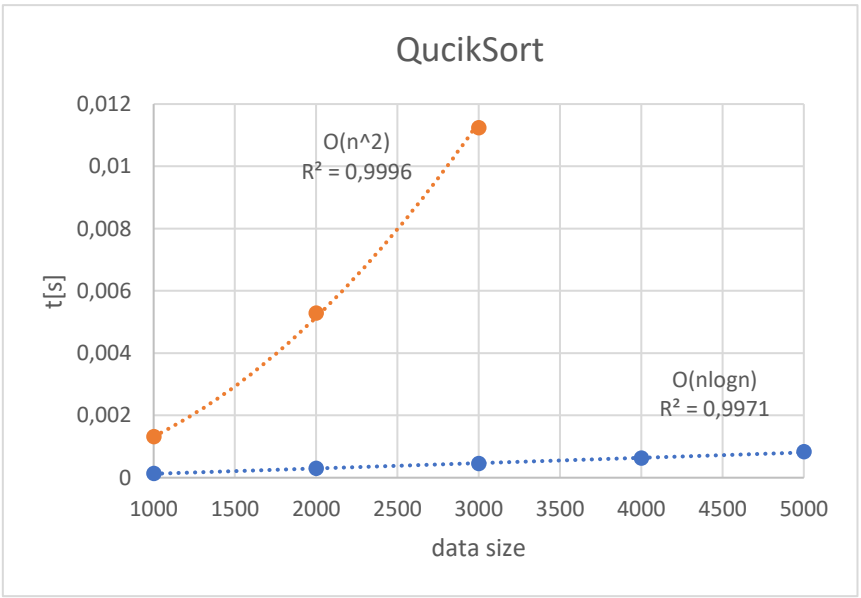
Sortowanie to stosuję zasadę „dziel i zwyciężaj”. Ma złożoność czasową O (n log n) oraz w przypadku pesymistycznym $O(n^2)$. Dla dobrego zilustrowania dwóch przypadków użyty został inny zestaw danych

A) Wyniki działania programu „Sorting_performance”

```
adrian@DESKTOP-BS4R2K2:/mnt/f/Studia/semestr 4/PAMSI/sortowanie/build$ ./sorting_performance
Quick sort for data size: 1000
Sorting time: 0.0001372

Quick sort for data size: 2000
Sorting time: 0.0002971

Quick sort for data size: 3000
Sorting time: 0.000453

Quick sort for data size: 4000
Sorting time: 0.0006251

Quick sort for data size: 5000
Sorting time: 0.0008278

Quick sort for data size: 6000
Sorting time: 0.0009598

Quick sort for data size: 7000
Sorting time: 0.0011511

Quick sort for data size: 1000 WORST
Sorting time: 0.0013144

Quick sort for data size: 2000 WORST
Sorting time: 0.0052842

Quick sort for data size: 3000 WORST
Sorting time: 0.0112329

Quick sort for data size: 4000 WORST
Sorting time: 0.0195694

Quick sort for data size: 5000 WORST
Sorting time: 0.0288104

Quick sort for data size: 6000 WORST
Sorting time: 0.040843

Quick sort for data size: 7000 WORST
Sorting time: 0.0555747
```

Rysunek 3 - Sorting_performance - QuickSort

B) Wykres i porównanie czasów wykonaniu algorytmu w zależności od liczby danych do posortowania (dla przypadku przeciętnego i pesymistycznego)



Rysunek 4 – Wykres czasu QuickSort

4. Sortowanie przez kopcowanie

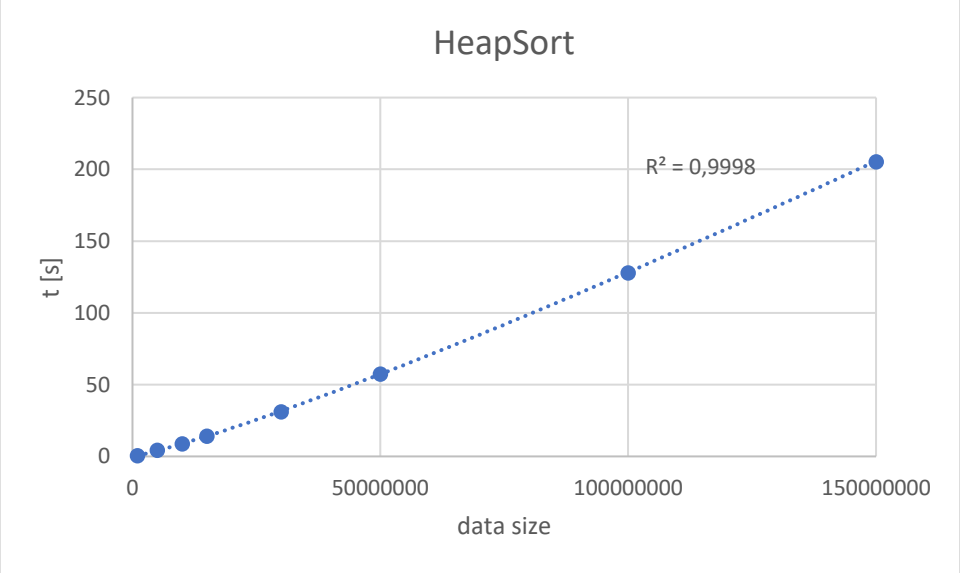
Sortowanie to stosuję implementacje kopca binarnego oraz ma złożoność czasową O (n log n).

A) Wyniki działania programu „Sorting_performance”

```
adrian@DESKTOP-BS4R2K2:/mnt/f/Studia/semestr 4/PAMSI/sortowanie/build$ ./sorting_performance
heap sort for data size: 1000000
Sorting time: 0.584905

heap sort for data size: 5000000
Sorting time: 4.18097

heap sort for data size: 10000000
Sorting time: 8.70821

heap sort for data size: 15000000
Sorting time: 14.1781

heap sort for data size: 30000000
Sorting time: 30.9983

heap sort for data size: 50000000
Sorting time: 57.4222

heap sort for data size: 100000000
Sorting time: 127.906

heap sort for data size: 150000000
Sorting time: 205.317
```

Rysunek 5 - Sorting_performance - HeapSort

B) Wykres czasu wykonaniu algorytmu w zależności od liczby danych do posortowania

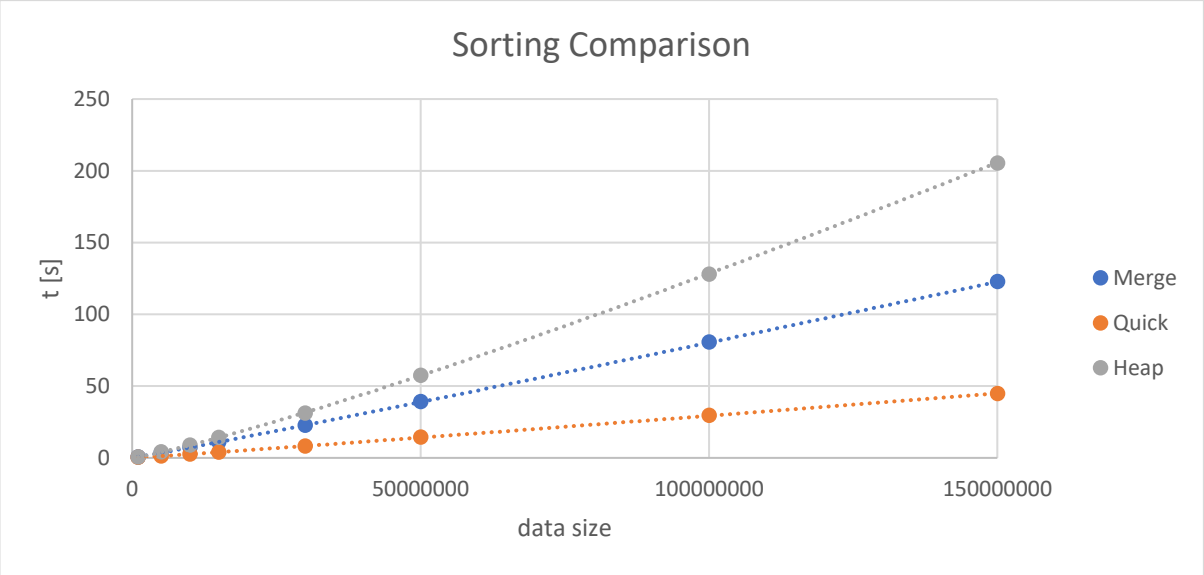


Rysunek 6 – Wykres czasu HeapSort

5. Porównanie badanych algorytmów

time [s] / Data size	QuickSort	MergeSort	HeapSort
10	0,0000017	0,0000051	0,0000025
100	0,0000116	0,0000517	0,000021
1000	0,0001353	0,0004744	0,00038
10000	0,0016976	0,0054986	0,004035
100000	0,0207575	0,0620587	0,0525769
1000000	0,231984	0,700554	0,680275
10000000	2,75809	7,78987	10,3461
100000000	30,8554	85,6841	144,599
1000000000	346,585	949,256	2008,72

Tabela 1 – Porównanie czasów działania 3 typów sortowania



Rysunek 7 – Porównanie czasów działania 3 typów sortowania

Jak można zaobserwować z powyższej tabeli oraz wykresu – najszybciej działającym typem sortowania spośród trzech badanych typów jest sortowanie szybkie. Niestety stosowanie go wiąże się z ryzykiem (znikomym) wystąpienia przypadku negatywnego, gdzie złożoność czasowa wynosi aż $O(n^2)$. Jeśli dla danego zbioru danych chcemy mieć pewność co do stałej złożoności czasowej dla wszystkich przypadków, możemy skorzystać z algorytmów sortowania przez scalanie lub kopcowanie. Mergesort jest jednak szybszy od HeapSort oraz jest jedynym (z badanych) stabilnym algorytmem sortowania.