Sprawozdanie Sortowanie

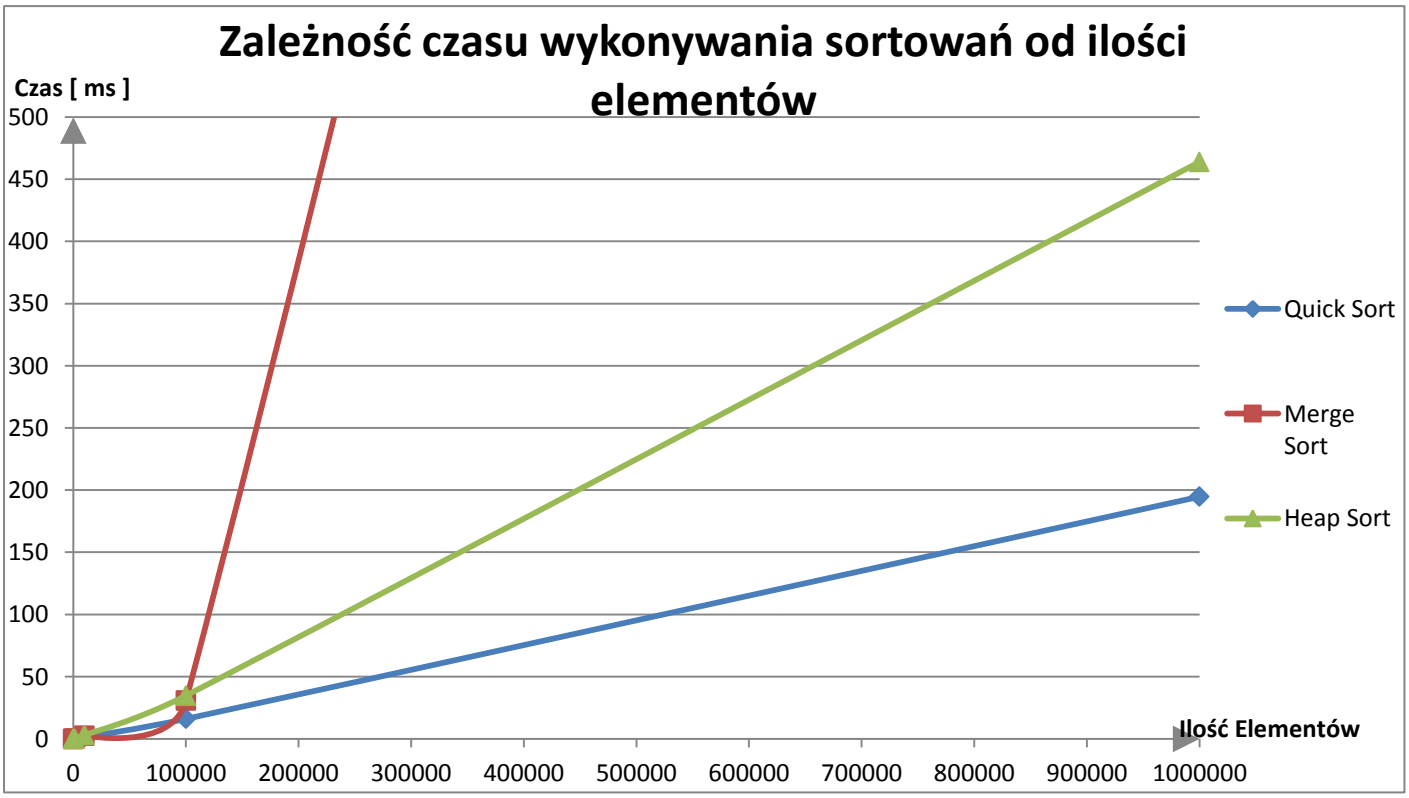
Do posortowania zbióru losowo wygenerowanych elementów użyłem :

- Quick Sort (Sortowanie Szybkie)
- Merge Sort (Sortowanie przez scalanie)
- Heap Sort (Sortowanie przez budowanie kopca)

Wyniki działania programu :

```
Czasy sortowania za pomoca metody Quick Sort:
 (10 elementow) ->> 0.001 ms
 (100 elementow) ->> 0.009 ms
 (1000 elementow) ->> 0.109 ms
 (10000 elementow) ->> 1.356 ms
 (100000 elementow) ->> 16.01 ms
 (1e+06 elementow) ->> 194.68 ms
Czasy sortowania za pomoca metody Merge Sort:
 (10 elementow) ->> 0.002 ms
 (100 elementow) ->> 0.016 ms
 (1000 elementow) ->> 0.182 ms
 (10000 elementow) ->> 2.427 ms
 (100000 elementow) ->> 30.68 ms
 (1e+06 elementow) ->> 3373.65 ms
Czasy sortowania za pomoca metody Heap Sort:
 (10 elementow) ->> 0.001 ms
 (100 elementow) ->> 0.015 ms
 (1000 elementow) ->> 0.229 ms
 (10000 elementow) ->> 3.125 ms
 (100000 elementow) ->> 34.742 ms
 (1e+06 elementow) ->> 463.924 ms
 Calkowity czas wykonywania to : 4121.16 ms
```

Dane wyświetlone za pomocą wykresu (wykres dostępny osobno w pliku wykres.pdf):



Wnioski:

- Algorytm Quick Sort okazał się najszybszy. Prawdopodobnie jest to spowodowane prostotą tego algortymu (opiera się na Buble Sort, co jest chyba najprostszym algorytmem możliwym). W przeciwieństwie do sortowania bąbelkowego, Quick Sort dzieli za każdym razem problem na dwie części, co znacznie przyśpiesza proces.
- Algorytm Merge Sort zdecydowanie najwolniejszy. Jest to prawdopodobnie spowodowane tym, że jako jedyny spośród tych trzech, podczas jego wykonywania jest tworzona dynamicznie tablica pomocnicza. Wydaje mi się, że to znacznie spowalnia tę metodę.
- Mimo tego, iż Heap Sort jest dosłownie trochę wolnijszy od Quick Sort, zawsze będę wybierał ten drugi algorytm (o ile będzie to możliwe) ze względu na jego prostotę.

Dokumentacja program dostępna w pliku pdf o nazwie refman (zapisana w LateX u) oraz w doxygen ie (dostępna w pliku : dox/html/index.html)