

Algorytmy Sortowania - Benchmark

Konrad Łakomy

30 03 2014

Zadanie do wykonania

1. Implementacja i benchmark algorytmów sortowania:
 - (a) Sortowanie przez scalanie (ang. MergeSort)
 - (b) Sortowanie szybkie (ang. QuickSort)
 - (c) Sortowanie przez kopcowanie (ang. HeapSort)
2. Dokładna analiza algorytmu QuickSort:
 - (a) Stały element Pivot(QuickSort1)
 - i. Losowe liczby
 - ii. Liczby posortowane w odwrotnej kolejności
 - (b) Losowy element Pivot(QuickSort2)
 - i. Losowe liczby
 - ii. Liczby posortowane w odwrotnej kolejności

Rezultaty benchmark'u

LiczbaElementów	MergeSort	HeapSort
10	1.5899e-05[s]	3.755e-06[s]
100	7.4595e-05[s]	0.000221813[s]
1000	0.0019067[s]	0.0117533[s]
10000	0.0211156[s]	0.86994[s]
100000	0.365772[s]	0.999881[s]

Tablica 1: Benchmark przy losowych elementach

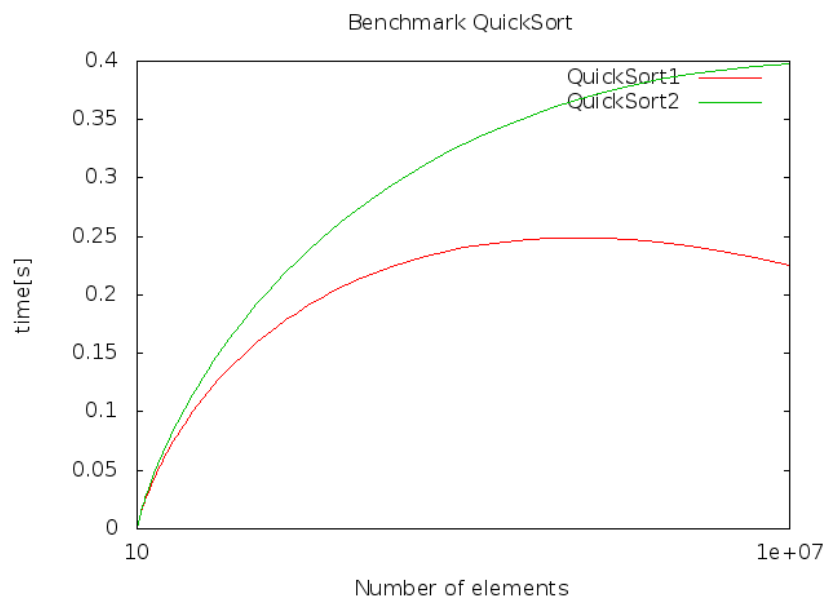
LiczbaElementów	QuickSort1	QuickSort2
10	2.477e-06[s]	2.183e-06[s]
100	2.5667e-05[s]	2.8629e-05[s]
1000	0.000223234[s]	0.000419652[s]
10000	0.00236572[s]	0.00281752[s]
100000	0.0353701[s]	0.0317925[s]
1000000	0.345989[s]	0.371444[s]
10000000	0.225309[s]	0.397442[s]

Tablica 2: Benchmark przy losowych elementach

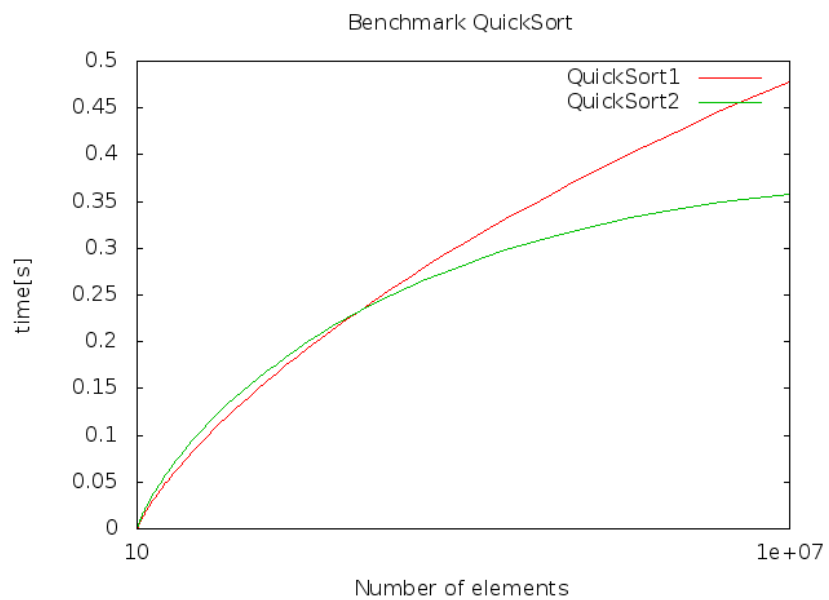
LiczbaElementów	QuickSort1	QuickSort2
10	1.414e-06[s]	4.91e-06[s]
100	1.0049e-05[s]	1.3996e-05[s]
1000	0.000247672[s]	0.000343511[s]
10000	0.00313185[s]	0.00220292[s]
100000	0.0215649[s]	0.0255427[s]
1000000	0.213807[s]	0.299311[s]
10000000	0.47854[s]	0.357441[s]

Tablica 3: Benchmark przy odwroconej kolejności elementów

QuickSort- Wykresy

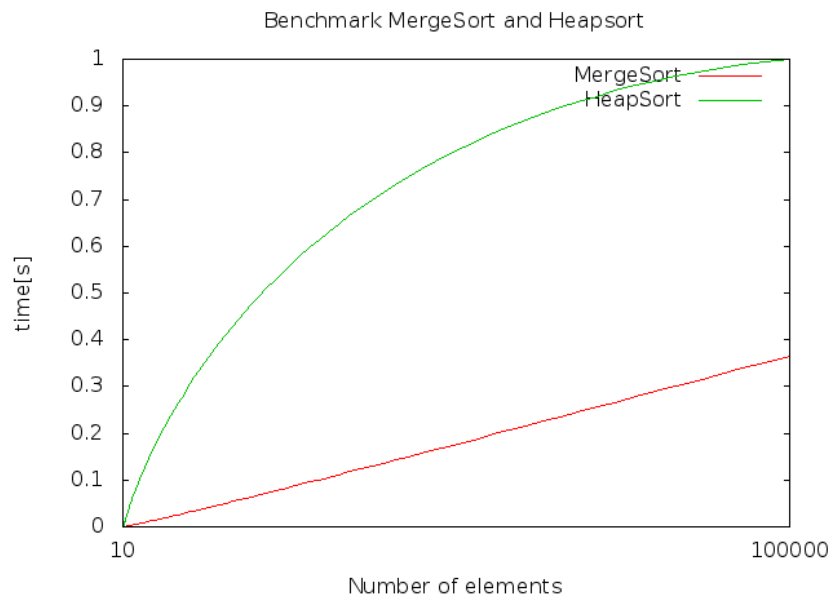


Rysunek 1: Wykres przy losowych elementach



Rysunek 2: Wykres przy odwroconej kolejności elementów

MergeSort, HeapSort - Wykres



Rysunek 3: Wykres przy losowych elementach

Wnioski

Efektywność algorytmów sortowania w dużej mierze zależy od tego w jakiej strukturze danych zaimplementowany jest dany algorytm. W zależności czy jest to stos czy kolejka oraz czy jest to implementacja w tablicy czy w liście będzie miało znaczenie przy mierzeniu czasu pracy danego algorytmu sortowania.

W przypadku algorytmu sortowania szybkiego jego najlepsza złożoność obliczeniowa to $O(n \log n)$, przypadek średni charakteryzuje się o 39 procent większą złożonością obliczeniową, natomiast najgorszy przypadek to $O(n^2)$

Z powyższych wykresów można wywnioskować że gdy liczby są przypadkowe, lepiej sprawdza się algorytm z elementem osiowym który jest wybierany jako mediana. Natomiast w przypadku gdy liczby są posortowane w odwrotnej kolejności lepsza okazuje się wersja algorytmu z losowo wybranym elementem osiowym.

Algorytmy sortowania przez kopcowanie i przez scalanie charakteryzuje się stałą złożonością obliczeniową we wszystkich przypadkach i jest ona równa $O(n \log n)$. Lecz algorytm sortowania przez kopcowanie jest algorytmem niestabilnym i wykazuje się większym czasem działania niż w przypadku algorytmu przez scalanie czy sortowania szybkiego.