Sprawozdanie ze złożoności obliczeniowej - PAiMSI

Patryk Jedrzejko

23.03.2014

Celem ćwiczenia było napisanie programu, który bedzie zawierał algorytmy sortowania, a nastepnie zbadanie złożoności obliczeniowej dla sortowania wczytanych danych z pliku. Program został napisany na podstawie wcześniejszej klasy Kolejka. Jest to klasa kolejki jako tablica. Kolejno algorytmy sortowały ilość elementów
n = 10, 100, 1000, 10000,100000.

Wybrane algorytmy sortowania:

QuickSort - czyli szybkiego sortowania,

MergeSort - sortowanie przez scalanie,

HeapSort - sortowanie kopcem.

Pierwszy z nich algorytm QuickSort jest przykładem zastosowania techniki "dziel i zwyciężaj"i składa sie z 3 faz: 1.najpierw wejściowy ciag liczb jest dzielony na dwa podciagi, 2. nastepnie te dwa podciagi sa sortowane rekurencyjnie, 3. posortowane podciagi scala sie w jeden posortowany ciag.

Algorytm MergeSort jest jednym z prostrzych przykładów zastosowania wyżej wymienionej techniki "dziel i zwyciężaj", to znaczy: 1. dzieli ciag na dwa ciagi o długości n/2, 2. sortuje te dwa ciagi przy użyciu sortowania przez scalanie, 3. posortowane ciagi długości n/2 scala w posortowany ciag długości n.

Trzeci algorytm sortowania HeapSort, czyli sortowanie przez kopcowanie opiera sie na stworzeniu kopca, a nastepnie na rozbiorze utworzonego kopca, w wyniku czego otrzymujemy posortowany ciag.

Analizujac wyniki obliczeń w arkuszu kalkulacyjnym otrzymanych czasów sortowania użytych algorytmów możemy wyciagnać wnioski takie, że:

Klasa złożoności obliczeniowej dla algorytmu QuickSort dla elementów posortowanych malejaco oraz posortowanych wynosi $O(n^2)$
czyli dla najgorszego przypadku (złożoność pesymistyczna), zaś dla najlepszego przypadku klasa złożoności przyjmuje wartość O(nlogn) dla

elementów losowych (złożoność optymistyczna oraz typowa). Przez co możemy określić iż sortowanie szybkie nie jest sortowaniem stabilnym.

Dla algorytmu MergeSort oraz HeapSort klasa złożoności wynosi O(n logn) dla elementów losowych, posortowanych jak i posortowanych malejaco. Porównujac czasy wykonania sortowania dla różnych ilości n elementów do posortowania widzimy, że dla małej liczby elementów tj. n = 10, to sa różnice nieznaczne, zaś dla n = 100000 elementów możemy już stwierdzić różnice czasowa w wykonaniu sortowania dla elementów losowych.

Nastepnie możemy odczytać z tabeli, że dla sortowania przez scalanie dla elementów losowych algorytm jest najwolniejszy. Dla elementów posortowanych malejaco algorytm wykonuje sortowanie najszybciej.

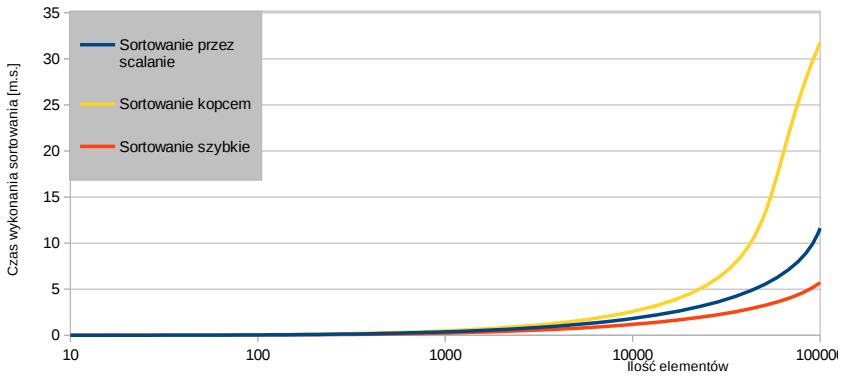
Zaś dla sortowania szybkiego widzimy znaczace zmiany dla elementów posortowanych i malejacych, co może wynikać z nieprawidłowego wykonania algorytmu sortowania badź też niestabilności sortowania szybkiego. Znaczace zmiany widzimy już dla n = 1000. Dlatego QuickSort jest najszybszy dla elementów losowych.

Dla sortowania przez kopcowanie mamy podobnie jak z SortMerge. Widzimy, że dla elementów posortowanych sortowanie wykonuje sie najszybciej, zaś dla malejacych najwolniej.

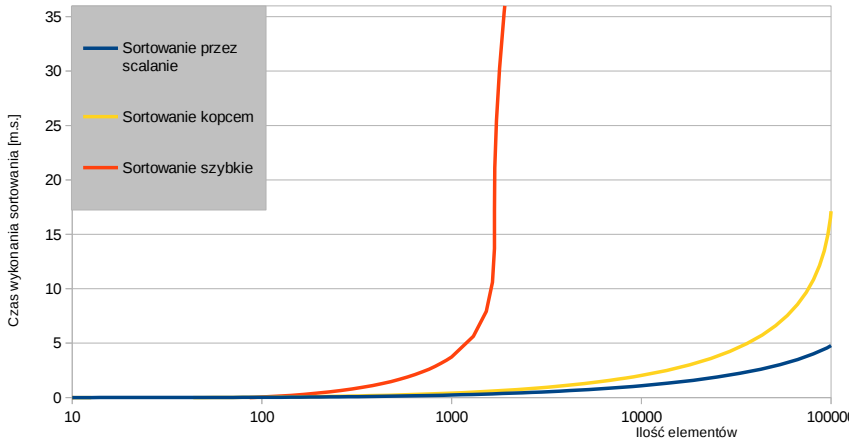
Możemy zatem stwierdzić, iż algorytm szybkiego sortowania jest najszybszy, ale tylko dla elementów losowych. Zaś najwolniejszym z nich jest algorytm kopcowania. Co za tym idzie algorytm QuickSort jest efektywniejszy od pozostałych algorytmów sortowania.
Tabela wyników oraz wykresy na stronie 3 oraz 4.

Sortowanie przez scalanie						
n	10	100	1000	10000	100000	1000000
tp	0,002	0,024	0,235	1,089	14,726	
tm	0,002	0,017	0,21	1,037	12,949	
tl	0,003	0,034	0,366	1,819	20,659	
Sortowanie szybkie						
n	10	100	1000	10000	100000	1000000
tp	0,003	0,047	3,736	120,714	12246,5	
tm	0,001	0,068	3,232	120,979	12018,6	
tl	0,001	0,021	0,24	1,176	11,77	
Sortowanie kopcem						
n	10	100	1000	10000	100000	1000000
tp	0,003	0,031	0,409	2,033	25,848	
tm	0,004	0,037	0,531	2,692	36,007	
tl	0,003	0,053	0,477	2,574	31,787	

Wykres złożoności dla losowych elementów:



Wykres dla posortowanych elementów:



Wykres dla malejących elementów: