Algorytmy sortowania: Quick Sort, Merge Sort, Heap Sort - porównanie wydajności.

Marcin Ochman

22 marca 2014

1 Opis problemu

Sortowanie to jedna z najczęstszych czynności wykonywana przez obecne komputery. Ogromne bazy danych, z którymi programiści mają do czynienia wymagają od nich odpowiedniej wiedzy, jak szybko móc te dane umieścić w określonym porządku. Sortowanie w świecie algorytmów ma swoje szczególne miejsce. Algorytmy sortowania są dobrze przeanalizowane, znamy ich złożoności obliczeniowe, zapotrzebowanie na pamięć. Jednymi z najpopularniejszych są tytułowe:

- sortowanie szybkie
- sortowanie przez scalanie
- sortowanie przez kopcowanie

Co ciekawe, wszystkie algorytmy wykorzystują w pewien sposób rekurencję.

2 Krótki opis algorytmów

W tej częsci przybliżę pokrótce każdy z algorytmów i postaram się podać jego mocne i słabe strony.

2.1 Quick Sort

Jest to obecnie najczęściej wykorzystywany algorytm sortowania. Średni czas wykonywania dla określonego rozmiaru problemu w porównaniu do innych algorytmów jest najmniejszy. Złożoność obliczeniowa w przypadku średnim wynosi $\mathcal{O}(n \log_2 n)$. Co ciekawe, przy pesymistycznym przypadku tj. dane posortowane odwrotnie, ma złożoność $\mathcal{O}(n^2)$. Pomimo tego, jest on jednym z najszybszych znanych algorytmów sortowania poprzez porównywanie. Dane dzielonę są na dwie części, w pierwszej części znajdują się elementy mniejsze od piwota, w drugiej większe. Podziału tego dokonujemy poprzez funkcję zwyczajowo nazywaną partition. Następnie wywołujemy rekurencyjnie funkcję osobno dla dwóch podtablic.

2.2 Merge Sort

Koncepcja sortowania przez scalanie jest bardzo prosta. Jeśli mamy już posortowane dwie tablice, to jedynym problemem jest połączenie je w jedną, dużą tablicę. Należy odpowiednio porównywać elementy z jednej tablicy i z drugiej, a następnie dodawać do tablicy wynikowej. Pojawia się pytanie „jak otrzymać posortowane już dwie tablice"? Wystarczy scalać tablice o rozmiarze 1. Algorytm ma złożoność $\mathcal{O}(n \log_2 n)$. Wadą tego algorytmu jest zapotrzebowanie na dodatkową pamięć.

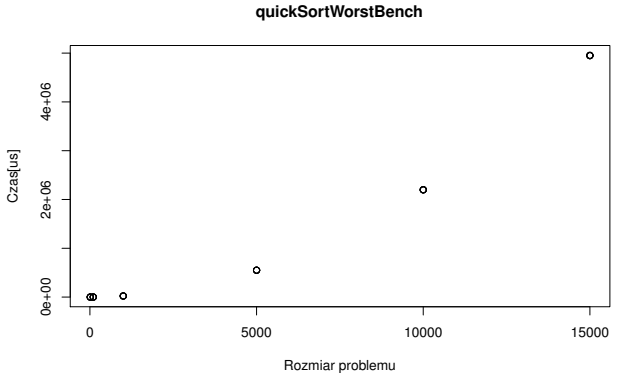
2.3 Heap Sort

Sortowanie przez kopcowanie, jak można się domyślić, wykorzystuje strukturę danych zwaną kopcem. Kopiec to drzewo binarne, w którym pomiędzy synem a ojcem występuje pewna relacja. Jeśli ojciec jest, większy od syna jest to kopiec typu maks, jeśli jest odwrotnie jest to kopiec typu min. Algorytm polega na zbudowaniu kopca, a następnie ściaganiu odpowiednich elementów. Złożoność obliczeniowa to $\mathcal{O}(n \log_2 n)$

3 Wyniki testów wydajnościowych

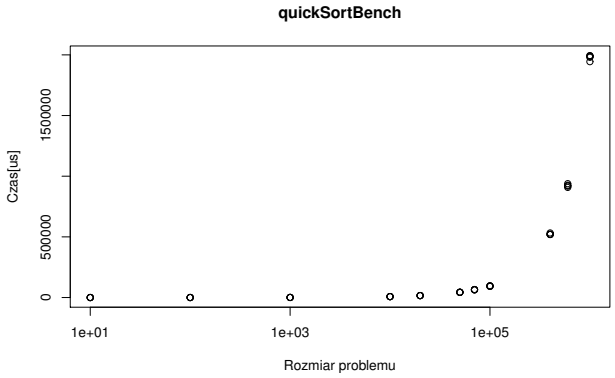
Poniżej zostały przedstawione wykresy zależności czasu wykonywania od rozmiaru problemu. Algorytmy zostały zbadane dla tych samych zbiorów danych. Jedynie w przypadku analizy sortowania szybkiego dla przypadku pesymistycznego użyto odrębnego zbioru.

3.1 Quick Sort - przypadek pesymistyczny



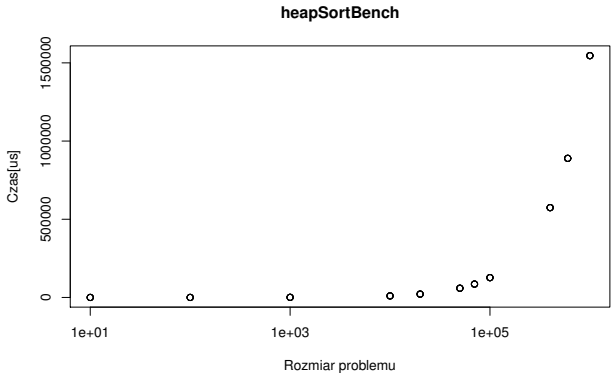
Rysunek 1: Zależność czasu wykonania od rozmiaru problemu - Quick Sort - przypadek pesymistyczny

3.2 Quick Sort - przypadek średni



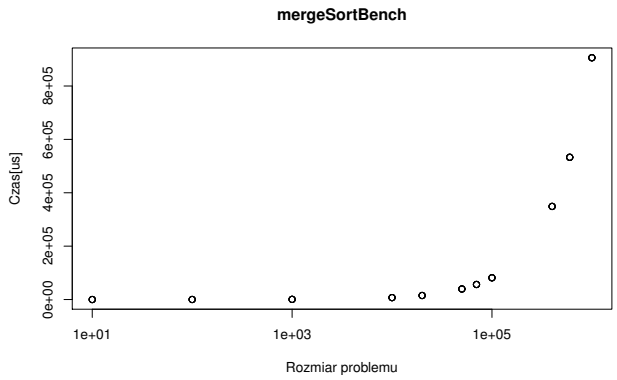
Rysunek 2: Zależność czasu wykonania od rozmiaru problemu - Quick Sort - przypadek średni

3.3 Heap Sort



Rysunek 3: Zależność czasu wykonania od rozmiaru problemu - Heap Sort

3.4 Merge Sort



Rysunek 4: Zależność czasu wykonania od rozmiaru problemu - Merge Sort

4 Wnioski i uwagi

- Aby pokazać złożoność algorytmów, w ostatnich trzech przypadkach użyłem skali logarytmicznej dla osi odciętych
- Widać, że quick sort jest bardzo wolny dla pesymistycznego przypadku
- Quick Sort nie był najszybszym algorytmem w tym teście nawet dla przypadku średniego, przyczyną może być wybór piwota, który jest zawsze ostatnim elementem tablicy
- Widać, że te trzy algorytmy mają złożoność $\mathcal{O}(\log_2 n)$, dla sortowania szybkiego tyczy się to przypadku średniego