Sprawozdanie - 3 sortowania w LaTeX-u QuickSort - wpływ wyboru pivotu na czas sortowania

Anna Pływaczyk, 200340

30.03.2014

1 Wstęp

Celem ćwiczenia jest zbadanie złożoności obliczeniowej algorytmów sortowania: przez scalanie, QuickSort i przez kopcowanie. Sortowanie szybkie ma zostać sprawdzone na dwa sposoby użycia pivotu:

- Pivot jako wartość losowa
- Pivot jako pierwszy element tablicy

Badania algorytmów zostały wykonane dla różnych typów danych wejściowych (liczby posortowane, losowe). Podane wartości pomiaru czasu są liczbami średnimi wykonanymi na podstawie kilku prób, aby pomiar czasu był bardziej dokładny i zbliżony do prawidłowego.
Pomiar czasu został zmierzony na podstawie różnicy aktualnego czasu systemowego po zakończeniu algorytmu, a czasem przed wykonaniem algorytmu sortowania.Podany wynik podany jest w milisekundach.
Pomiary zostały wykonane na systemie Windows, wyniki te mogły zostać przez system sfałszowane poprzez uruchomienie procesów w tle.

2 Sortowania

2.1 Sortowanie przez kopcowanie(heap)

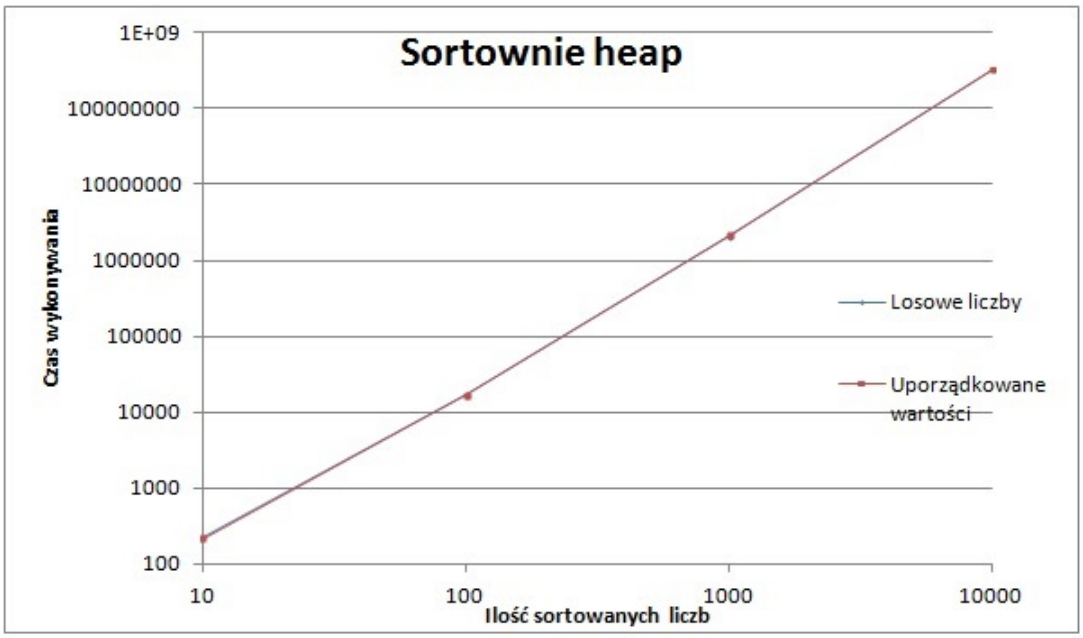
1. Działanie funkcji
   Sortowanie przez kopcowanie polega na utworzeniu kopca max, tzn. tablicy rozpatrywanej jako prawie pełne drzewo binarne, w którym rodzic zawsze jest większy lub równy od swoich dzieci. Czyli cały czas mamy dostęp do elementu maksymalnego, który znajduje się w korzeniu. Po odłożeniu go na koniec tablicy , wstawiamy w jego miejsce ostatni element znajdujący się na liście i przywracamy własności kopca (wartość w węźle

jest nie mniejsza niż u jego potomków). Powtarzając te czynności po kolei odkładamy największe elementy tworząc tablicę posortowaną. Złożoność obliczeniowa wynosi $O(nlogn)$
W praktyce algorytm sortowania przez kopcowanie jest nieco wolniejszy niż sortowanie szybkie.

2. Test sortowania

   Czasy zmierzone odpowiednio dla wartości:10, 100, 1000, 10000

   - Losowe dane: 226,2; 16948,1; 2131090; 327717000
   - Dane uporządkowane: 216,2; 16716,4; 2123260; 325362000
   - Wykres złożoności obliczeniowej.



Rysunek 1: Wykres pomiaru czasu od zadanej liczby próbek

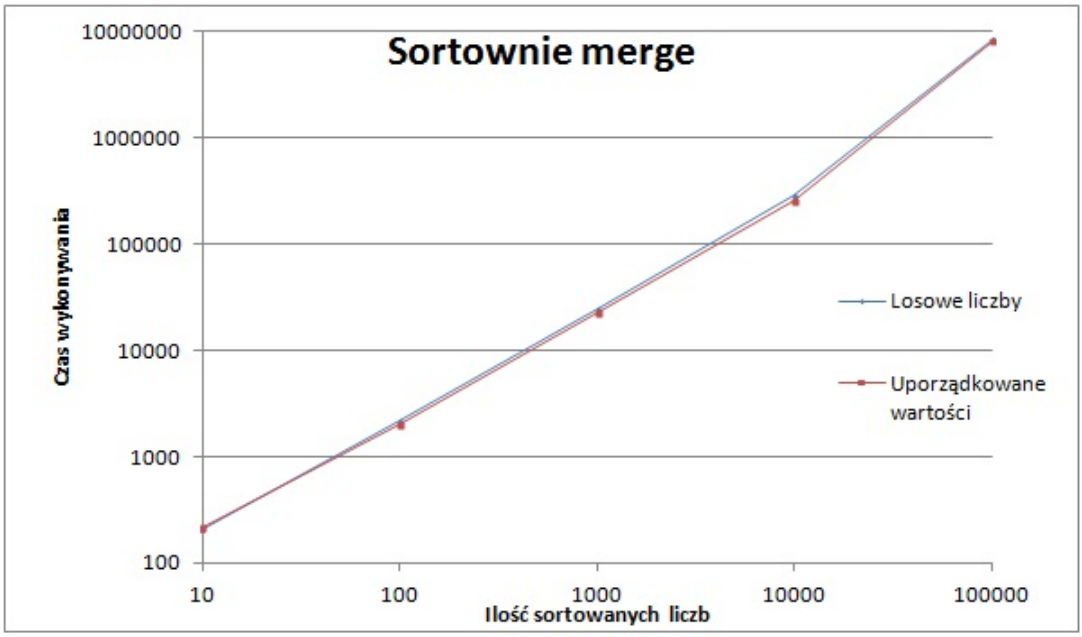
2.2 Sortowanie przez scalanie (merge)

1. Działanie funkcji
   Sortowanie przez scalanie należy do algorytmów, które wykorzystują założenie "dziel i zwyciężaj". Złożoność obliczeniowa algorytmu wynosi $O(nlogn)$. Głównym założeniem sortowania jest podzielenie ciągu liczb na dwie równe części, następnie każdą z nich uporządkowujemy i łączymy w jeden. Podzielone części sortujemy w sposób rekurencyjny wykorzystując ten sam algorytm dla każdego z podciągów. Musimy pamiętać aby określić, kiedy zaprzestajemy podziału ciągu.

2. Test sortowania

   Czasy zmierzone odpowiednio dla wartości: 10, 100, 1000, 10000, 100000

   - Losowe dane: 207,5; 2161,9; 24210,9; 276170; 8352300
   - Dane uporządkowane: 214; 2146,9; 22577; 277291; 8238310
   - Wykres złożoności obliczeniowej.



Rysunek 2: Wykres pomiaru czasu od zadanej liczby próbek

2.3 Sortowanie szybkie(quicksort)

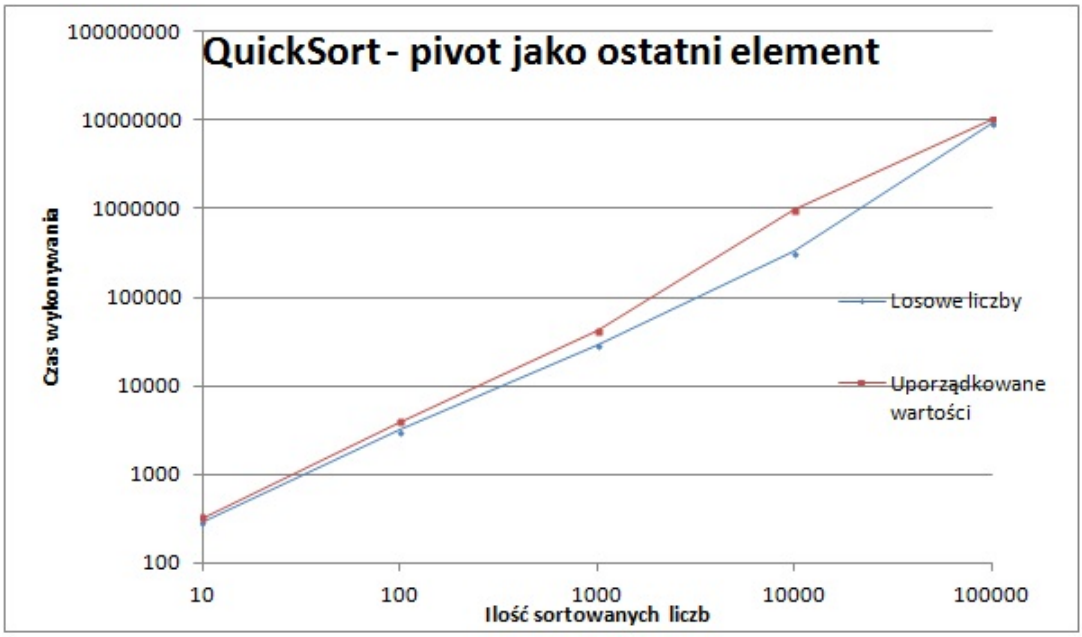
1. Działanie funkcji
   Jedno z najpopularniejszych sortowań wykorzystujących metodę dziel i zwyciężaj, w której dany problem dzielony jest na kilka mniejszych problemów. Następnie problemy te rozwiązywanie są rekurencyjnie, a rozwiązania wszystkich mniejszych problemów łączone są w celu utworzenia rozwiązania, czyli posortowaniu tablicy. W sortowaniu wyróżniamy 4 główne części programu.

   - Podzielenie ciągu liczb na dwie części względem pewnego ustalonego elementu (jest to ostatni element tablicy) tak aby na lewo od niego znajdowały się elementy mniejsze a na prawo większe.
   - Sortujemy elementy po lewej stronie.
   - Sortujemy elementy po prawej stronie.
   - Rekurencja kończy się gdy kolejny fragment zawiera pojedynczy element (jednoelementowa tablica nie wymaga sortowania).

2.3.1 Mediana jako ostatni argument tablicy

Test sortowania Czasy zmierzone odpowiednio dla wartości:10, 100, 1000, 10000, 100000

- Losowe dane: 292,9; 3139,7; 29049,5; 330319; 9240370
- Dane uporządkowane: 323; 3936; 42177; 961650; 10263410
- Wykres złożoności obliczeniowej.

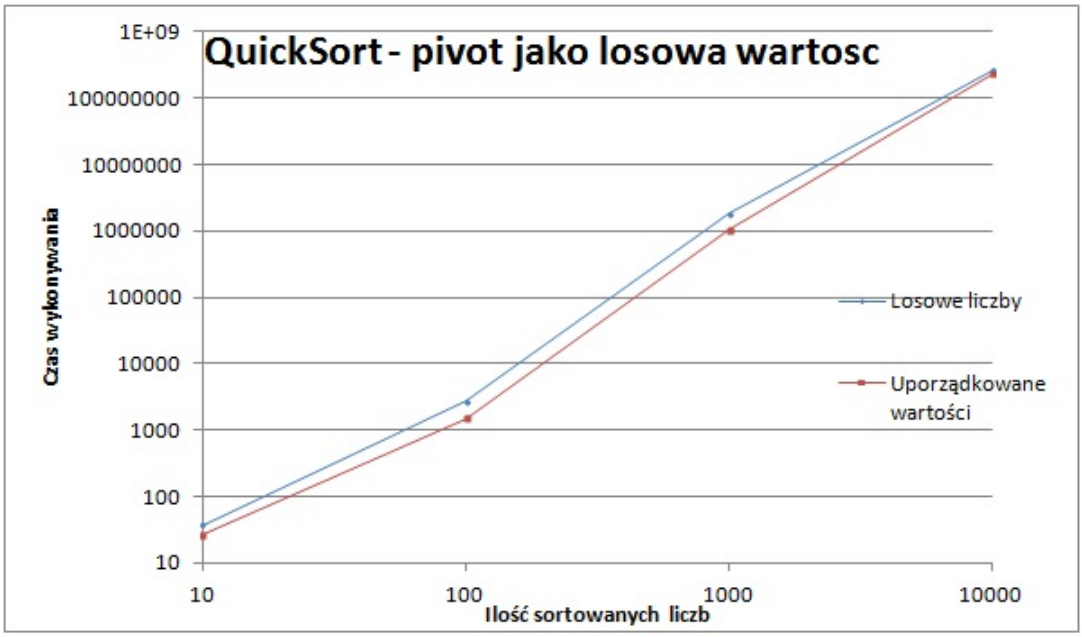


Rysunek 3: Wykres pomiaru czasu od zadanej liczby próbek

2.3.2 Mediana jako losowa wartość elementu tablicy

Test sortowania Czasy zmierzone odpowiednio dla wartości:10, 100, 1000, 10000, 100000

- Losowe dane: 37,8; 2677,2; 1799380; 262613100
- Dane uporządkowane: 26,7; 1521; 1022940; 232613100
- Wykres złożoności obliczeniowej.



Rysunek 4: Wykres pomiaru czasu od zadanej liczby próbek

3 Wnioski

Przy użyciu sortowania przez scalanie różnica jest niewidoczna, a wyniki w zależności od liczby elementów są różne. Nie możemy jednoznacznie określić jaki wpływ ma uporządkowanie danych na szybkość sortowania.
Dla sortowania przez kopcowanie zmniejszyłam liczbę elementów do 10000, gdyż przy tej liczbie widać jak szybko rośnie czas wykonywania algorytmu. W tym sortowaniu możemy zauważyć, iż również na wykresie nie ma znaczenia czy sortujemy liczby uporządkowane czy losowe. Natomiast w danych liczby te niewiele się od siebie różnią. Sortowanie liczb uporządkowanych poprzez kopcowanie jest szybsze niż losowych.

W sortowaniu szybkim gdy pivot jest pierwszym elementem wektora widać znacząca różnice przy sortowaniu liczb uporządkowanych bądź losowych. Gdy liczby są posortowane czas działania algorytmu znacznie się wydłuża. Natomiast przy losowym wyborze elementów algorytm szybciej sortuje uporządkowane dane. Dodatkowo zauważyć możemy, ze dla mniejszych danych metod ta szybciej wykonała sortowanie. Zastosowanie wyboru pivotu usprawnia działanie algorytmu.
Zauważyć możemy ze wszystkie algorytmy poza sortowaniem szybkim czas wykonywania działania maja zbliżony dla danych uporządkowanych i losowych.
Najdłużej wykonującym działanie okazał się algorytm sortowania przez kopco-

wanie, natomiast algorytm sortowania przez scalanie wykonał to najszybciej. Wywnioskować możemy, iż wybór metody ma znaczący wpływ na szybkość sortowania. Nawet przy tym samym typie sortowania a różnym wyborze medianę otrzymane wyniki mogą się różnic.