Porównanie algorytmów sortujących: InsertionSort, MergeSort oraz QuickSort

Łukasz Knigawka

28 października 2018

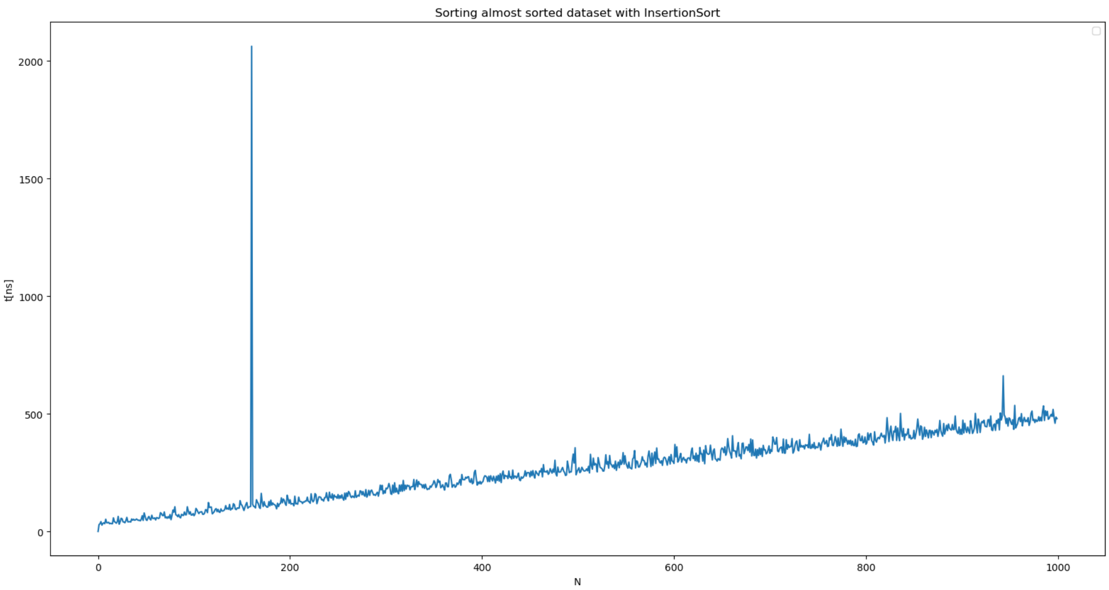
Spis treści

1 Wprowadzenie

Przeprowadziłem badania złożoności obliczeniowej trzech algorytmów sortowania: przez wstawienie, szybkiego i przez scalanie. Jako optymistyczne dane wejściowe przyjąłem ciąg już posortowany, jako pesymistyczne dane wejściowe przyjąłem ciąg posortowany w odwrotnej kolejności, a trzecim rodzajem ciągu wejściowego jest ciąg losowy.

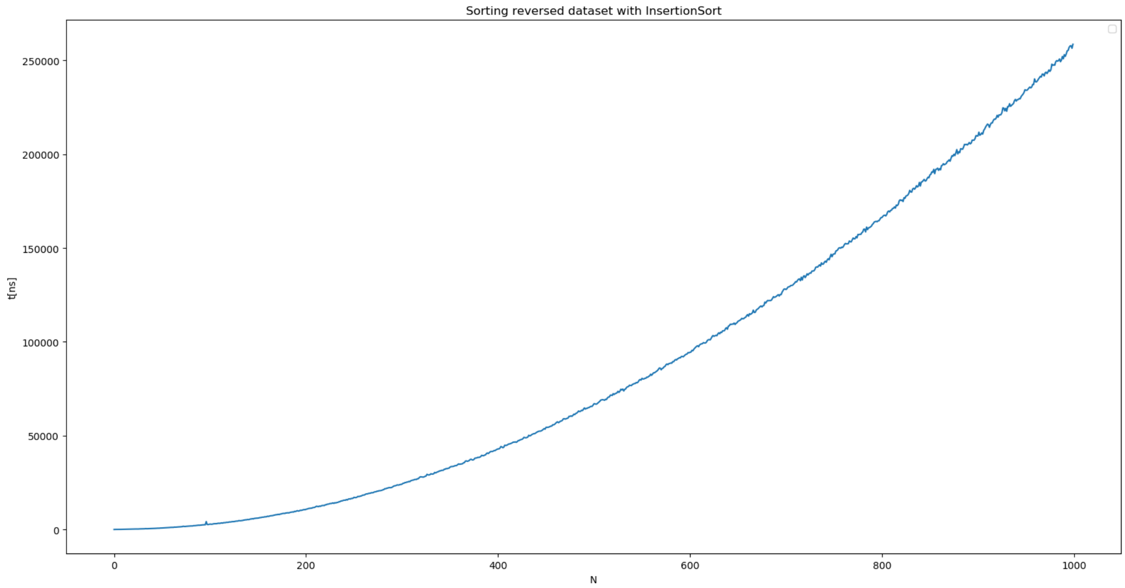
2 Wyniki badań algorytmu sortowania przez wstawianie

2.1 Wyniki dla optymistycznych danych wejściowych



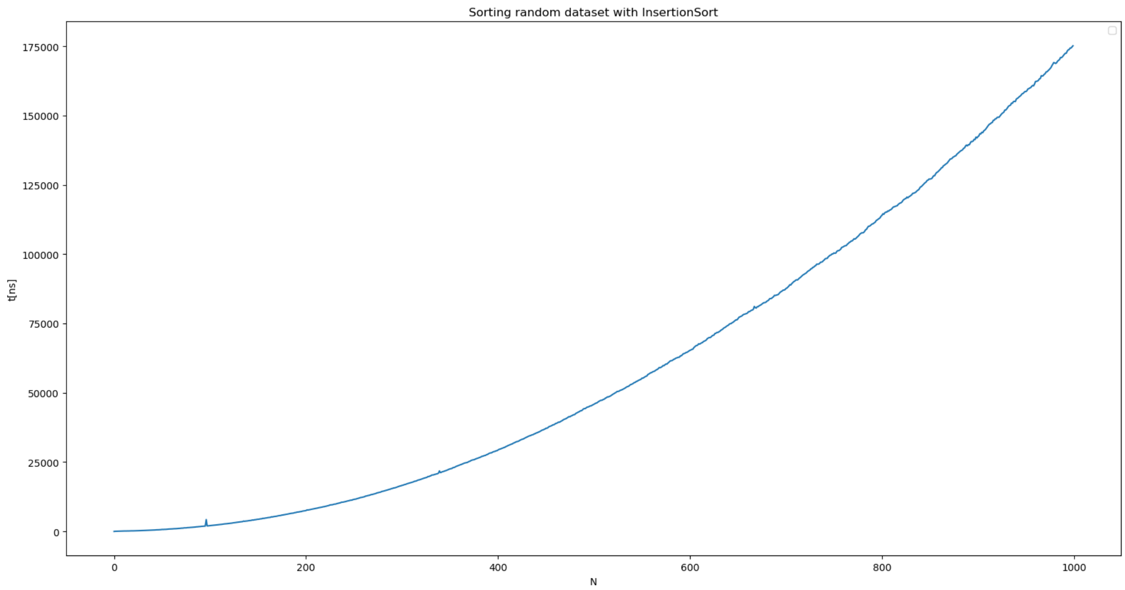
Rysunek 1: Sortowanie rosnącego ciągu wejściowego algorytmem InsertionSort jest bardzo efektywne. Otrzymujemy złożoność liniową.

2.2 Wyniki dla pesymistycznych danych wejściowych



Rysunek 2: Sortowanie malejącego ciągu wejściowego algorytmem InsertionSort. Otrzymujemy złożoność kwadratową.

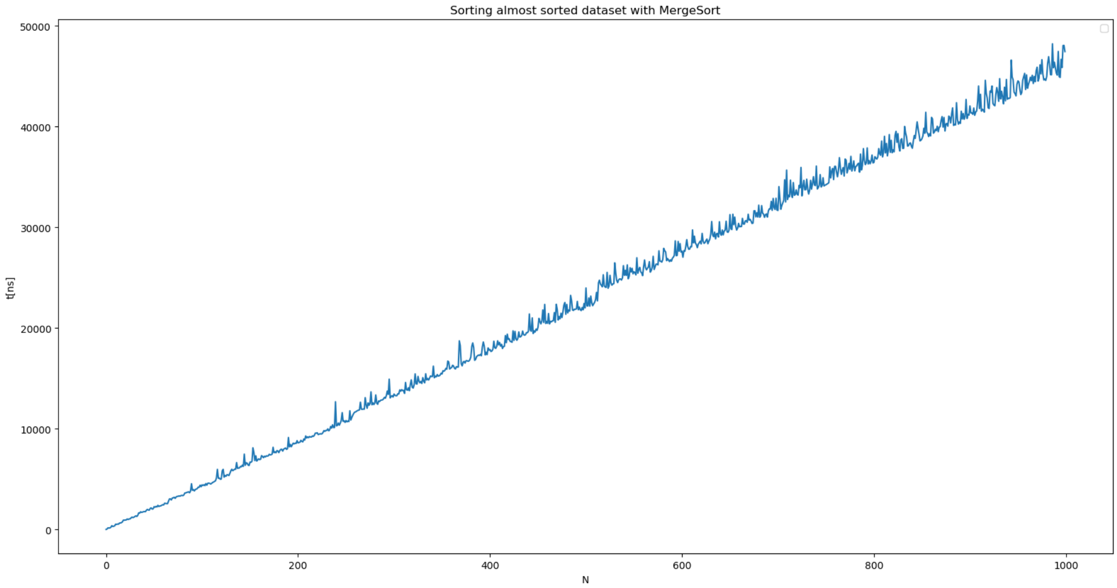
2.3 Wyniki dla pseudolosowych danych wejściowych



Rysunek 3: Sortowanie losowego ciągu wejściowego algorytmem InsertionSort. Otrzymujemy złożoność kwadratową. Sortowanie trwa jednak nieco krócej niż w przypadku danych pesymistycznych.

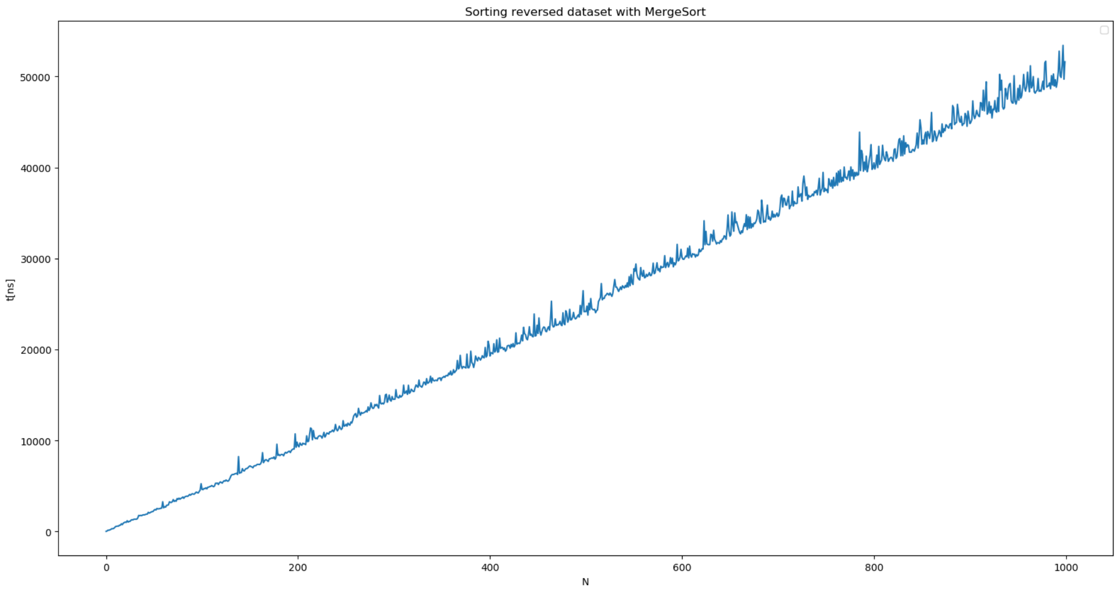
3 Wyniki badań algorytmu sortowania przez scalanie

3.1 Wyniki dla optymistycznych danych wejściowych



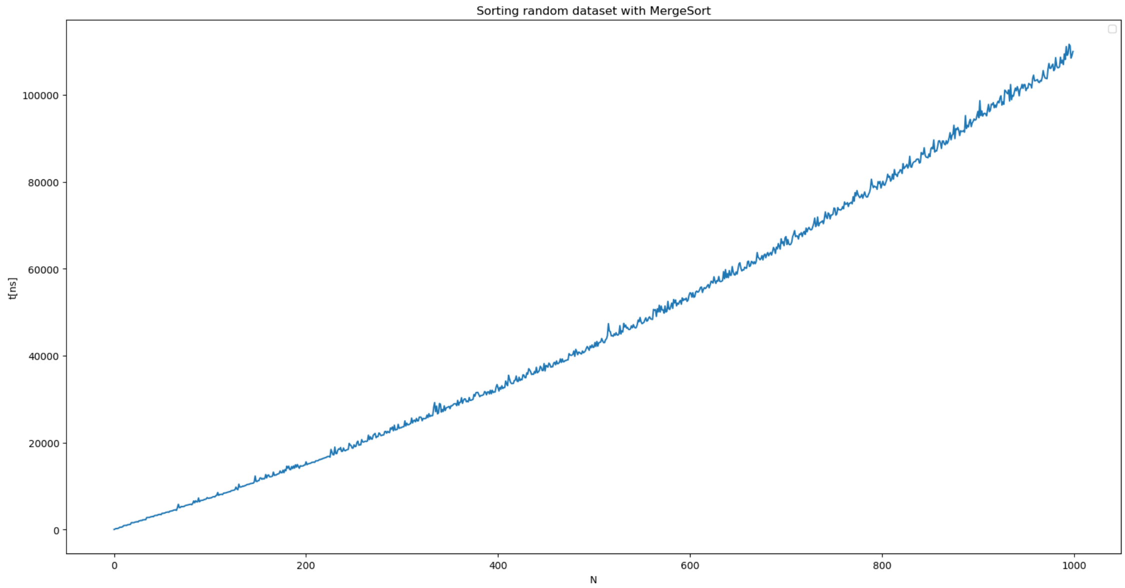
Rysunek 4: Sortowanie rosnącego ciągu wejściowego algorytmem MergeSort. Otrzymujemy złożoność *O(nlogn)*.

3.2 Wyniki dla pesymistycznych danych wejściowych



Rysunek 5: Sortowanie malejącego ciągu wejściowego algorytmem MergeSort. Otrzymujemy złożoność *O(nlogn)*. Czasy sortowania niewiele dłuższe niż dla praktycznie posortowanego ciągu.

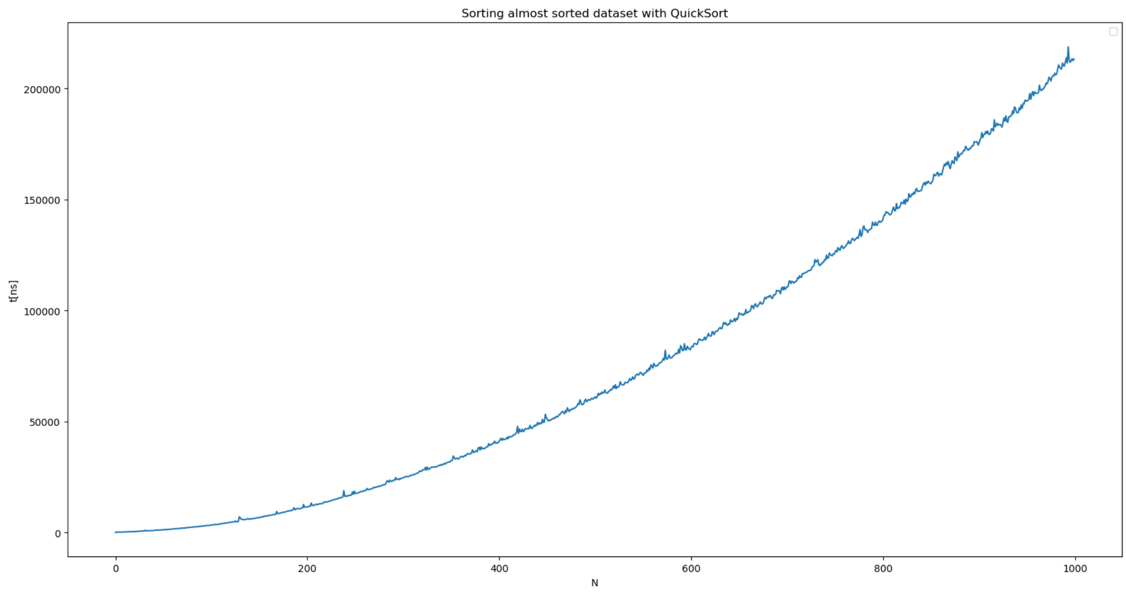
3.3 Wyniki dla pseudolosowych danych wejściowych



Rysunek 6: Sortowanie losowego ciągu wejściowego algorytmem MergeSort. Otrzymujemy złożoność *O(nlogn)*. Ten algorytm z ciągiem losowym radzi sobie znacznie gorzej niż z ciągiem posortowanym odwrotnie.

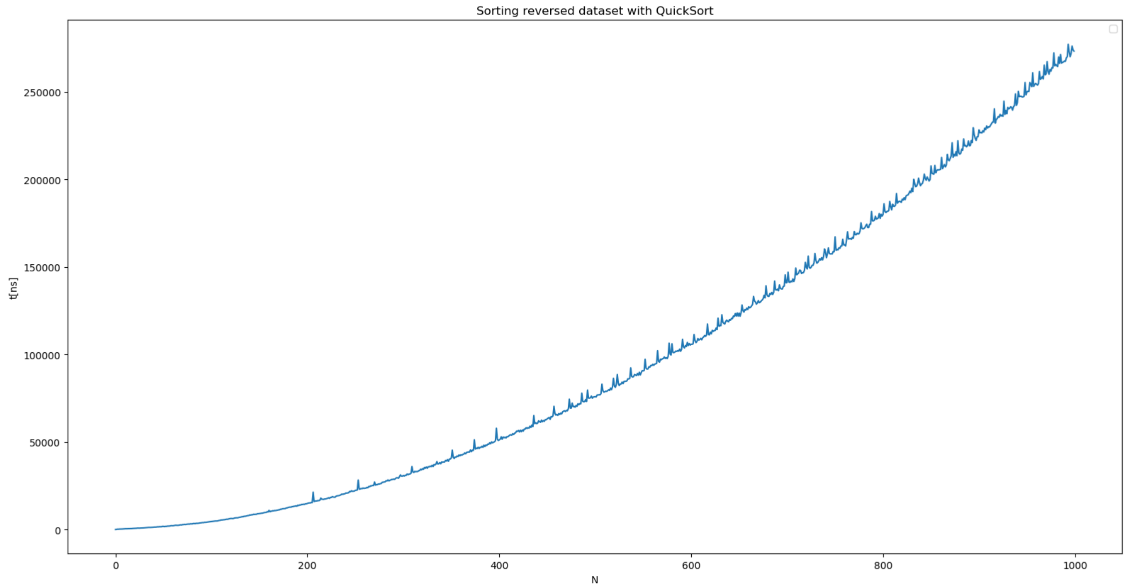
4 Wyniki badań algorytmu sortowania szybkiego

4.1 Wyniki dla optymistycznych danych wejściowych



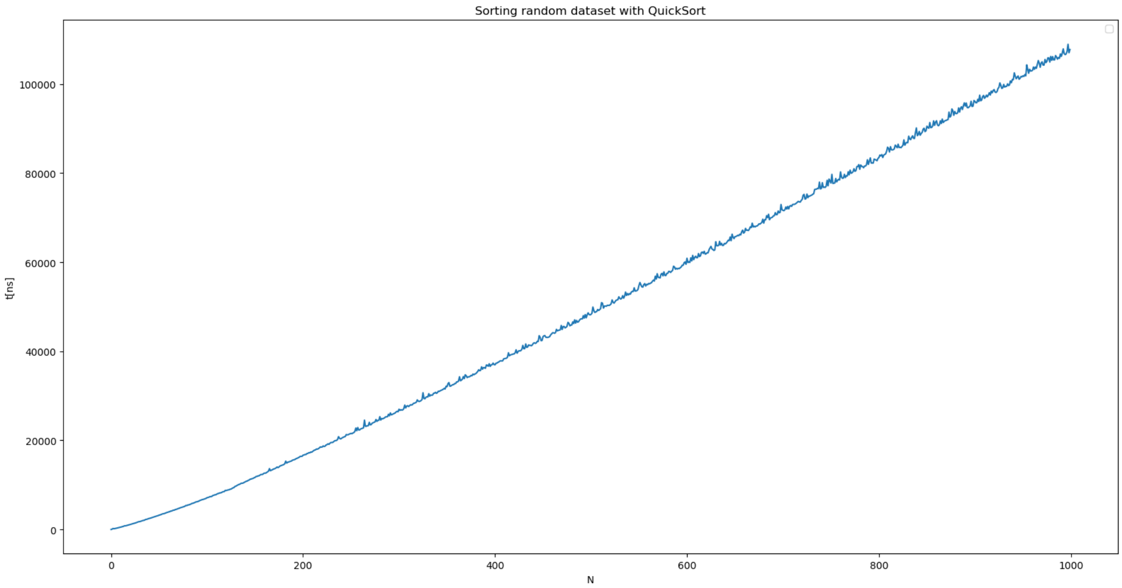
Rysunek 7: Sortowanie rosnącego ciągu wejściowego algorytmem QuickSort. Otrzymujemy złożoność *O(n^2)*. W tym wypadku dane które założyliśmy początkowo jako optymistyczne, okazują się być skrajnie pesymistyczne. Otrzymalibyśmy złożoność *O(nlogn)*, gdyby usprawnić algorytm mechanizmem sprytniejszego wybierania elementu dzielącego.

4.2 Wyniki dla pesymistycznych danych wejściowych



Rysunek 8: Sortowanie malejącego ciągu wejściowego algorytmem QuickSort. Otrzymujemy złożoność $O(n^2)$. Skuteczność sortowania odwróconego ciągu posortowanego zbliżona do skuteczności sortowania ciągu posortowanego.

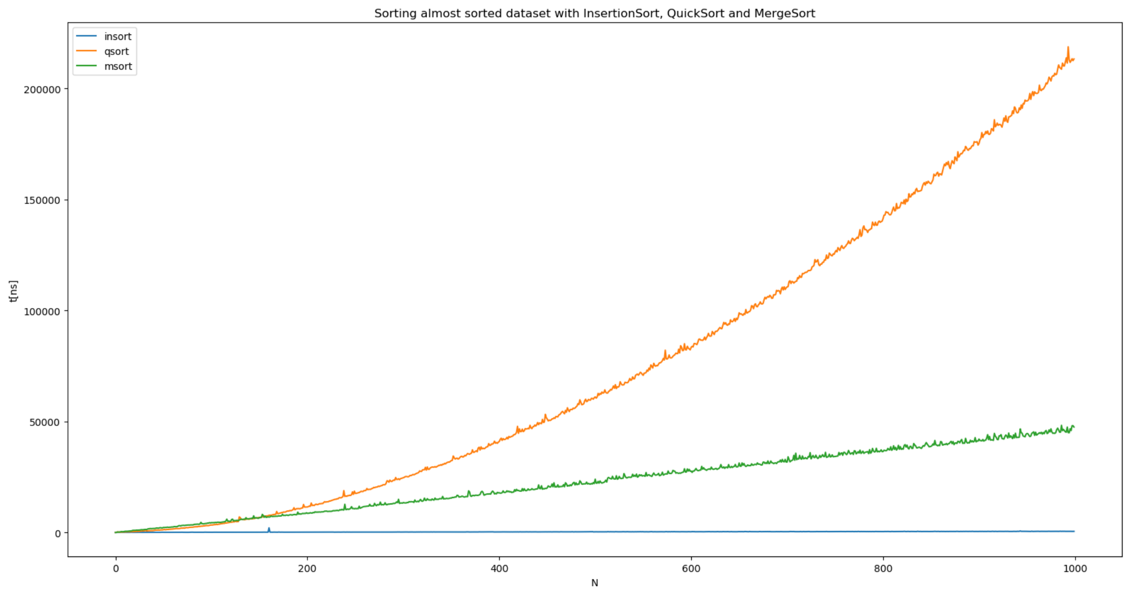
4.3 Wyniki dla pseudolosowych danych wejściowych



Rysunek 9: Sortowanie losowego ciągu wejściowego algorytmem QuickSort. Otrzymujemy złożoność *O(nlogn)*. Losowe dane okazały się znacznie bardziej przyjazne algorytmowi niż ciąg posortowany odwrotnie. W przypadku ciągów losowych, nie ma różnicy czy jako element dzielący weźmiemy element pierwszy, ostatni, środkowy czy jakikolwiek inny.

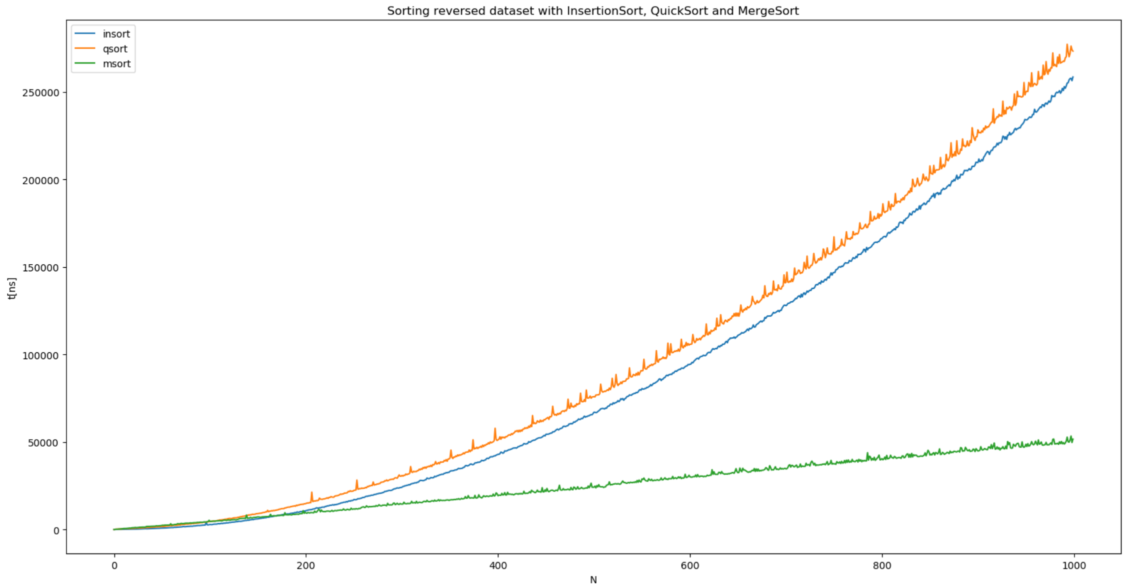
5 Porównanie wyników badań dla różnych algorytmów

5.1 Wyniki dla optymistycznych danych wejściowych



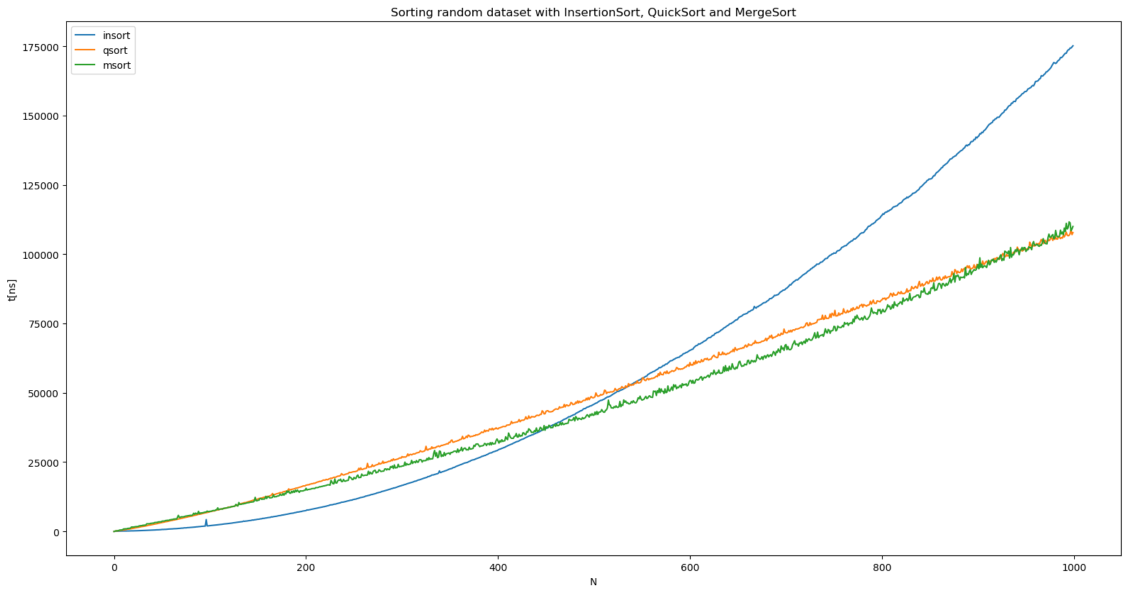
Rysunek 10: Dane określone wstępnie jako optymistyczne okazały się wyjątkowo pesymistyczne dla algorytmu sortowania szybkiego, jego złożoność odbiega więc od pozostałych algorytmów w tym przypadku. Wykres ładnie prezentuje złożoność liniową, kwadratową oraz *O(nlogn)*.

5.2 Wyniki dla pesymistycznych danych wejściowych



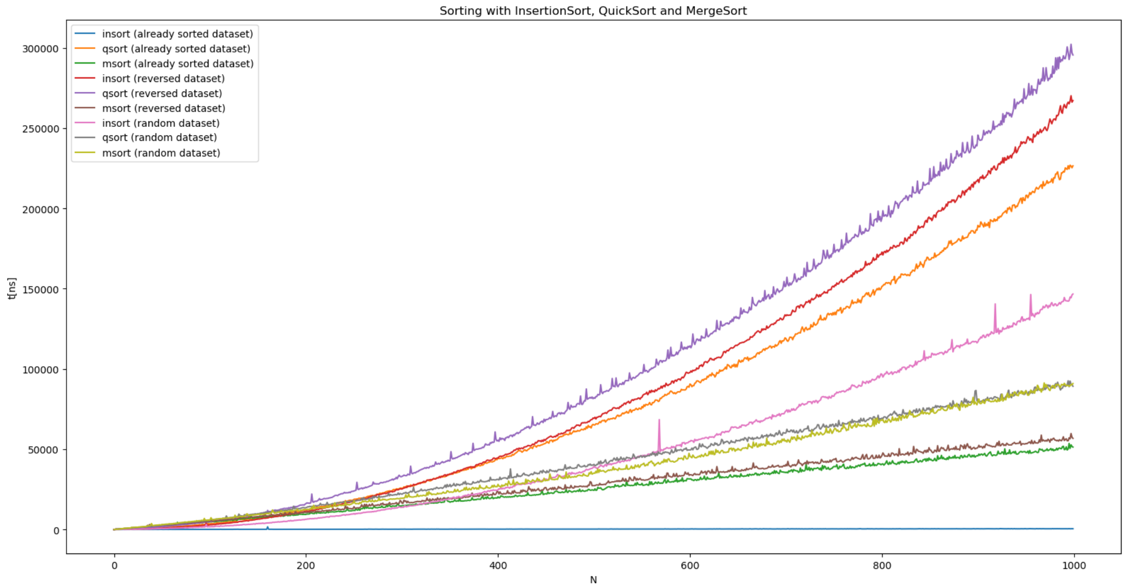
Rysunek 11: W tym przypadku drastycznie zmalała skuteczność algorytmu przez wstawianie, który przyjął kwadratową złożoność.

5.3 Wyniki dla pseudolosowych danych wejściowych



Rysunek 12: Najbardziej miarodajny wykres; widać, że dla małej liczby elementów w losowym ciągu warto wykorzystać algorytm sortowania przez wstawianie, dla większej liczby elementów jest on mało efektywny.

5.4 Zestawienie opracowanych danych an jednym wykresie



Rysunek 13: Próba przedstawienia wyników badania na jednym wykresie.

6 Wnioski

- Zaobserwowałem wysoką zbieżność wyników przeprowadzonych badań z teoretyczną złożonością obliczeniową algorytmów.
- Liczba porównań wykonywanych w algorytmie InsertionSort jest proporcjonalna do liczby inwersji w tablicy wejściowej, tzn. par takich, że a[i]>a[j] dla i<j. Jeżeli ciąg wejściowy jest prawie uporządkowany, tzn. gdy liczba inwersji jest $O(n)$, to złożoność algorytmu jest liniowa.
- W ramach modyfikacji mającej na celu usprawnienie algorytmu sortowania szybkiego, do sortowania danych o małych rozmiarach użyty może zostać algorytm sortowania przez wstawianie.
- W przeprowadzonych badaniach nie zdecydowałem się na skorzystanie z możliwych modyfikacji algorytmu sortowania szybkiego. Nie wykorzystuję algorytmu sortowania przez wstawianie, nie zastosowałem próbkowania losowego, a jako element dzielący wykorzystuję element spod niezmiennego indeksu. Wprowadzenie modyfikacji zmieniłoby niektóre złożoności.
- Bardzo ważny jest dobór elementu dzielącego (*ang. pivot*) w algorytmie sortowania szybkiego. W przeprowadzonych badaniach jako element dzielący zawsze wybierany był pierwszy element ciągu. W takie sytuacji zarówno dane nazwane wcześniej jako optymistyczne i pesymistyczne - czyli ciągi posortowane i posortowane odwrócone - prowadziły do uzyskania pesymistycznej złożoności.