Algorytmy i Struktury Danych

Krzysztof Wyszyński

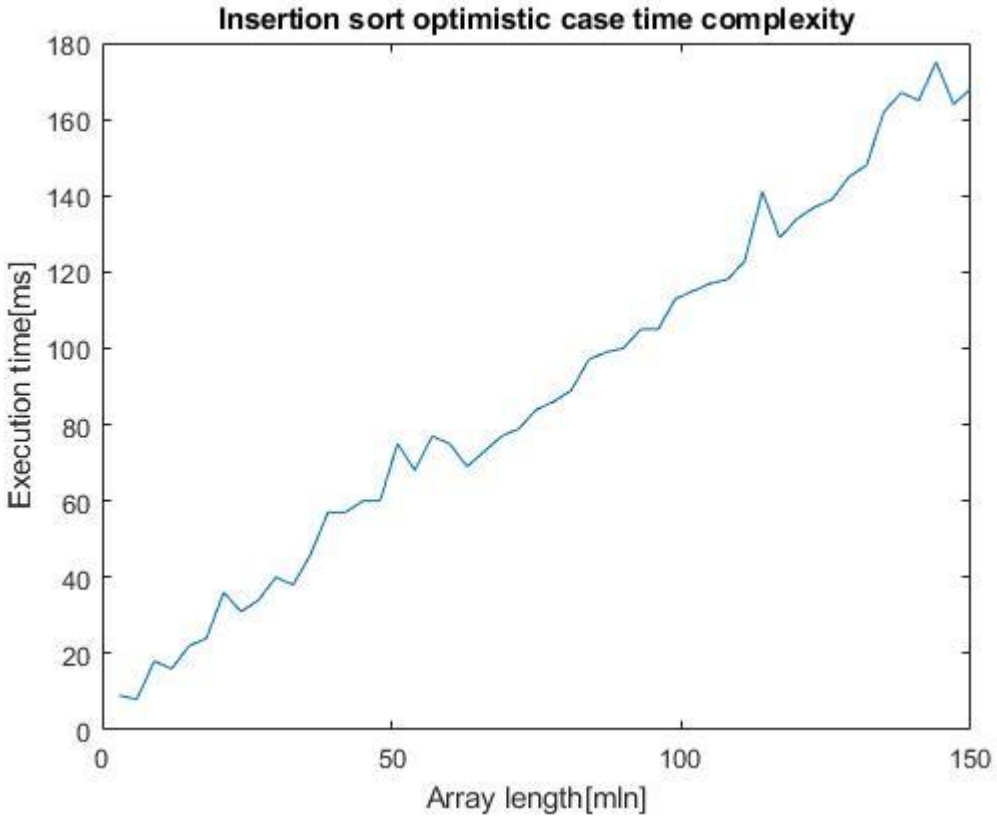
Porównanie złożoności obliczeniowych algorytmów sortowania

1. Insertion sort

   a) Złożoność optymistyczna

   Algorytm sortowania przez wstawianie przeszukuje tablicę w poszukiwaniu pierwszego mniejszego elementu niż aktualnie wskazywany. W związku z tym, jeśli tablica będzie już posortowana, algorytm przeiteruje jednokrotnie przez wszystkie jej elementy i zakończy działanie. Złożoność optymistyczna w tym przypadku wynosi zatem O(n).

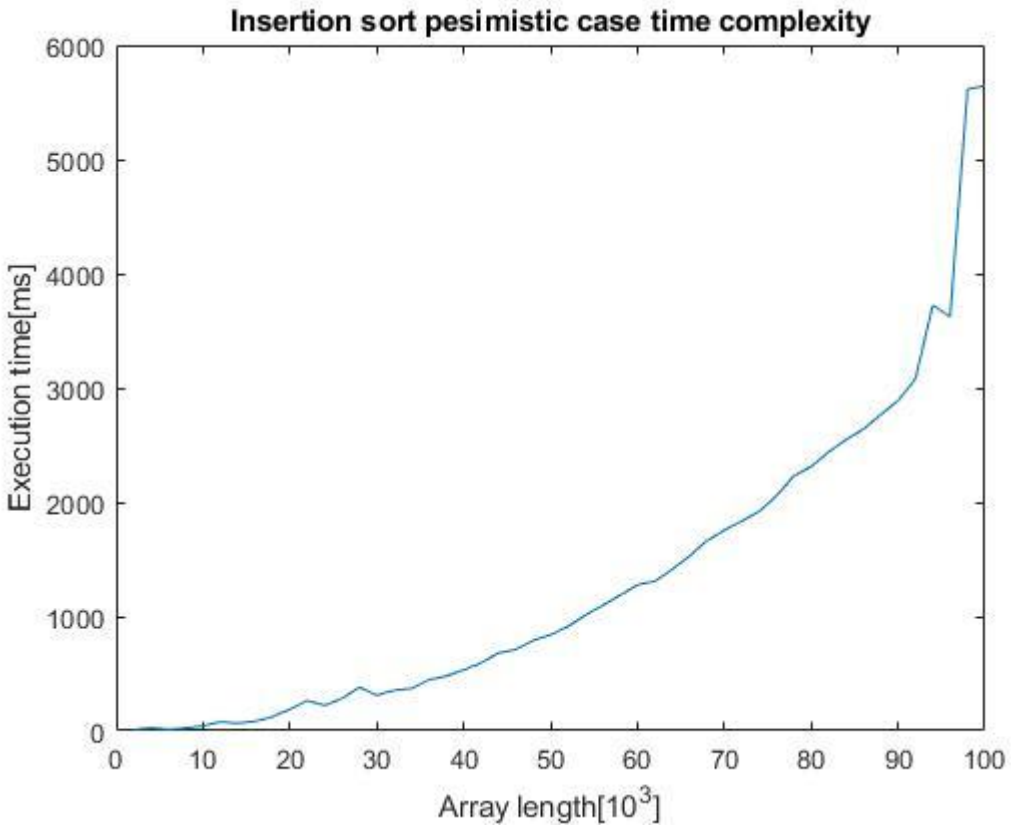
   W celu wyznaczenia złożoności czasowej wykorzystano 50 próbek – każda z nich to tablica posortowana, wypełniona liczbami [1…n], gdzie n to długość tablicy. Każda kolejna tablica jest większa od poprzedniej o 3 000 000 elementów.



   b) Złożoność pesymistyczna

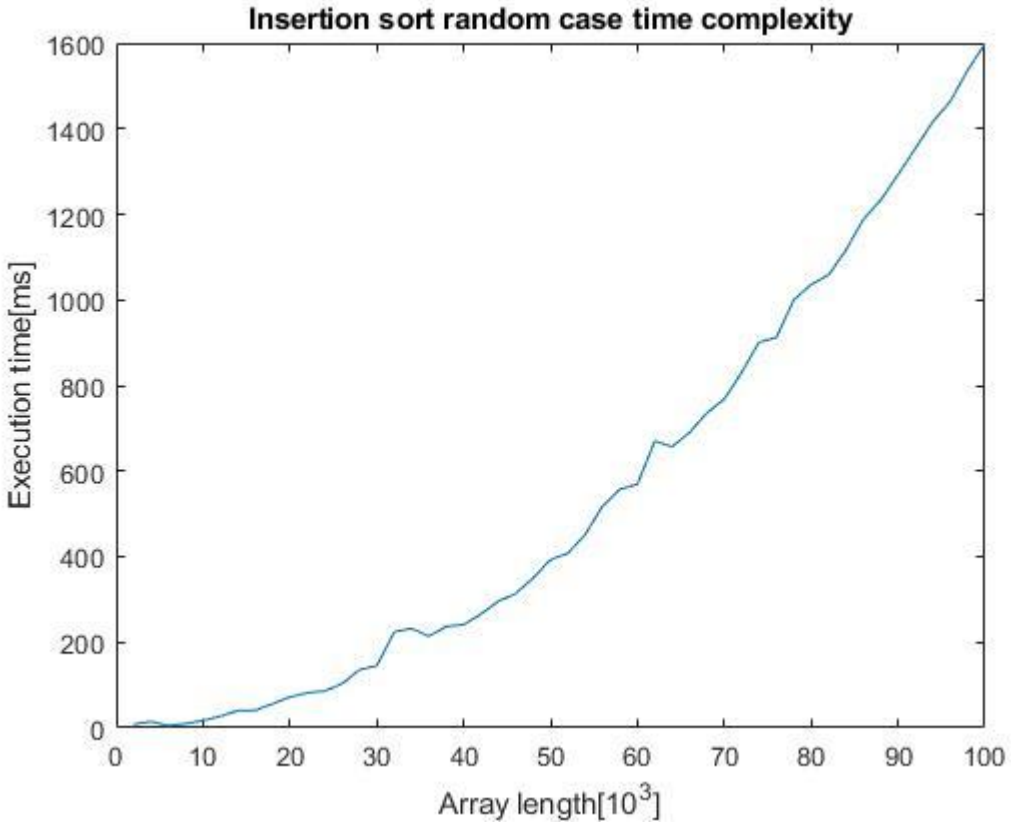
   Pesymistycznym przypadkiem dla algorytmu przez wstawianie jest sytuacja odwrotna – tablica posortowana malejąco. Wtedy złożoność czasowa algorytmu wynosi O(n^2).

W celu wyznaczenia złożoności czasowej wykorzystano 50 próbek – każda z nich to tablica posortowana malejąco, wypełniona liczbami [1…n], gdzie n to długość tablicy. W tym przypadku konieczne było ograniczenie rozmiarów tablic ze względu na czas wykonywania algorytmu. Każda kolejna tablica jest większa od poprzedniej o 2000 elementów.

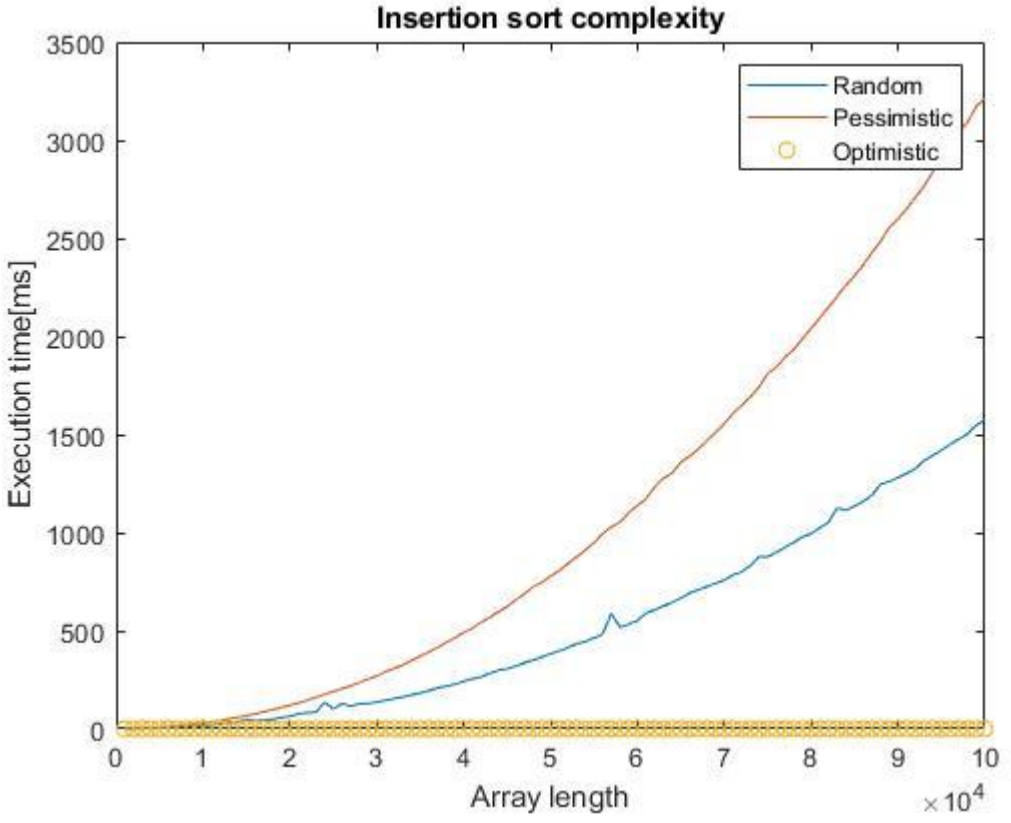


c) Zbiór danych losowych

Zbiór danych losowych składa się z 50 próbek – każda z nich to tablica wypełniona liczbami pseudolosowymi. Każda kolejna tablica jest większa od poprzedniej o 2000 elementów.



d) Wszystko w jednym



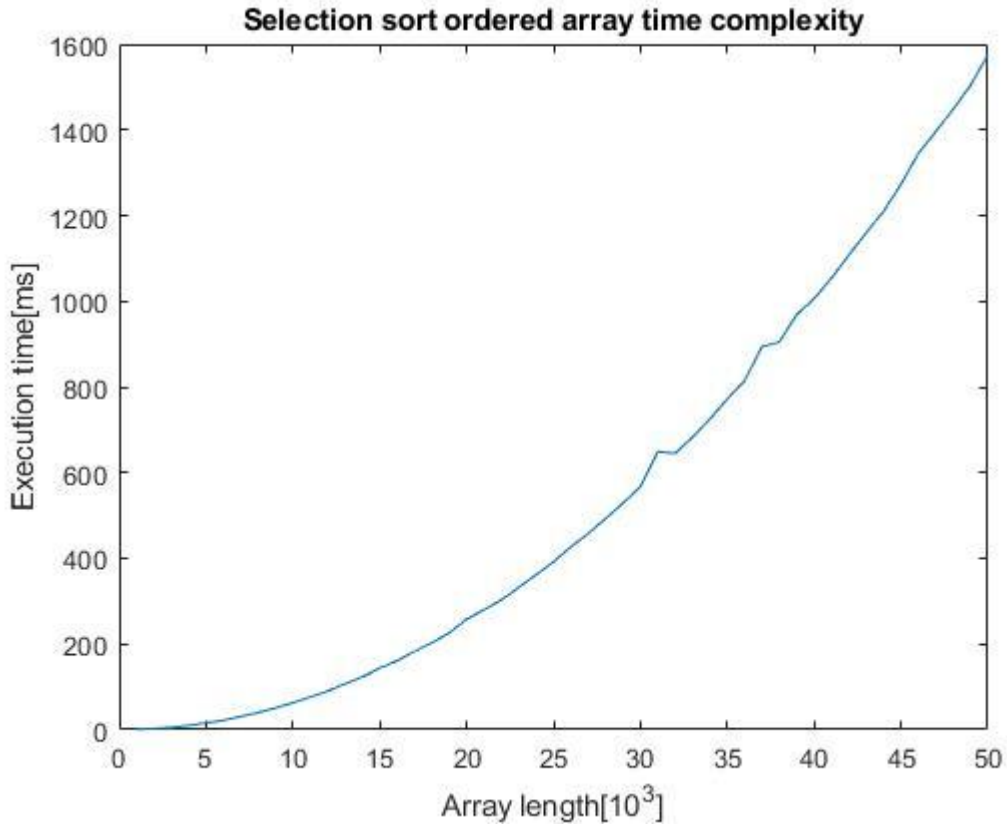
2. Selection sort

Złożoność czasowa algorytmu sortowania przez wybieranie jest niezależna od rodzaju danych zawartych w tablicy. Algorytm w każdym przypadku wybiera element tablicy począwszy od pierwszego, a następnie przemieszcza się po podtablicy z wyłączeniem wybranego elementu, w poszukiwaniu najmniejszej wartości mniejszej niż aktualnie wybrana. W związku z tym złożoność czasowa algorytmu w każdym przypadku wynosi O(n^2) – czas wykonywania algorytmu zależy jedynie od długości tablicy.

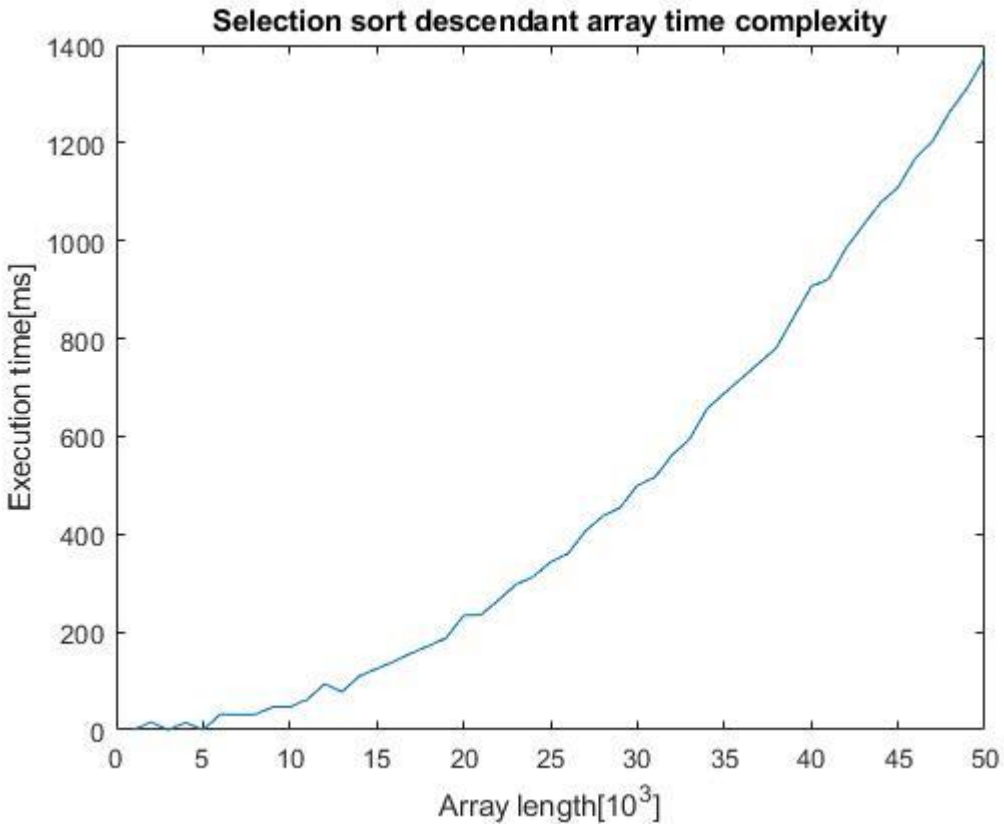
Aby udowodnić powyższą tezę przeprowadzone zostaną trzy eksperymenty.

a) Dane posortowane rosnąco

W celu wyznaczenia złożoności czasowej wykorzystano 50 próbek – każda z nich to tablica posortowana rosnąco, wypełniona liczbami [1…n], gdzie n to długość tablicy. Każda kolejna tablica jest większa od poprzedniej o 1000 elementów.



b) Dane posortowane malejąco

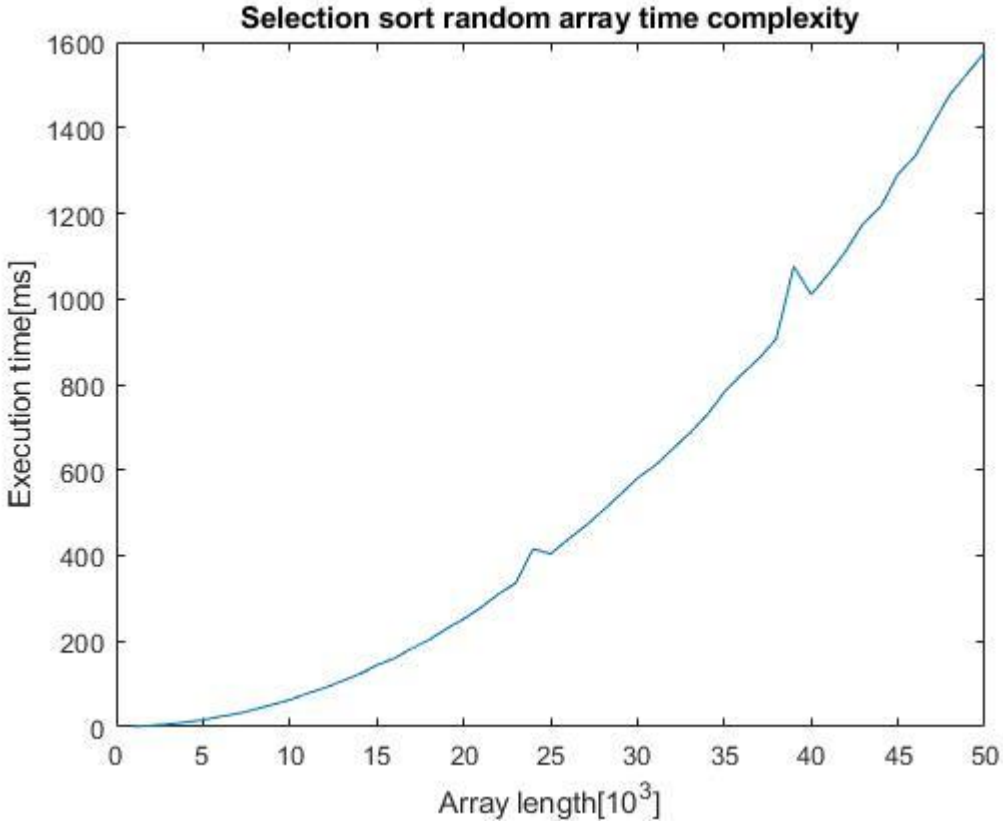


W celu wyznaczenia złożoności czasowej wykorzystano 50 próbek – każda z nich to tablica posortowana malejąco, wypełniona liczbami [1…n], gdzie n to długość tablicy. Każda kolejna tablica jest większa od poprzedniej o 1000 elementów.

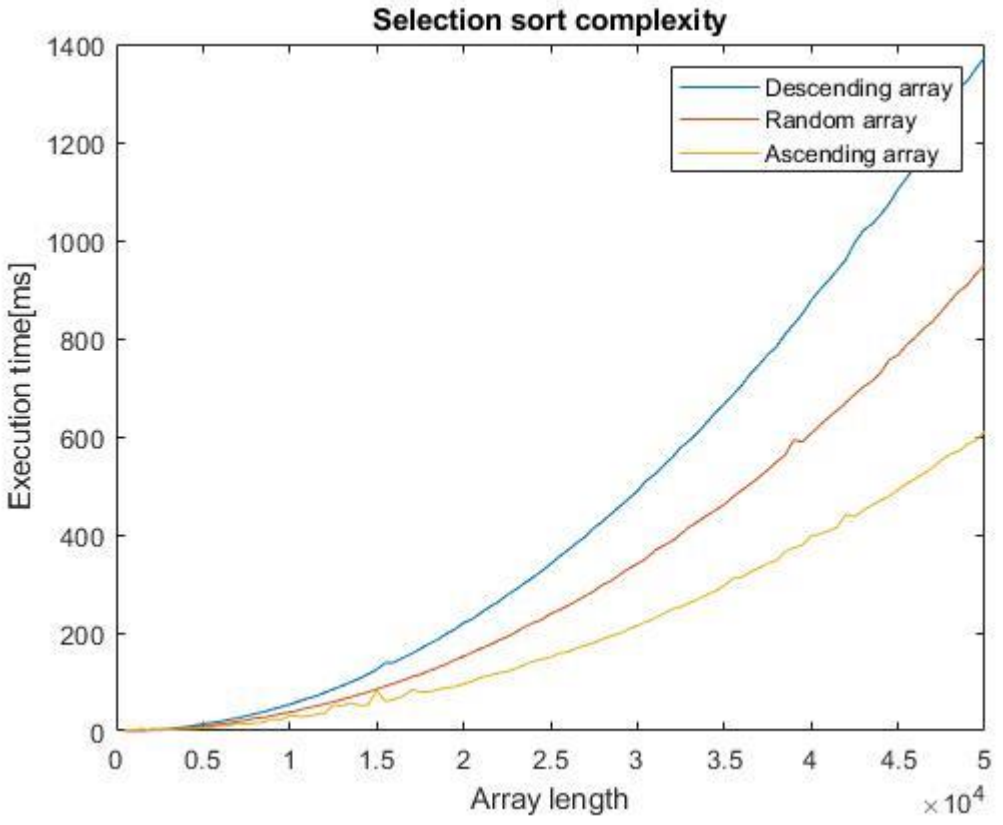
c) Zbiór danych losowych

Zbiór danych losowych składa się z 50 próbek – każda z nich to tablica wypełniona liczbami pseudolosowymi. Każda kolejna tablica jest większa od poprzedniej o 1000

elementów.



d) Wszystko w jednym



3. Quicksort

Algorytm sortowania szybkiego opiera swoje działanie na rekurencyjnym podziale tablicy na podtablice, poprzez rekurencyjne umieszczanie elementów mniejszych niż wybrany po lewej, a większych po prawej stronie tablicy. Wybór elementu rozdzielającego ma istotny wpływ na złożoność czasową algorytmu.

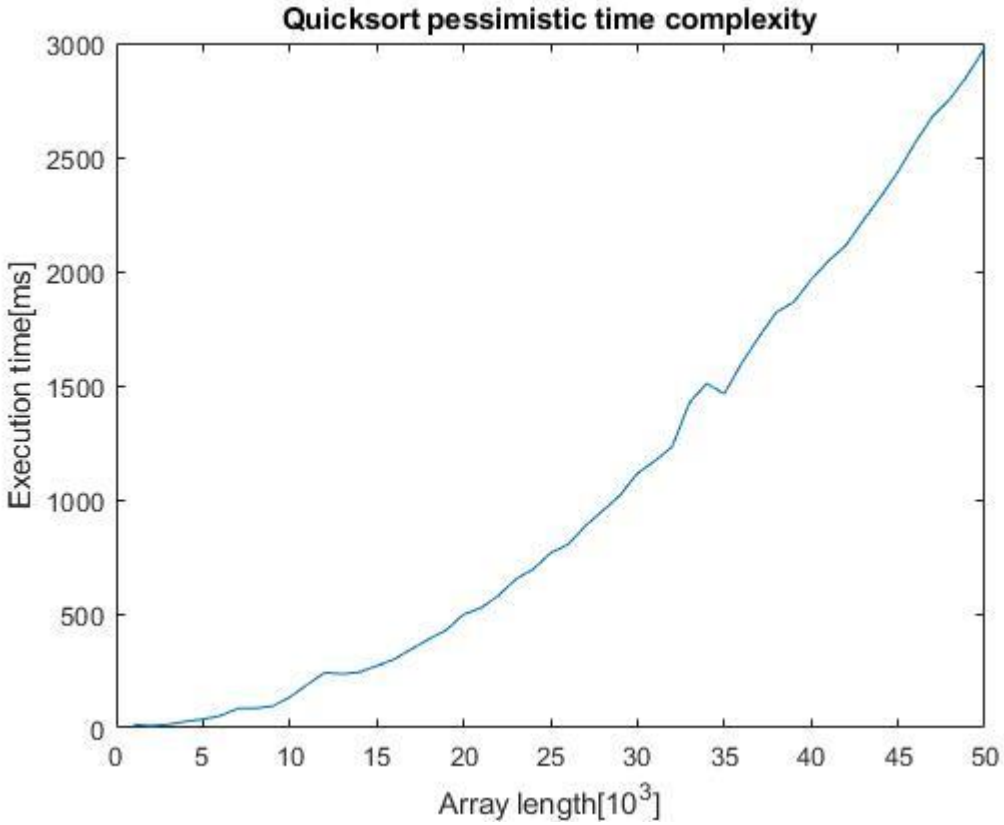
a) Złożoność optymistyczna

W ramach laboratorium początkowym elementem rozdzielającym jest pierwszy element sortowanej tablicy. Przypadkiem optymistycznym jest tablica, w której „pivot" znajduje się zawsze dokładnie pomiędzy początkiem, a końcem każdej podtablicy. Wtedy złożoność algorytmu wynosi O(nlogn). Niestety wygenerowanie takiego zbioru danych nie jest łatwym zadaniem. Z tego powodu wykres w tym przypadku zostanie pominięty.

b) Złożoność pesysmityczna

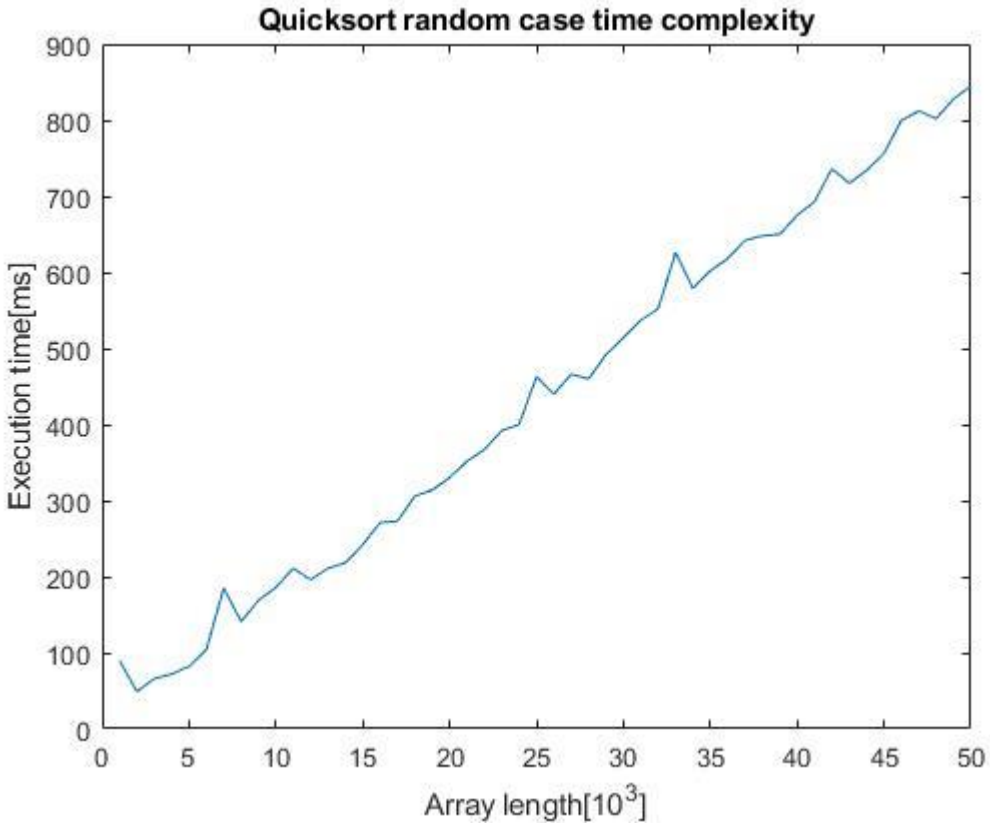
Przypadkiem pesysmistycznym jest sytuacja, w której „pivot" jest zawsze skrajnym elementem głównej tablicy oraz podtablic. W tym przypadku złożoność czasowa algorytmu wynosi O(n^2). Takim przykładem może być tablica już posortowana rosnąco bądź malejąco.

W celu wyznaczenia złożoności czasowej wykorzystano 50 próbek – każda z nich to tablica posortowana rosnąco, wypełniona liczbami [1…n], gdzie n to długość tablicy. Każda kolejna tablica jest większa od poprzedniej o 2000 elementów.

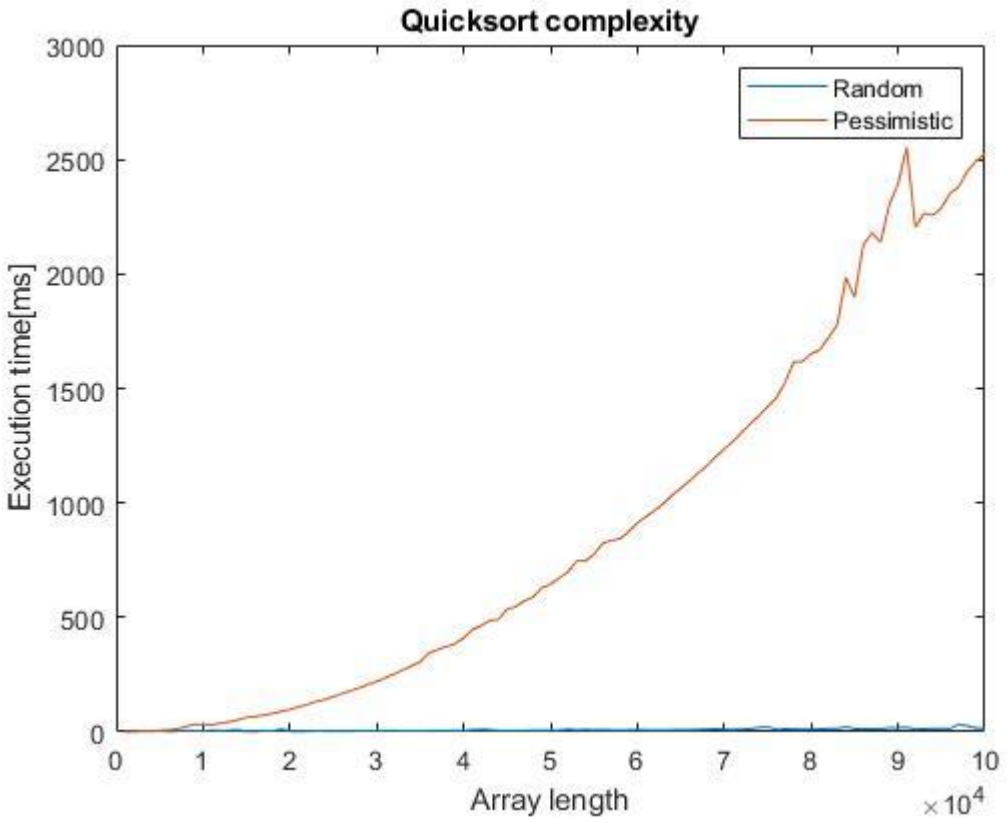


c) Zbiór danych losowych

Zbiór danych losowych składa się z 50 próbek – każda z nich to tablica wypełniona liczbami pseudolosowymi. Każda kolejna tablica jest większa od poprzedniej o 100 000 elementów.



d) Wszystko w jednym



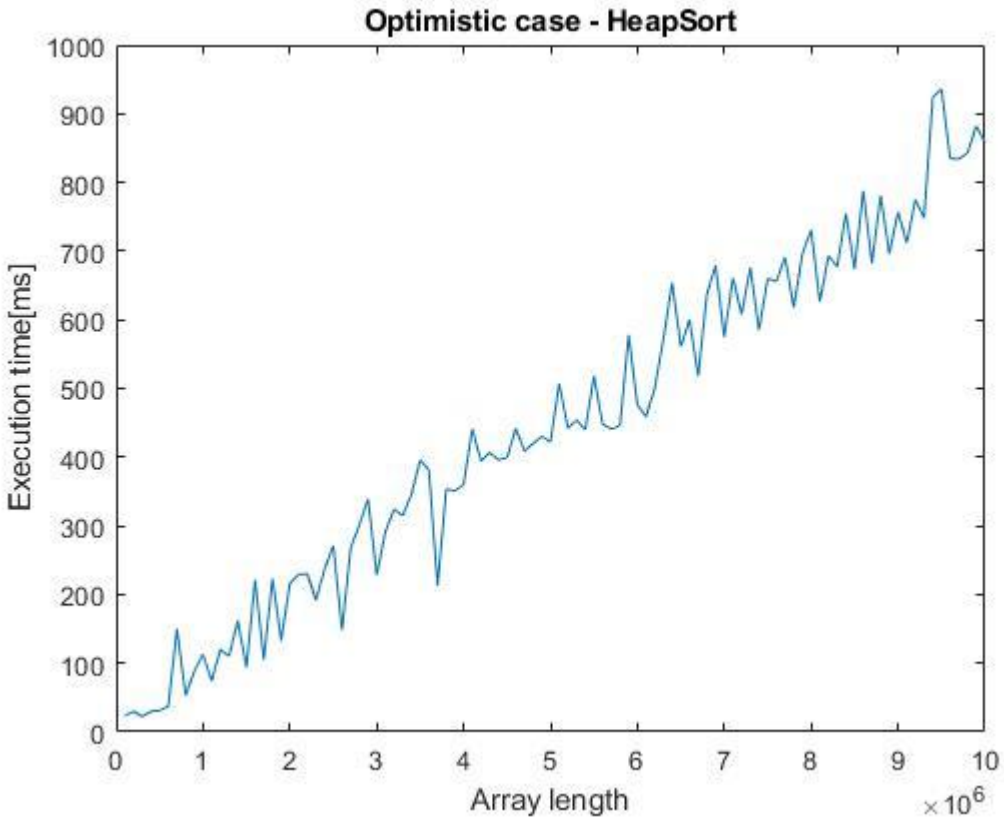
4. Heap sort

Algorytm sortowania przez kopcowanie wykorzystuje operacje wstawiania i wyjmowania kolejnych elementów z kopca.

a) Złożoność optymistyczna

Optymistycznym przypadkiem dla algorytmu sortowania przez kopcowanie jest sytuacja, kiedy wszystkie elementy sortowanej tablicy są sobie równe. Wtedy złożoność czasowa wynosi O(n).

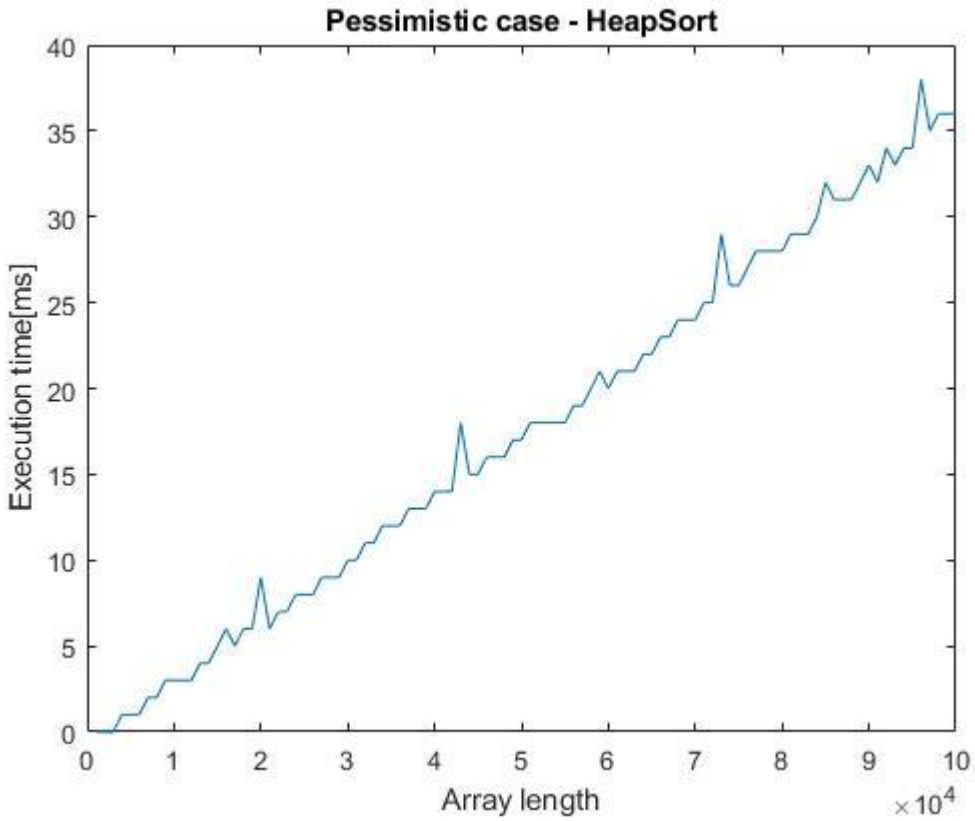
W celu wyznaczenia złożoności czasowej wykorzystano 100 próbek – każda z nich to tablica wypełniona liczbami o wartości 1. Każda kolejna tablica jest większa od poprzedniej o 100 000 elementów, począwszy od 100 000.



b) Złożoność pesymistyczna

Algorytm sortowania przez kopcowanie ma złożoność pesymistyczną O(nlogn). Występuje ona w każdym przypadku tablicy, z wyjątkiem wcześniej wspomnianego przypadku optymistycznego.

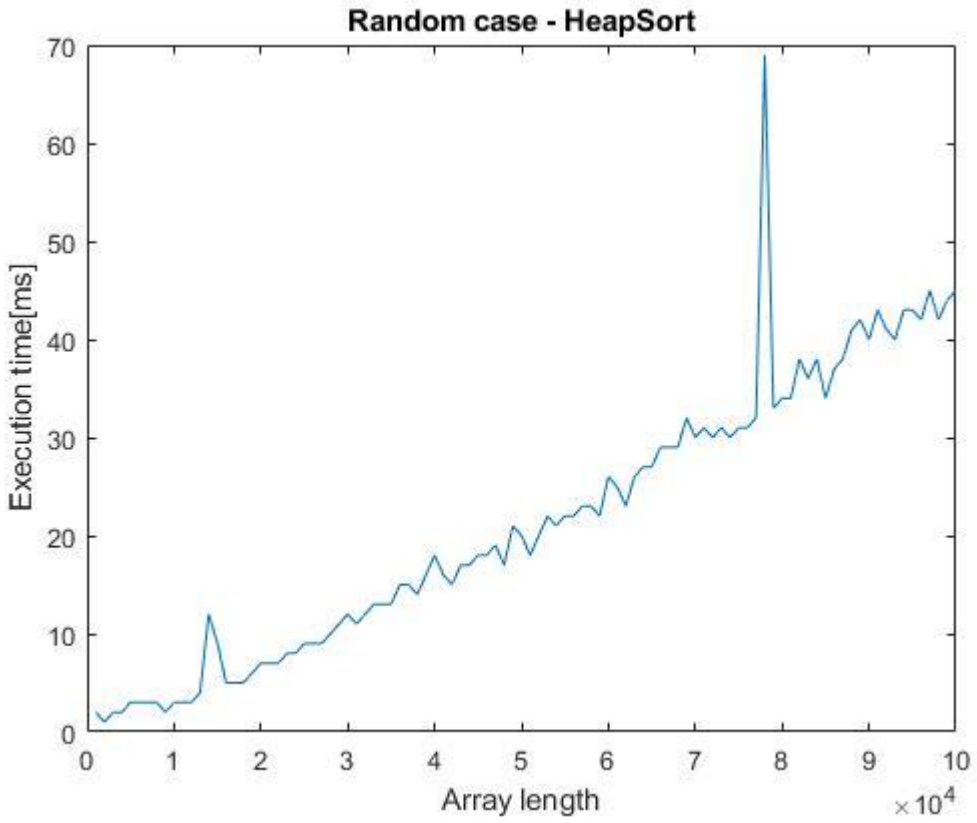
W celu wyznaczenia złożoności czasowej wykorzystano 100 próbek – każda z nich to tablica wypełniona rosnąco. Każda kolejna tablica jest większa od poprzedniej o 10000 elementów, począwszy od 10000.



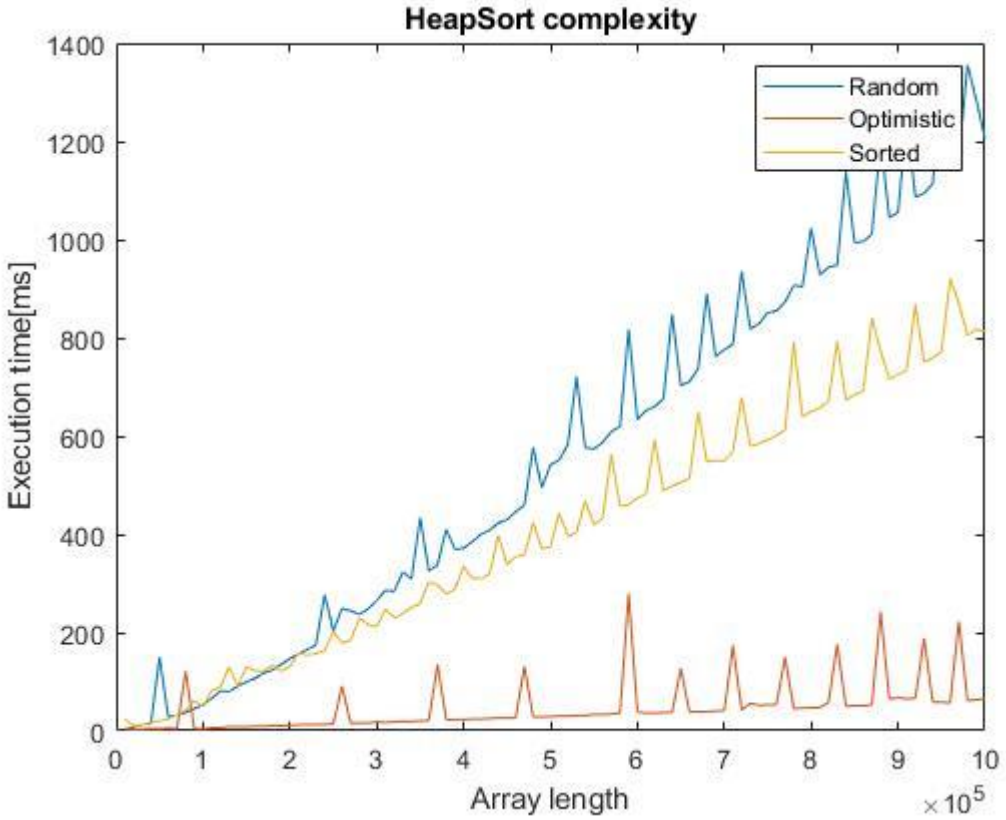
Powyższy wykres potwierdza, że złożoność pesymistyczna algorytmu sortowania przez kopcowanie jest rzędu O(nlogn).

c) Zbiór danych losowych

W celu wyznaczenia złożoności czasowej wykorzystano 100 próbek – każda z nich to tablica wypełniona liczbami pseudolosowymi. Każda kolejna tablica jest większa od poprzedniej o 1000 elementów, począwszy od 1000



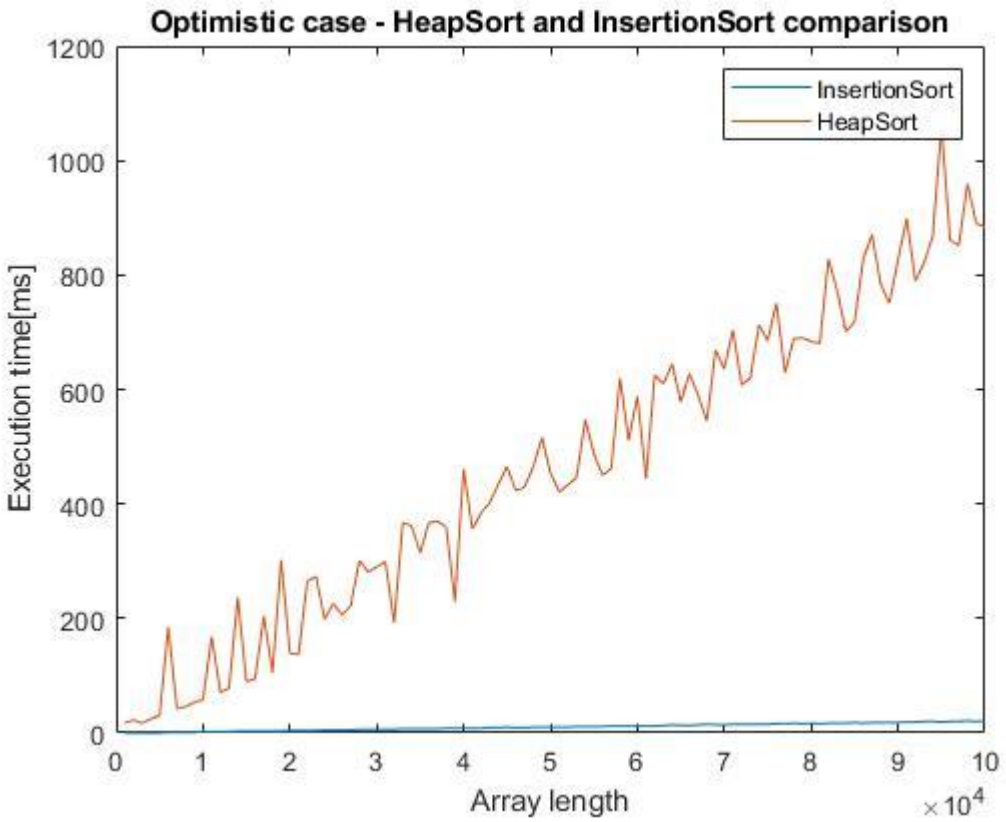
d) Wszystko w jednym



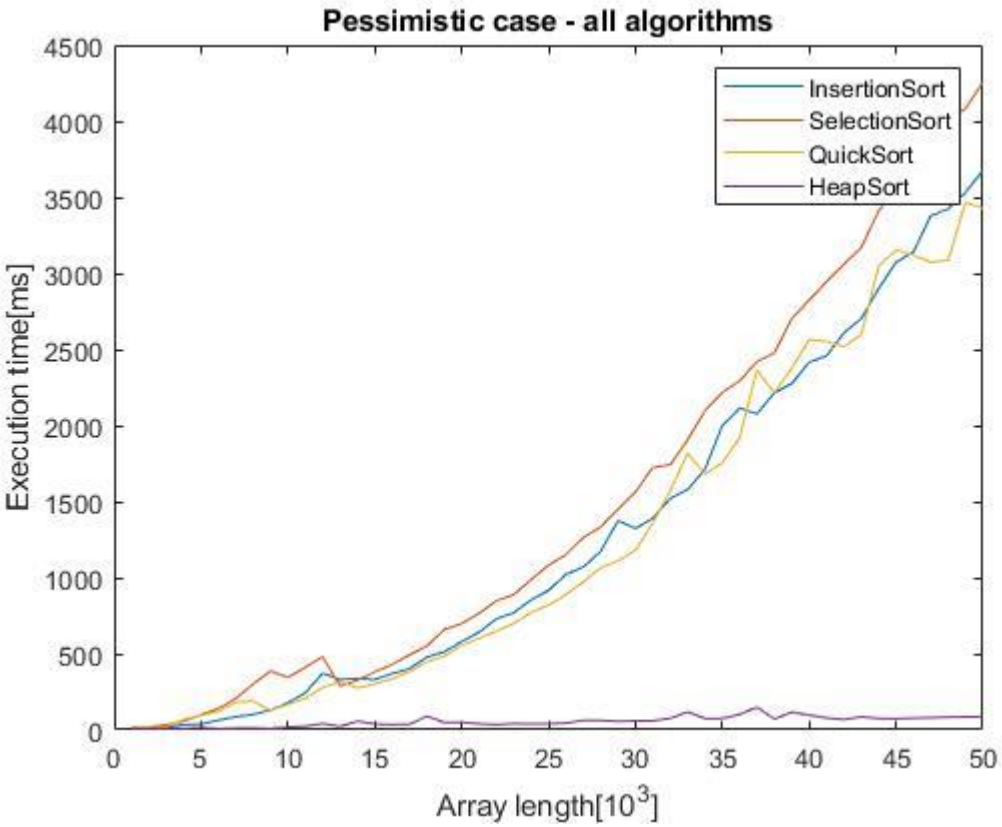
5. Porównanie działania algorytmów

a) Przypadki optymistyczne

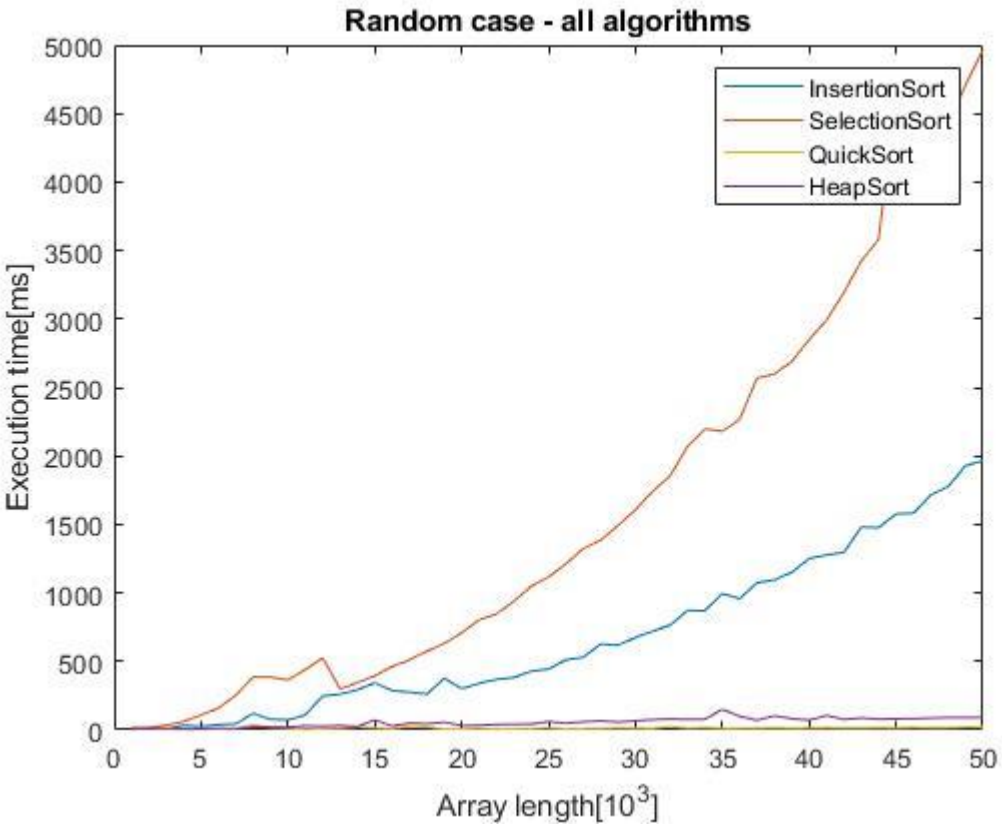
Ponieważ złożoność algorytmu w przypadku optymistycznym dla algorytmu HeapSort oraz InsertionSort wynosi O(n), a dla algorytmu sortowania przez wybór O(n^2), dokonano porównania jedynie algorytmów HeapSort oraz InsertionSort, ze względu na dysproporcje czasu wykonywania.



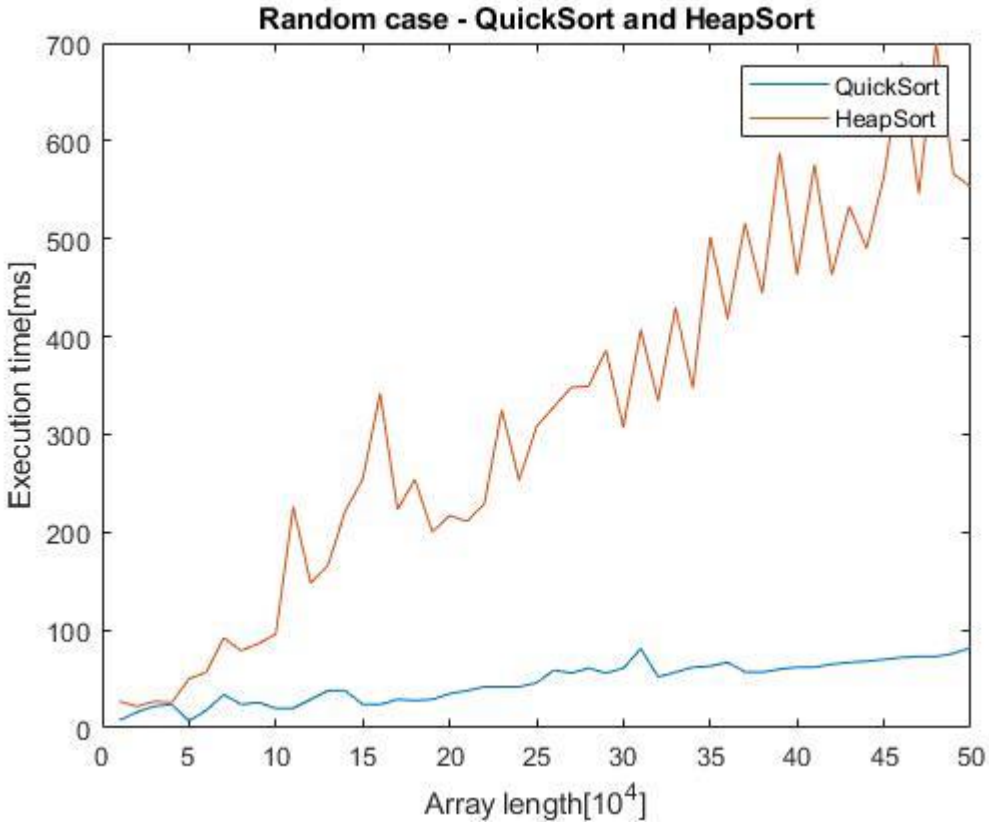
b) Przypadki pesymistyczne



Jak można zauważyć, wszystkie algorytmy z wyjątkiem sortowania przez kopcowanie mają złożoność czasową pesymistyczną rzędu O(n^2) (dla algorytmu sortowania przez wybór jest to każda złożoność). Algorytm HeapSort w przypadku pesymistycznym wykazuje złożoność O(nlogn). Z tego powodu czas wykonywania tego algorytmu w przypadku pesymistycznym jest najkrótszy w porównaniu do pozostałych metod.



W przypadku danych pseudolosowych algorytmy sortowania przez wstawianie oraz przez wybór osiągnęły zauważalnie gorsze rezultaty niż HeapSort oraz QuickSort. Ze względu na długi czas wykonywania pierwszych dwóch rozwiązań, trudno na podstawie wykresu porównać QuickSort oraz HeapSort. W związku z tym dokonano osobnej analizy tych algorytmów z wykorzystaniem tablic o większych rozmiarach.


Random case - QuickSort and HeapSort
QuickSort
HeapSort
Execution time[ms]
Array length[10^4]
0
100
200
300
400
500
600
700
0
5
10
15
20
25
30
35
40
45
50