Porównanie algorytmów sortowania

Pesymistyczne przypadki posiadają algorytmy **InsertionSort** (tablica z odwróconą kolejnością elementów) oraz **QuickSort** (tablica już posortowana).  
W tych przypadkach złożoność obliczeniowa algorytmów ulega pogorszeniu. Pozostałe algorytmy otrzymały do posortowania zbiór danych losowych.

Najlepsze wyniki prezentują (poza **RefAlgorithm**) **HeapSort** oraz **InsertionSort**.

Optymistyczne przypadki posiadają algorytmy **HeapSort** (tablica już posortowana) oraz **InsertionSort** (tablica już posortowana). W tych przypadkach złożoność obliczeniowa algorytmów jest najlepsza. Pozostałe algorytmy otrzymały  
do posortowania zbiór danych losowych.

Najlepsze wyniki prezentują (poza **RefAlgorithm**) **HeapSort** oraz **InsertionSort**.  
Złe wyniki **QuickSort** mogą wskazywać na duże uporządkowanie wygenerowanych losowo danych oraz naiwną implementację, która za każdym razem obiera ten sam element osiowy w trakcie partycjonowania.

Najlepsze wyniki w przypadku losowo wygenerowanych zestawów danych prezentują (poza **RefAlgorithm**) **HeapSort** oraz **InsertionSort**. Złe wyniki **QuickSort** mogą wskazywać na duże uporządkowanie wygenerowanych losowo danych oraz naiwną implementację, która za każdym razem obiera ten sam element osiowy w trakcie partycjonowania.

# Podsumowanie

Wykresy prezentują zdecydowaną dominację algorytmów **HeapSort**  
oraz **InsertionSort**. **HeapSort** jest szybkim algorytmem, którego dodatkowym atutem jest stała złożoność pamięciowa. **InsertionSort** sprawdza się bardzo dobrze  
w przypadku częściowo uporządkowanych już danych. Najgorsze wyniki uzyskał **SelectionSort**, który niezależnie od zestawu danych ma taką samą kwadratową złożoność obliczeniową. Złe wyniki **QuickSort** mogą wskazywać na duże wstępne uporządkowanie wylosowanych danych, co pogarsza czas sortowania w przypadku tego algorytmu.