Paweł Janic 242521

# Sprawozdanie z sortowania Shellsort

1. Opis danych: 5000, 10000, 50000 lub 100000 losowych liczb całkowitych, losowanych z całego przedziału Integera. Wykorzystuję procesor Intel Core i7-7500U 2,7 GHz i 16 GB RAMu.

2. Tabela czasów wykonywania się algorytmów dla tych samych danych losowych:

Shellsort za pomocą insertSorta:

| Algorytm\Ilość danych | 5000 | 10000 | 50000 | 100000 |
|---|---|---|---|---|
| $h_{i+1} = 3h_i + 1$ | 0 | 1 | 4 | 17 |
| $2^k - 1$ | 0 | 1 | 3 | 14 |
| $2^k + 1$ | 1 | 1 | 3 | 16 |
| Fibonacci | 0 | 2 | 5 | 24 |

Shellsort za pomocą insertSorta co h i bubbleSorta co 1:

| Algorytm\Ilość danych | 5000 | 10000 | 50000 | 100000 |
|---|---|---|---|---|
| $h_{i+1} = 3h_i + 1$ | 33 | 115 | 2179 | 9113 |
| $2^k - 1$ | 67 | 358 | 4231 | 21437 |
| $2^k + 1$ | 30 | 133 | 2141 | 8594 |
| Fibonacci | 89 | 281 | 4725 | 18877 |

Shellsort za pomocą bubbleSorta co h i insertSorta co 1:

| Algorytm\Ilość danych | 5000 | 10000 | 50000 | 100000 |
|---|---|---|---|---|
| $h_{i+1} = 3h_i + 1$ | 21 | 81 | 832 | 3804 |
| $2^k - 1$ | 39 | 174 | 1377 | 6299 |
| $2^k + 1$ | 34 | 151 | 1662 | 7801 |
| Fibonacci | 61 | 424 | 3000 | 14136 |

3. Wnioski:

Algorytm insertSort jest dużo bardziej wydajny przez co shellSort tylko za pomocą insertSorta jest zdecydowanie najszybszy. Jest to spowodowane tym, że bubbleSort jest najgorszym ze wszystkich algorytmów sortowania, wykonuje największą ilość porównań. Algorytmy najwolniej działają dla liczb Fibonacciego, ponieważ najwięcej przejść po tablicach i porównań muszą wykonać dla małych liczb odstępów wykorzystywanych między liczbami porównywanymi. Szybkość działania reszty algorytmów zależy od wylosowanych danych, żadna nie dorównuje jednak czystemu insertSortowi w shellSorcie. Czasy działania zależą od procesora i jego obciążenia.