INFO 6205
Program Structures & Algorithms
Fall 2018
Assignment 3

In this assignment, I conducted a benchmark experiment for three different sorting algorithms. *InsertionSort*, *SelectionSort* and *ShellSort*.

1. Conclusion

Some useful abbreviations:

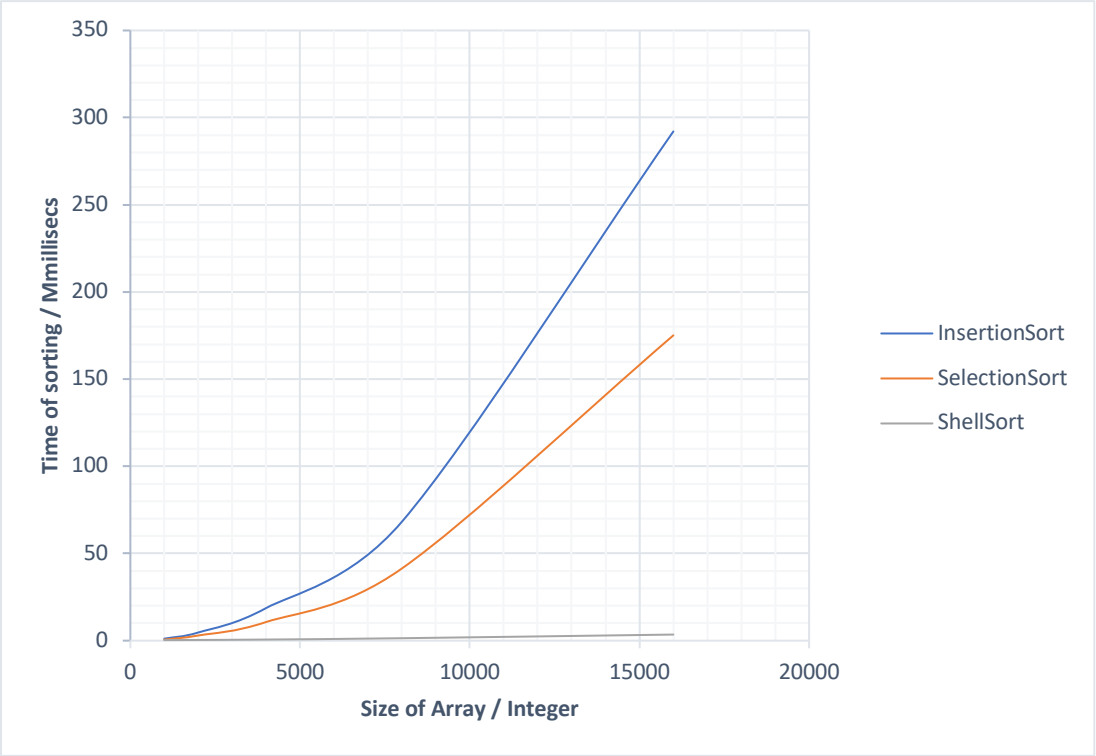
- n – size of the array needs to be sorted.
- m – time of experiment for every algorithm.

After sufficient experiments (with 5 different large enough n and for every n we have m(m>100) times experiments), we get the conclusion for the benchmark of different algorithms.
Performance for different Data set:
Random Order: *ShellSort >>>SelectionSort > InsertionSort ;*
Sorted Order: *InsertionSort / ShellSort >>> SelectionSort;*
Reverse Order: *ShellSort >>> SelectionSort > InsertionSort;*
Partly Sorted: *ShellSort >>> SelectionSort > InsertionSort*;

So we could draw the conclusion that:
For its $O(nlog^2n)$ time complexity, *Shellsort* is best for most random cases, while *InsertionSort* and *SelectionSort* has a $O(n^2)$ time complexity.

2. Graph and explain of various situation for different algorithms
   a) For Random data
      Source: produced by nextInt() method;

	1000	2000	4000	8000	16000
InsertionSort	1.12992338	4.67188572	18.6964583	68.1968617	292.0918659
SelectionSort	0.6036611	2.95166392	10.6871654	41.3826487	175.1245528
ShellSort	0.32518708	0.40627188	0.63440826	1.39468696	3.49580596

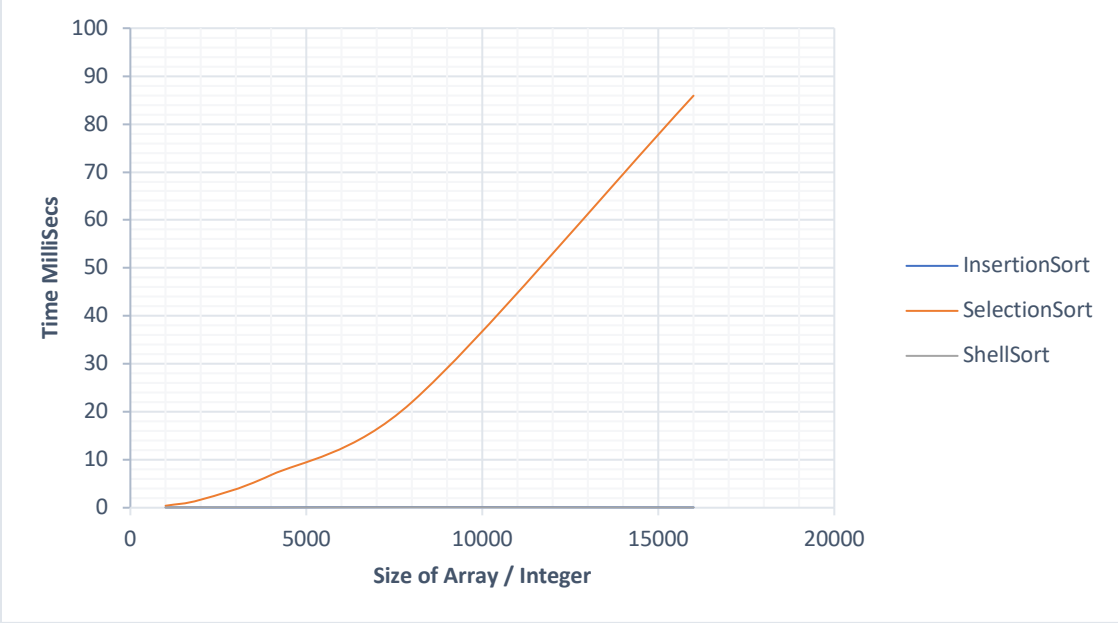


As you can see from the image, that for random order, *ShellSort* is far more faster for its $O(nlog^2n)$ complexity.

b) For Sorted Data

Source: Produced with a for loop;

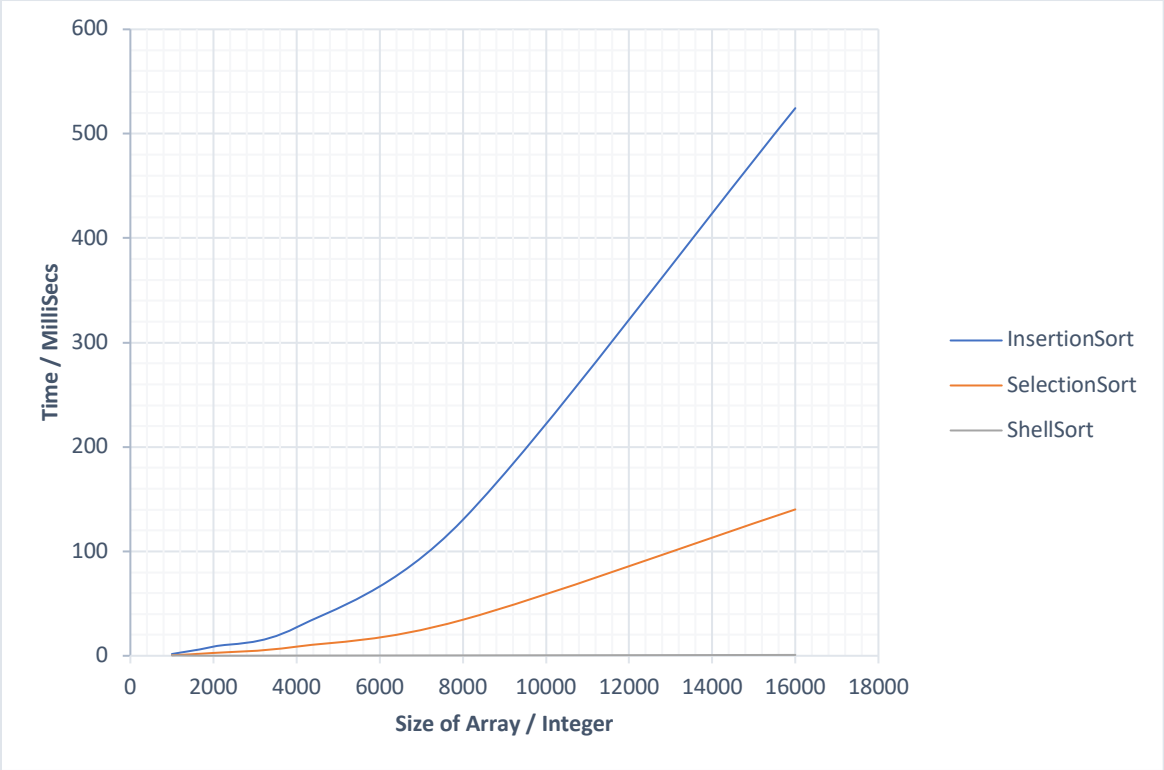
	1000	2000	4000	8000	16000
InsertionSort	0.0678279	0.061348	0.0625198	0.1158173	0.0662771
SelectionSort	0.4074584	1.6798575	6.8274746	22.0263852	85.9312073
ShellSort	0.037217	0.0838823	0.0508151	0.0939515	0.0538052



As you can see, for sorted data, *InsertionSort/ShellSort* shows almost constant time for sorting a sorted array. However, the time of *SelectionSort* increase more than quadraticly.

c) For Reverse Ordered Data
Source: Produced with a for loop;

	1000	2000	4000	8000	16000
InsertionSort	1.6894648	8.7929925	27.3101327	130.217923	524.5186773
SelectionSort	0.5505521	2.7275985	8.7617838	34.5135393	140.1878233
ShellSort	0.40227476	0.08288832	0.18800922	0.37741542	0.82977806

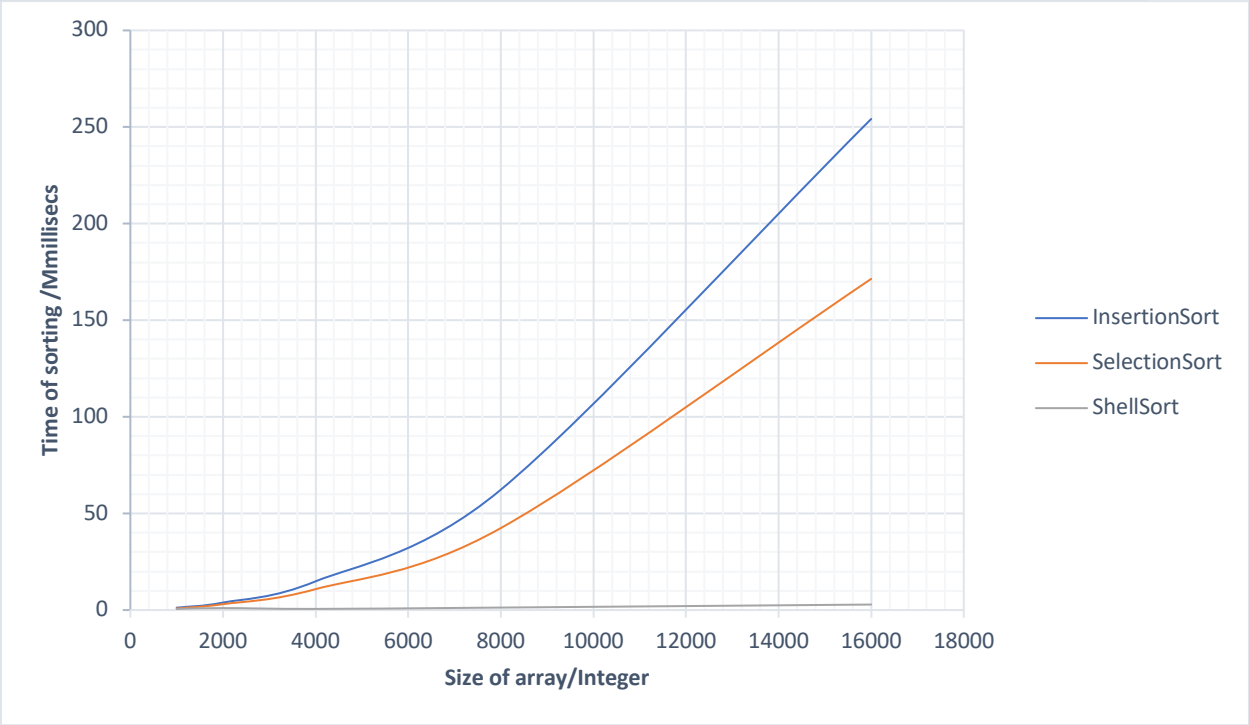


As you can see, for reversed ordered data, *InsertionSort* is slowest, and *ShellSort* is fastest.

d) For Partly Ordered Data

Source: 1/3 is random, 1/3 is sorted, 1/3 is reversed;

	1000	2000	4000	8000	16000
InsertionSort	1.2156589	3.8631012	14.861532	62.2823114	254.1612396
SelectionSort	0.8227874	3.0475271	10.8237339	42.4124837	171.3978983
ShellSort	0.5871897	1.0134824	0.6013666	1.2861508	2.8530249



As you can see, the *SelectionSort*/InsertionSortis bad for Partly Ordered Data, and *ShellSort* is the best.