CO322: Data Structures and Algorithms Sorting Algorithms
Lab 01 Sorting Algorithms

E/19/129
K.H. Gunawardana

The submission contains 2 Java implementations,

- Sorting.java – implementations of sorting algorithms and how they sort in each case.
- SortingMeasurements.java – implementation for different input sizes in 3 cases, measure the performance of each algorithm with time.

By using the Sorting_Algo_Measurements.java we can get the outputs as follows:

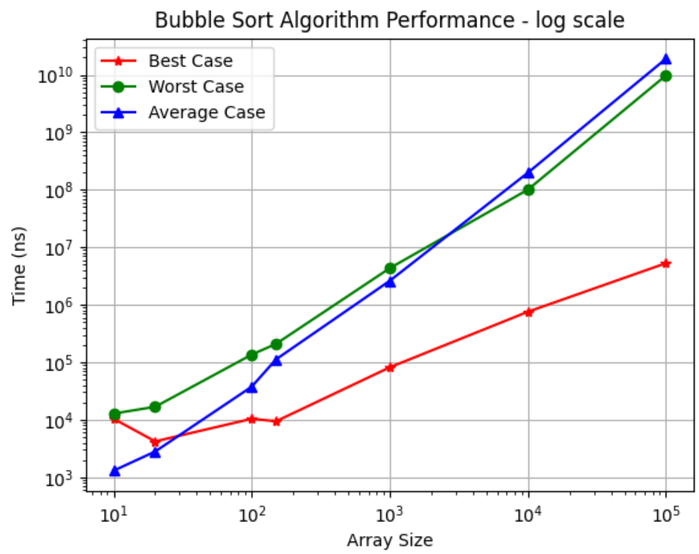
```
Best-Case Data
-----------------------------------------------------------------------
Array Size   | Bubble Sort       | Selection Sort    | Insertion Sort    |
-----------------------------------------------------------------------
10           |          10400 ns |          24100 ns |          19100 ns |
20           |           4200 ns |          21100 ns |           3700 ns |
100          |          10400 ns |         303400 ns |          10600 ns |
150          |           9400 ns |         581500 ns |          15900 ns |
1000         |          81700 ns |        9396100 ns |          95900 ns |
10000        |         756600 ns |       55137100 ns |         880600 ns |
100000       |        5292100 ns |     1534489800 ns |        3406600 ns |
-----------------------------------------------------------------------
```

```
Worst-Case Data
-----------------------------------------------------------------------
Array Size   | Bubble Sort       | Selection Sort    | Insertion Sort    |
-----------------------------------------------------------------------
10           |          12800 ns |         100000 ns |          14500 ns |
20           |          16800 ns |          14700 ns |          11000 ns |
100          |         135900 ns |         255100 ns |          86400 ns |
150          |         208800 ns |         109800 ns |         145500 ns |
1000         |        4338300 ns |        2797200 ns |        1338600 ns |
10000        |      100810000 ns |       38245400 ns |       64721300 ns |
100000       |     9794596700 ns |     4266750900 ns |     6504190400 ns |
-----------------------------------------------------------------------
```

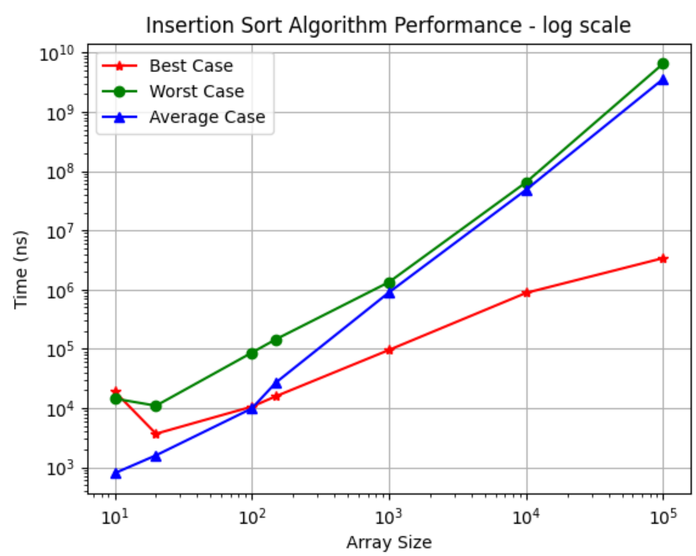
```
Average-Case Data
-----------------------------------------------------------------------
Array Size   | Bubble Sort       | Selection Sort    | Insertion Sort    |
-----------------------------------------------------------------------
10           |           1300 ns |           2000 ns |            800 ns |
20           |           2800 ns |           2400 ns |           1600 ns |
100          |          37600 ns |          14100 ns |           9900 ns |
150          |         112300 ns |          27000 ns |          27100 ns |
1000         |        2615300 ns |         427700 ns |         897600 ns |
10000        |      198475800 ns |       48429500 ns |       48365000 ns |
100000       |    19118995400 ns |     1614373600 ns |     3588064000 ns |
-----------------------------------------------------------------------
```

Q1. How does the performance vary with the input size?

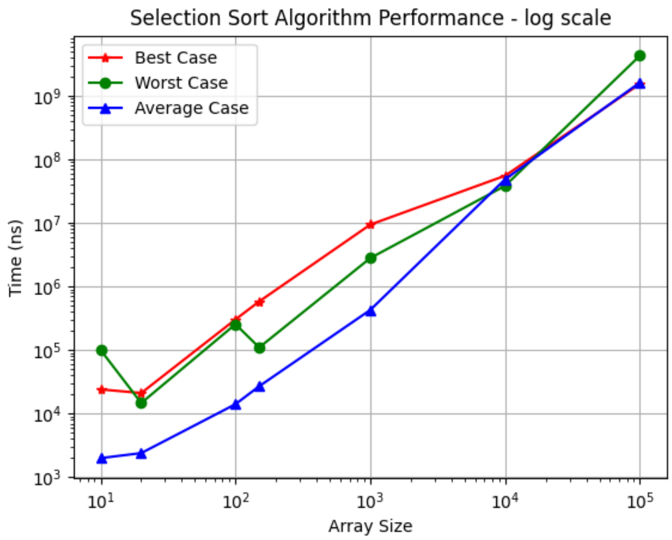
By observation of all the 3 cases(best/worst/average) in the algorithms, overall, when the input size increases the time that it takes to sort the input array also increases.



The above figures show how the time varies in bubble sort algorithms when increasing the input size of each case.



The above figures show how the time varies in insertion sort algorithms when increasing the input size of each case.



The above figures show how the time varies in selection sort algorithms when increasing the input size of each case.

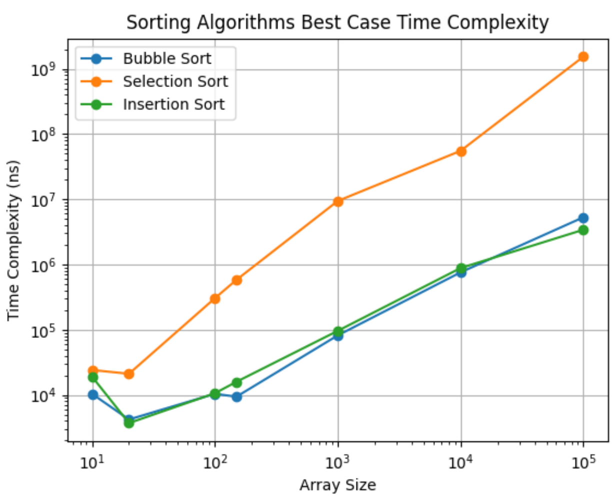
Thus, we can observe that the in-bubble sort and insertion sort, the best case has a lower increasing rate than the other cases when the size of the array increases. In those sorting algorithms, the worst-case and average-case kinds behave similarly for larger array sizes. In the selection sort, all three cases act as same when increasing the array size.

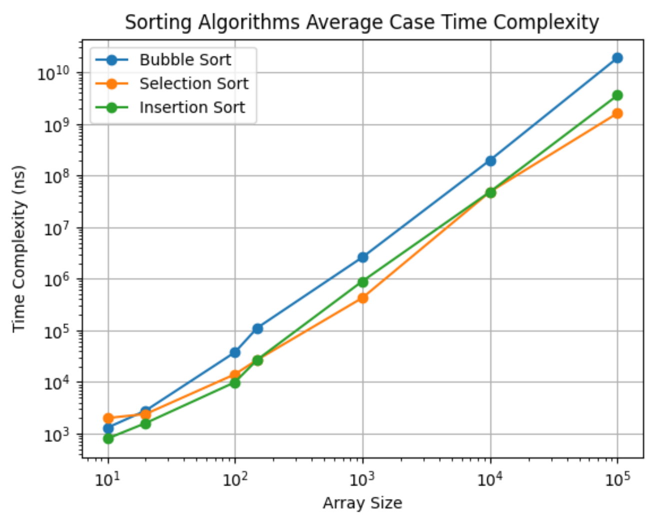
Q2. Does the empirical results you get agree with the theoretical analysis?

- The theoretical analysis of each algorithm is shown below:

Algorithm	**Best Case**	**Average Case**	**Worst Case**
Bubble Sort	O(N)	O(N^2)	O(N^2)
Selection Sort	O(N^2)	O(N^2)	O(N^2)
Insertion Sort	O(N)	O(N^2)	O(N)

- The empirical results of each algorithm are shown below:





The practical test results show that when we have the best possible starting order of data, Bubble Sort gets slower as the data gets bigger, Selection Sort is consistently slow, and Insertion Sort stays relatively steady. In the worst-case situation, both Bubble Sort and Selection Sort are consistently slow for different data sizes, but Insertion Sort stays steadier. In an average situation, Bubble Sort is consistently slow, Selection Sort gets a bit better but is still not as good as Insertion Sort, which stays efficient for different data sizes.

This matches what we expected from the theory, where Bubble Sort and Selection Sort, with their $O(N^2)$ complexities, struggle as the data gets larger. Insertion Sort, also $O(N^2)$, performs better, especially when the data is partially sorted.

Q3. How did/should you test your code? Explain the test cases you used and the rationale foruse them.

In Sorting.java, the testing done for each algorithm is implemented in the best case, average case and worst case. In each case, an array of size 10 is used and measures the time that takes to sort it. In the best case, a sorted array was used and in the worst case, a reversed sorted array was used. In the case of average, a randomly generated array was used. The output is shown below.

```
------ Bubble Sort ------
Best Case:
=======
0  1  2  3  4  5  6  7  8  9
=======
=======
0  1  2  3  4  5  6  7  8  9
=======
Time: 14900ns

Average Case:
=======
61  50  75  51  50  58  39  13  57  38
=======
=======
13  38  39  50  50  51  57  58  61  75
=======
Time: 45500ns

Worst Case:
=======
10  9  8  7  6  5  4  3  2  1
=======
=======
1  2  3  4  5  6  7  8  9  10
=======
Time: 29200ns

----- Selection Sort -----
Best Case:
=======
0  1  2  3  4  5  6  7  8  9
=======
=======
0  1  2  3  4  5  6  7  8  9
=======
Time: 36900ns

Average Case:
=======
16  61  9  81  4  65  79  28  56  12
=======
=======
4  9  12  16  28  56  61  65  79  81
=======
Time: 18200ns

Worst Case:
=======
10  9  8  7  6  5  4  3  2  1
=======
=======
1  2  3  4  5  6  7  8  9  10
=======
Time: 39300ns
```

```
----- Insertion Sort -----
Best Case:
=======
0  1  2  3  4  5  6  7  8  9
=======
=======
0  1  2  3  4  5  6  7  8  9
=======
Time: 25500ns

Average Case:
=======
56  9  14  55  53  85  13  91  20  68
=======
=======
9  13  14  20  53  55  56  68  85  91
=======
Time: 12000ns

Worst Case:
=======
10  9  8  7  6  5  4  3  2  1
=======
=======
1  2  3  4  5  6  7  8  9  10
=======
Time: 31200ns
```

In the best-case scenario, the array is sorted in ascending order already, presenting the ideal conditions for a sorting algorithm. This signifies that Bubble Sort requires no swaps, while for Selection Sort, it represents a scenario with a minimal number of required swaps. In the case of Insertion Sort, it is the situation where minimal shifts are needed.

In the average scenario, we have a bunch of random numbers in an array. This is like real-life situations where things aren't in a special order. We're checking how good the sorting method is when the data isn't already sorted. This matters more for Bubble Sort and Selection Sort because, in real life, our data is usually not sorted to begin with. For Insertion Sort, we're looking at how well it can put things in order when the array is only partly sorted.

In the worst-case scenario, we have an array sorted in reverse order. This means the array is the opposite of what we want. For Bubble Sort and Selection Sort, it's a situation where we need the maximum number of swaps. In Insertion Sort, it's when we must make the maximum number of shifts to get things in the right order.

The SortingMeasurements.java uses input array sizes as 10, 20, 100, 150, 1000, 10000 and 100000 for each best, worst and average case. In each size, the same array is sorted using the 3 algorithms and calculates the time that it takes to sort. Q1 and Q2 explain this scenario.