Practical 4: Elementary Sorting

Name: Rajit Banerjee
Student Number: 18202817

What am I doing today?
Today's practical focuses on 3 things:

1. Writing several elementary sorting algorithms
2. Developing a testing framework to assess the performance of your algorithms
3. Summarizing the results

Instructions
Try all the questions. Ask for help from the demonstrators if you get stuck.

*****Grading: Remember** if you complete the practical, add the code to your GitHub repo which needs to be submitted at the end of the course **for an extra 5%**

Quick Questions

1. **How many compares does insertion sort make on an input array that is *already sorted*?**

Constant	
Logarithmic	
Linear	**X**
Quadratic	

2. **What is a stable sorting algorithm?**

   It is an algorithm that doesn't change the existing relative order of equal-key elements.

3. **What is an external sorting algorithm?**

A. Algorithm that uses tape or disk during the sort

B. Algorithm that uses main memory during the sort

C. Algorithm that involves swapping

D. Algorithm that are considered 'in place'

4. **Identify 6 ways of classifying sorting algorithms?**

1.	Comparison vs Non-comparison
2.	Time Complexity
3.	Space Complexity
4.	Stability
5.	Internal vs External
6.	Recursive vs Non-recursive

Algorithmic Development

Today your mission is to develop a Java class that implements several elementary (and silly) sorting algorithms. The problem we want our algorithms to solve is sort an input array of integers into ascending order and output the resulting array.

Possible steps to follow

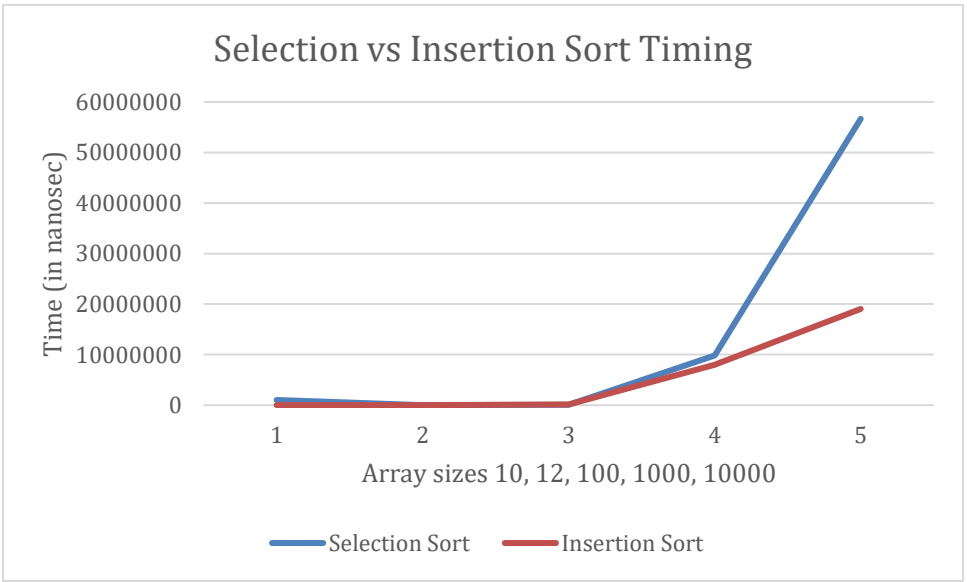
1. Create a new java class
2. Implement the following sorting algorithms as public static functions within your class that take an array of integers and sorts the array, outputting a sorted array of integers:
    a. Selection sort
    b. Insertion Sort
    c. A silly sort (either from the list below or of your own making)
3. Create a simple framework for generating input arrays of various sizes (e.g., 10, 1000, 100,000) and then testing the performance over several runs
4. Print the resulting sorted array: Implement a function to print out all elements in the array
5. Time the performance of the previous step on your 3 algorithms and output the execution times for various input sizes (e.g. 10,100,1000) on a graph
6. Justify the results of your experiments for the algorithms by proposing the algorithm complexity in big-O notation
7. BONUS: adjust your insertion sort algorithm to be an unstable sort

Sorting: Timing Analysis

1. Selection Sort – O(n^2)
2. Insertion Sort – O(n^2)
3. Bogo Sort – O(infinity) in the worst case

# of elements	Time (in nanosec)		
Array Size	Selection Sort	Insertion Sort	Bogo Sort
10	1017400	8100	517700200
12	3700	4100	2013038500
100	89500	114200	
1000	9822600	7987900	
10000	56658200	19044800	

Bogo sort takes too long (in nanoseconds) to be plotted on the same graph as Selection and Insertion Sorts.



Timing analysis: Selection, Insertion and Bogo Sorts

For anything above array size 12, Bogo Sort takes too long to finish.

```
Welcome to Practical 4!
1. Run timing analysis.
2. See sorted arrays (only small sizes).
Choose 1 or 2: 1

--------------------------------------------------------------------------------

-Selection Sort-
Time taken for array of size 10 =       1017400 nanosec
Time taken for array of size 12 =       3700 nanosec
Time taken for array of size 100 =      89500 nanosec
Time taken for array of size 1000 =     9822600 nanosec
Time taken for array of size 10000 =    56658200 nanosec

--------------------------------------------------------------------------------

-Insertion Sort-
Time taken for array of size 10 =       8100 nanosec
Time taken for array of size 12 =       4100 nanosec
Time taken for array of size 100 =      114200 nanosec
Time taken for array of size 1000 =     7987900 nanosec
Time taken for array of size 10000 =    19044800 nanosec

--------------------------------------------------------------------------------

-Bogo Sort-
Time taken for array of size 10 =       517700200 nanosec
Time taken for array of size 12 =       2013038500 nanosec
```

Visualise the 3 sorts: Selection, Insertion, Bogo

Larger array sizes take too much space to print in the command line.

```
Welcome to Practical 4!
1. Run timing analysis.
2. See sorted arrays (only small sizes).
Choose 1 or 2: 2

Orignal array (size 10):        { 7, 4, 3, 5, 2, 5, 4, 6, 3, 8 }
After selection sort:           { 2, 3, 3, 4, 4, 5, 5, 6, 7, 8 }

Orignal array (size 12):        { 4, 4, 4, 4, 8, 3, 3, 10, 11, 0, 1, 6 }
After selection sort:           { 0, 1, 3, 3, 4, 4, 4, 4, 6, 8, 10, 11 }

Orignal array (size 10):        { 7, 4, 3, 5, 2, 5, 4, 6, 3, 8 }
After insertion sort:           { 2, 3, 3, 4, 4, 5, 5, 6, 7, 8 }

Orignal array (size 12):        { 4, 4, 4, 4, 8, 3, 3, 10, 11, 0, 1, 6 }
After insertion sort:           { 0, 1, 3, 3, 4, 4, 4, 4, 6, 8, 10, 11 }

Orignal array (size 10):        { 7, 4, 3, 5, 2, 5, 4, 6, 3, 8 }
After bogo sort:                { 2, 3, 3, 4, 4, 5, 5, 6, 7, 8 }

Orignal array (size 12):        { 4, 4, 4, 4, 8, 3, 3, 10, 11, 0, 1, 6 }
After bogo sort:                { 0, 1, 3, 3, 4, 4, 4, 4, 6, 8, 10, 11 }
```