Practical 4: Elementary Sorting Algorithms

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

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

Timing analysis: Selection, Insertion and Bogo Sorts

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

```
-Sorting Analysis-
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 = 85400 nanoseconds
Time taken for array of size 12 = 16800 nanoseconds
Time taken for array of size 100 = 651400 nanoseconds
Time taken for array of size 1000 = 17440000 nanoseconds
Time taken for array of size 10000 = 185364400 nanoseconds
-----------------------------------------------------------
-INSERTION_SORT-
Time taken for array of size 10 = 129800 nanoseconds
Time taken for array of size 12 = 17900 nanoseconds
Time taken for array of size 100 = 430300 nanoseconds
Time taken for array of size 1000 = 17495200 nanoseconds
Time taken for array of size 10000 = 249887400 nanoseconds
-----------------------------------------------------------
-BOGO_SORT-
Time taken for array of size 10 = 34550400 nanoseconds
Time taken for array of size 12 = 1442261400 nanoseconds
```

Visualise the 3 sorts: Selection, Insertion, Bogo

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

```
-Sorting Analysis-
1. Run timing analysis.
2. See sorted arrays (only small sizes).
Choose 1 or 2: 2

Array size: 10
BEFORE SELECTION_SORT:  [7, 3, 7, 9, 4, 4, 1, 1, 6, 9]
AFTER SELECTION_SORT:   [1, 1, 3, 4, 4, 6, 7, 7, 9, 9]

Array size: 12
BEFORE SELECTION_SORT:  [5, 2, 9, 8, 1, 6, 2, 9, 6, 4, 3, 0]
AFTER SELECTION_SORT:   [0, 1, 2, 2, 3, 4, 5, 6, 6, 8, 9, 9]

Array size: 10
BEFORE INSERTION_SORT:  [7, 3, 7, 9, 4, 4, 1, 1, 6, 9]
AFTER INSERTION_SORT:   [1, 1, 3, 4, 4, 6, 7, 7, 9, 9]

Array size: 12
BEFORE INSERTION_SORT:  [5, 2, 9, 8, 1, 6, 2, 9, 6, 4, 3, 0]
AFTER INSERTION_SORT:   [0, 1, 2, 2, 3, 4, 5, 6, 6, 8, 9, 9]

Array size: 10
BEFORE BOGO_SORT:   [7, 3, 7, 9, 4, 4, 1, 1, 6, 9]
AFTER BOGO_SORT:    [1, 1, 3, 4, 4, 6, 7, 7, 9, 9]

Array size: 12
BEFORE BOGO_SORT:   [5, 2, 9, 8, 1, 6, 2, 9, 6, 4, 3, 0]
AFTER BOGO_SORT:    [0, 1, 2, 2, 3, 4, 5, 6, 6, 8, 9, 9]
```

Sorting: Timing Analysis

1. Selection Sort – O(n^2)
2. Insertion Sort – O(n^2)
3. Bogo Sort – O(infinity)

Array Size	Time (in nanoseconds)		
	Selection Sort	Insertion Sort	Bogo Sort
10	85400	129800	34550400
12	16800	17900	1442261400
100	651400	430300	
1000	17440000	17495200	
10000	185364400	249887400	

Bogo sort takes too long to finish for array size > 12, and cannot be realistically plotted on the same graph as Selection and Insertion sorts, in order for us to be able to compare the latter two.