CS 245 - Practice Assignment 3 Spring, 2020

Practice Assignment 3 - Searching & Sorting

The goal of this assignment is to practice the implementation of the searching & sorting algorithms we discussed and to determine the efficiency as measured by (unreliable) wall clock time.

Background

We discussed two algorithms for searching: linear search and binary search. Binary search has two variants: a recursive version and an iterative version. We determined that linear search has a running time of O(n) and that binary search has a running time of $O(\log_2 n)$, with the iterative version being more efficient than the recursive version. This assignment will have you implement these algorithms as functions and test the timing of the implementations.

In addition, we discussed three algorithms for sorting: selection sort, bubble sort and insertion sort. These algorithms have the same running time — $O(n^2)$. This assignment will have you implement these algorithms as functions and test the timing of the implementations.

Requirements (Process)

Requirement 1: Get the files you need. You will copy them to your own GitHub repository using the following procedure:

1. Log into GitHub. If you do not have a GitHub account, create one via github.com > Sign Up.
2. Point your browser to the URL https://classroom.github.com/a/6JxJwiHb.
3. If necessary, authorize GitHub Classroom by selecting the button "Authorize github."
4. If available, select your name from the list of names available. This will link your GitHub ID.
5. Accept the assignment by selecting the appropriate button.

If successful, your repository should contain four (4) Java files:

- Practice03Factory.java — the factory for the class Practice03Search
- Practice03Search.java — the interface for the search classes
- Practice03Sort.java — the interface for the sorting classes
- Practice03Test.java — the main file

Requirement 2: Add five (5) classes to the code in order to make the implementation run. Specifically, you must create the classes listed under "Class Required" using the interface in Table 1. You are not allowed to make changes to any of the files you downloaded in Requirement 1 above.

Each class has one required function specified by its interface. You may add any additional functions to get it running or successfully debugged. You may [read: should] use any Integrated Development Environment (IDE) while implementing. IDEs include Eclipse, IntelliJ, Sublime, etc. You may also choose not to use an IDE, though this is not recommended.

Class Required	**Interface**
`BinaryIterativeSearch.java`	`Practice03Search.java`
`BinaryRecursiveSearch.java`	`Practice03Search.java`
`LinearSearch.java`	`Practice03Search.java`
`SelectionSort.java`	`Practice03Sort.java`
`BubbleSort.java`	`Practice03Sort.java`
`InsertionSort.java`	`Practice03Sort.java`

Table 1: Required classes, interfaces and functions

Requirement 3: Test your implementation by compiling and running the implementation. When you do this successfully, you should see output such as the following (although your values may be different):

```
linear search (on 50000-item array): total search time = 861ms. Average search time =
0.01722ms.
linear search (on 100000-item array): total search time = 1500ms. Average search time =
0.015ms.
linear search (on 150000-item array): total search time = 2317ms. Average search time =
0.015446667ms.
```

The sorting algorithms may take quite a long time to run. You should do so only when you have the time to collect the time to leave it running uninterrupted.

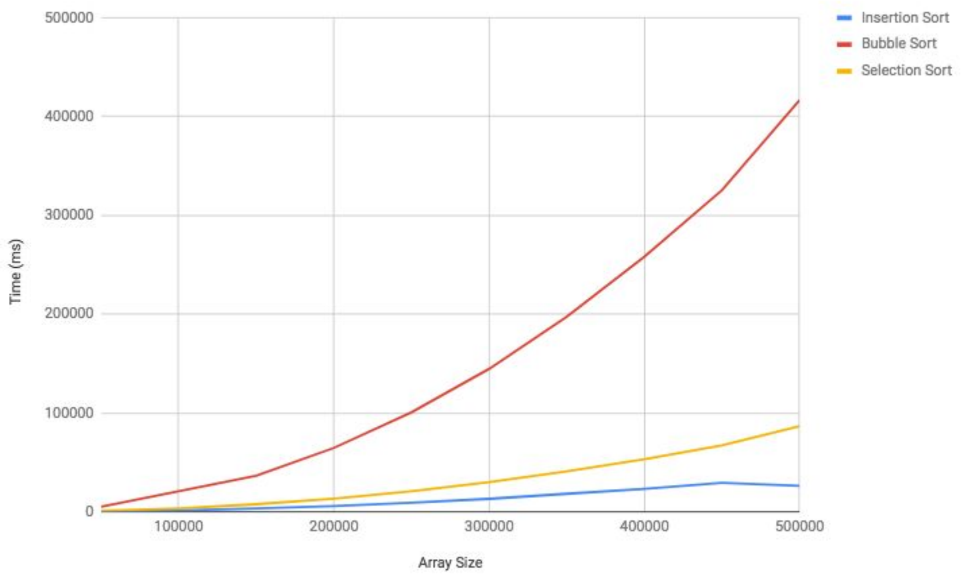

Figure 1: Chart of run timing

Requirement 4: Produce a chart of running times for the sorting algorithms. In order to do this, you will need to execute the main file, which will run your implementations against the values in the Practice03Test class. You may use any software to produce this chart — including Google Sheets, Numbers or Excel — by copying the output of the sort times to the software in question and inserting a chart of the numbers in the output. An example chart is shown in Figure 1.

Grading

Grades will be determined as follows:

- 30% = Correct implementation of each search algorithm (10% each)
- 48% = Correct implementation of each sorting algorithm (16% each)
- 15% = Graph with all (30) data points (0.5% each data point)
- 7% = Style, including consistent indentation, variable naming, etc.

This assignment is formally graded only on correctness and style. Each of the three required classes is worth 25% toward your grade, and your style (variable and class names, etc.) constitute the remaining 25% of the grade for this assignment.

Submission

Use GitHub to check in the code to complete the assignment. Once your implementation is on GitHub, submit the URL for your GitHub repository on Canvas.