Assignment 1 — Algorithm Analysis

Analysis of algorithms concerns two broad concepts: *correctness* and *complexity*. Given that an algorithm produces the correct output for any input, we are interested in the resources it requires in different situations. In this assignment, you'll measure and predict the running times of three algorithms that solve the same problem in different ways.

1 Requirements

For this assignment (and throughout the class), you may implement your solution in either Python 3 or Java 18. To make automated grading possible, you will also need to submit two short shell scripts: one to compile your submission (`compile.sh`) and one to run it (`run.sh`). (Note that `compile.sh` will be a no-op for Python but still must be turned in.)

All source code should be in the root directory of the fit repository, not in directories or compressed files.

You *may not* use any third party libraries, even if they are installed on Cal Poly's Unix servers.

2 GitHub Classroom

For this assignment (and again throughout the class), you'll submit your programs through GitHub Classroom. You're hopefully already familiar with `git` from previous classes but if you are not, the following may be helpful:

- `https://docs.github.com/en/articles/set-up-git`
- `https://docs.github.com/en/articles/cloning-a-repository`
- `https://docs.github.com/en/articles/pushing-commits-to-a-remote-repository`

To facilitate the use of GitHub Classroom, I'll need to be able to match your GitHub account to your Cal Poly id. If you have not already, please fill out the form I sent out to collect GitHub usernames.

To begin this assignment, you'll need to accept the GitHub Classroom assignment (link on Canvas). This will create a new repository for you on GitHub which you will use to submit this assignment as well as provide some sample inputs and their expected outputs.

3 Sorting Algorithms

Recall that there are several common algorithms for sorting, including:

- SelectionSort repeatedly *selects* the smallest element from the unsorted elements.
- InsertionSort repeatedly *inserts* the next unsorted element into a sorted section.
- MergeSort recursively sorts the two halves of a sequence, then *merges* the sorted halves back together.

In your progamming language of choice, implement these three algorithms for sequences of integers. Your program must accept as a command line argument the name of a file containing such a sequence and will print to `stdout` (i.e. print out normally) the running times and results of applying the three algorithms. For example:

```
$ ./compile.sh
$ ./run.sh test_files/in1.txt
Selection Sort (0.02 ms): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
Insertion Sort (0.02 ms): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
Merge Sort     (0.04 ms): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
```

Your running times will likely vary from mine and that's expected. The exact number of times doesn't matter so long as you're consistent in how you do your timing. Excluding the run times, your program will be tested using `diff`, so make sure your output matches mine *exactly*.

4 Analysis

Record the running times of each algorithm for sequences of randomly generated unsigned 16 bit integers (i.e. from 0 to 65,535 inclusive) between 5,000 and 10,000 elements in length, in increments of 10. That is to say:

1. Randomly generate a sequence of 5,000 integers. Run the three sorts and record their running times.
2. Randomly generate a sequence of 5,010 integers. Run the three sorts and record their running times.
3. Randomly generate a sequence of 5,020 integers. Run the three sorts and record their running times.
4. ...

You may (and probably should) write a program to automate this process, however, not that this is separate from and should not be a part of the program specified in the previous part.

Plot these timing results, with the lengths of the sequences on the horizontal axis and the running times of the vertical axis. You should have a scatter plot with 500 points for each algorithm.

Finally, using these empirical running times, extrapolate to answer the following questions:

1. How long should each algorithm take to sort 25,000 random integers?
2. How many elements should each be able to sort in 10 minutes?

Consider fitting a curve to your plots in order to answer these questions.

5 Submission

The following items must be demonstrated/presented to me in lab on the day of the deadline.

- Plots of the three sorting algorithms' running times for varying sequence lengths.
- Extrapolations of their predicted performance.

The following files are required and must be pushed to your GitHub Classroom repository by the deadline:

1. `compile.sh` — A bash script to compile your submission (even if it does nothing).
2. `run.sh` — A bash script to run your submission.
3. `*.py` or `*.java` — Your source code in Python or Java of a working program to compare the three sorting algorithms.