Assignment 1 — Algorithm Analysis

Due: Friday, September 27th

Credit: This is a lab by Christopher Siu.

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.

Deliverables:

GitHub Classroom:	`https://classroom.github.com/a/Bc17ge2K`
Required Files:	`compile.sh`, `run.sh`
Optional Files:	`*.py`, `*.java`, `*.clj`, `*.kt`, `*.js`, `*.sh`

Part 1: Ground Rules

Throughout this class, you may implement your assignments in one of the following languages:

- Python 3.7.4
- Clojure 1.10.1[1]
- Node.js 12.10.0
- Java 12.0.2
- Kotlin 1.3.50[2]
- GNU Bash 5.0.2[3]

However, Python is recommended as it is the only language I can help you with.

To make automated grading possible, you will also need to submit two short shell scripts: one to compile your submission, one to run it. Further details, along with sample submissions in all supported languages, can be found here: `https://github.com/csc349fall19/asgn-samples`

Furthermore:

- Submit only source code files as they are. Any directories or compressed files will be ignored.
- For record-keeping purposes, you must hand in original source files, not pre-compiled files.
- You may not use *any* third party libraries[4], even if they are installed on Cal Poly's Unix servers.

Part 2: GitHub Classroom

In this class, you'll submit your programs through GitHub, an online host for tool known simply as "git". Git tracks changes you make to files so that you can easily undo or combine changes while working on a project.

You should already be familiar with the basic features of git from previous classes, such as CSC 202 and CSC 203. If not, the following resources might be helpful:

- `https://help.github.com/en/articles/set-up-git`
- `https://help.github.com/en/articles/cloning-a-repository`
- `https://help.github.com/en/articles/managing-files-using-the-command-line`
- `https://help.github.com/en/articles/pushing-to-a-remote`

GitHub Classroom allows your instructors to accept electronic submissions through GitHub.

1. Enter your Cal Poly and GitHub usernames in this form: `https://forms.gle/fkYfJRpyLNCtRWAs9`. This allows us to determine who should get credit for your electronically submitted assignments.
2. Accept this GitHub Classroom assignment: `https://classroom.github.com/a/Bc17ge2K`. 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.

[1]You may assume that the Clojure JARs are in the classpath and that the `clj` and `clojure` commands are available.
[2]You may assume that the Kotlin `lib` is in the classpath and that the Kotlin `bin` is in the path, for JVM-backed Kotlin.
[3]Not that I'd recommend it — this is very much *not* what Bash was designed for.
[4]And your submission won't have Internet access when it's run, so you cannot use `apt`/`pip`/`lein`/`npm` to install them.

Part 3: Sorting Algorithms

Recall that there exist several algorithms commonly used to solve the sorting problem, including:

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

In your programming language of choice per Part 1, 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, printing to `stdout` the results of applying the three algorithms. For example:

```
>$ ./compile.sh
>$ ./run.sh in1.txt
Selection Sort: -16, -15, -14, -7, -6, -5, -4, 0, 1, 2, 5, 7, 8, 13, 14, 15
Insertion Sort: -16, -15, -14, -7, -6, -5, -4, 0, 1, 2, 5, 7, 8, 13, 14, 15
Merge Sort    : -16, -15, -14, -7, -6, -5, -4, 0, 1, 2, 5, 7, 8, 13, 14, 15
```

Your program will be tested using `diff`, so its printed output must match *exactly.*

Part 4: Analysis

Record the running times of each algorithm for sequences of randomly generated integers 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 certainly write a program to automate this process, however, note that this is separate from and should not be a part of the program specified in Part 3.

Plot these timing results, with the length of the sequence on the horizontal axis and the running time on the vertical axis — you should have a scatter plot with 500 points for each algorithm.

Finally, using these empirical running times, extrapolate: How long should each algorithm take to sort $25,000$ elements? 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.

Part 5: Submission

The following items must be demonstrated in lab on the day of the deadline:

- Plots of the three sorting algorithms' running times for varying sequence lengths, together with extrapolations of their predicted performance.

The following files are required and must be pushed to your GitHub Classroom repository by 8pm of the due date:

- `compile.sh` — A Bash script to compile your submission (even if it does nothing), as specified.
- `run.sh` — A Bash script to run your submission, as specified.

The following files are optional:

- `*.py`, `*.java`, `*.clj`, `*.kt`, `*.js`, `*.sh` — The Python, Java, Clojure, Kotlin, Node.js, or Bash source code files of a working program to compare three sorting algorithms, as specified.

Any files other than these will be ignored.

Additionally:

- On the date before the due date, the grader will be run at 8pm and provides feedback containing only what your program scores. Only submissions made before 8pm are guaranteed to receive feedback. What your program scores on the official run (the due date) is the final score.
- Late submissions made within 24 hours of the deadline receive a .7 multiplier. For example, if your late submission scores 80%, the final score will be 80% * .7 = 56%.
- If you decide to make a late submission, please notify me by sending an email to vnguy143@calpoly.eduwith both the subject and the body in the format: "CSC349,late,asgn<i>", with i being the assignment number. For example, if you opt to submit this assignment late, both the subject and the body of the email should be "CSC349,late,asgn1" (without the quotation marks, of course).