Assignment 1 — Divide and Conquer

Due: Monday, September 28th

Credit: This is a lab by Christopher Siu

A "divide and conquer" algorithm is one that divides a problem into smaller subproblems, solves those subproblems, and then combines the "sub-solutions" into a solution to the original problem. In this assignment, you'll apply the divide and conquer approach to solve two variations on a problem.

Deliverables:

GitHub Classroom: `https://classroom.github.com/a/M9J-uX0t`

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]

Python and Java are recommended because I am familiar with them.

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/1q6hvwL8VCNemhfc7`. 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/M9J-uX0t`. 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: Finding the Singleton Element

A *singleton* is a single item, as opposed to two (or more) items grouped together, as in the "singleton pattern", describing an object that may only be instantiated once, or a "singleton set", a set containing only one element.

Consider the following problem: you are given a sorted sequence of integers. Within this sequence, one and only one element is unique; the other elements are all duplicated once. For example:

$$(-2, -2, 5, 5, 12, 12, 67, 67, 72, 80, 80)$$

In your programming language of choice, implement an algorithm to find that lone, unique, singleton element. In the example above, your algorithm should return 72.

Your program must accept as a command line argument the name of a file containing a sequence as described above, then print to `stdout` the singleton element. For example:

```
>$ ./compile.sh
>$ ./run.sh in1.txt
72
```

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

As you solve this problem, recall that there are two broad types of divide and conquer algorithms: those that look like binary search and those that look like merge sort. Which of those approaches will lead to a more efficient algorithm for this problem?

Part 4: Dealing with Multiple Copies

Consider the following generalization of the problem: you are given a sorted sequence of integers. Within this sequence, one and only one element is unique; the other elements are duplicated *at least* once. For example:

$$(-2, -2, 5, 5, 5, 67, 67, 72, 80, 80, 80, 80)$$

Does this alteration affect your algorithm? If so, write new pseudocode accordingly, however, *do not* adjust your implementation to match — your program need only solve the original variation of the problem.

Part 5: Submission

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

- Pseudocode for efficient algorithms to solve both variations of the problem.
- Proofs of correctness and analyses of complexity for all pseudocode.

The following files are required and must be pushed to your GitHub Classroom repository by 8 pm 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 find the singleton element, as specified.

Any files other than these will be ignored.

Test run: On the date before the due date, the grader will be run after 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 submission:

- Late submissions made within 48 hours of the deadline receive no penalties.
- Late submissions made after 48 hours of the deadline receive a .8 multiplier. For example, if your late submission scores 80%, the final score will be 80% * .8 = 64%.
- If you decide to make a late submission, please send an email to vnguy143@calpoly.edu.

Resubmission: No resubmissions will be accepted. What you get on a submission (other than the test run) will be your final score. Please do not include the `compile.sh` and `run.sh` in your repository if you don't intend to submit yet.