Assignment 2 — Divide and Conquer

Due: Friday, October 4th

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/1jevVm_6`
Required Files:	`compile.sh`, `run.sh`
Optional Files:	`*.py`, `*.java`, `*.clj`, `*.kt`, `*.js`, `*.sh`

Part 1: 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 per Assignment 1, 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 2: 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 3: 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.