Assignment 2 — Divide and Conquer

A "divide and conquer" algorithm is one that divides a problem into smaller sub-problems, solves those sub-problems, and then combines the sub-solutions into a solution to the original problem. Depending on the original problem, this could reduce the number of sub-problems that must be solved, or it could merely reorganize the order in which they are solved.

1 Finding the Unique Element

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

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

Note that, because the sequence is sorted, any duplicate elements must appear consecutively. As a result, this problem could be solved in linear time by simply iterating over the elements of the sequence, checking each element to see if it is distinct from those immediately before and after.

2 A Binary-Search Based Approach

Further, suppose that all duplicated elements each appear exactly twice, exactly one element appears exactly once, and no elements appear three or more times. For example:

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

In the above sequence, the unique element is 72. In this situation, it should be possible to discard sub-problems which cannot possibly contain the unique element, in a manner similar to binary search.

Design and implement a divide and conquer algorithm to find the unique element within a sorted sequence of integers, where duplicated elements must each appear exactly twice. Your algorithm should have complexity $O(\log n)$, having avoided a substantial amount of potential work.

Your program must accept as a command line argument the name of a file containing a sequence as described (note that the format of the input file is different compared to Assignment 1, see the given example files), then print to `stdout` the unique element. For example:

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

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

3 A Merge-Sort Based Approach

Now suppose, as a generalization of the problem in Section 2, that duplicated elements could appear three or more times. For example:

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

In the above sequence, the unique element is still 72. However in order to find it, it is no longer possible to determine (at least not in constant time) which sub-problems must necessarily contain the unique element. As a result, all sub-problems must be solved, in a manner similar to that of merge sort.

Design (provide pseudocode) and analyze (prove your algorithm is correct and analyze its time complexity) a divide and conquer algorithm to find the unique element within a sorted sequence of integers, where duplicated elements may each appear two or more times. Your algorithm should have complexity $O(n)$, having achieved no improvement over the basic linear algorithm.

Note, *do not* adjust your implementation in Section 2, your program should only solve the original variation of the problem.

4 Submission

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

- Pseudocode for an efficient divide and conquer algorithm to solve the generalized problem in Section 3.
- A proof of correctness and analysis of time complexity for your pseudocode.

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 find the unique element.