L7

Problem 1 (Find the Rotation Point) *You have a rotated array, which means that it is "sorted" but the starting point is actually in the middle of the array. For example, the following array is rotated:*

$$[10, 14, 43, 61, 63, 87, 2, 6, 8, 9].$$

We would say this is rotated 6 spots, because the smallest entry is entry $A[6]$ and it increases from there. The rotation point would be $A[6]$.

(a) Design an $O(\log n)$ algorithm to find the index of the rotation point. Prove that it is correct, and that it runs in that time.

(b) (Bonus) Design an algorithm that, given some integer x, determines whether x is in the array. It should have the same runtime.

PROBLEM 2 (FIND THE FIXED POINT) *A fixed point in an array of integers A is an index i such that $A[i] = i$.*

(a) Array A has length n, and is sorted in increasing order ($A[1] < A[2] < A[3] < \ldots < A[n]$). Design a divide and conquer algorithm that either finds a fixed point or determines that one doesn't exist.

(b) Array B is unsorted. One thing you might think to do is simply sort the array and run the algorithm above. Is that a good idea? Why or why not?

(c) (Bonus) Array C is sorted in increasing order, and only contains positive integers. Design an algorithm that either finds a fixed point or determines that one doesn't exist. This algorithm should be asymptotically faster than your algorithm in part (a).

PROBLEM 3 (BACK TO DAY 1!) *As mentioned, two of the problems from Day 1 are divide-and-conquer problems. Go back to the slides from Day 1, and try to identify them.*

(a) One of the problems can be solved using the approach we discussed today. Figure out which one it is, and write an algorithm to solve it. Prove that it is correct, and analyze its runtime.

(b) (Bonus) Another problem on the list can also be solved using divide-and-conquer, but it's a bit more tricky. We'll discuss that problem Thursday. Take a guess on which problem it is! Why do you think it's that problem?

PROBLEM 4 (FIND THE MAJORITY) *You're helping count votes for an election. You only care if a candidate has more than half of the votes – otherwise everyone will have to vote again anyway. This is the algorithm you're going to run:*

Algorithm 1 Majority-Finder – only returns if a candidate got the majority.

```
1: tracker ← an empty array                          ▷ Stores vote counts as (candidate, count)
2: for each vote v do
3:     if v's candidate is not in the tracker then
4:         Add (candidate, 0) to tracker.
5:     end if
6:     Add 1 to the candidate's count.
7: end for
8: if some count is greater than n/2 then
9:     return that candidate
10: end if
11: return None
```

(a) This algorithm takes $O(n^2)$ time to run. Why?

(b) Design a divide-and-conquer algorithm that works in $O(n \log n)$ time.

(c) (Bonus) Describe how, using a different data structure for tracker, we can bring this runtime down to $O(n)$.

*(d) (Bonus) The algorithm above has what is called **space complexity** of $O(n)$ – it uses up to $O(n)$ space to solve the problem. After all, tracker can get up to size n. Write an algorithm that solves the problem in $O(n)$ time and $O(1)$ **space**.*