I collaborated with:

Problem

An array $A[1 \dots n]$ is said to have a *majority element* if **more than half** of its entries are the same. Given an array , the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form $A[i] > A[j]$. You should think of the array elements as, say, JPEG files. However, you *can* answer questions of the form: $A[i] = A[j]$ in $O(1)$ time.

(a) Show how to solve this problem in $O(n \log n)$ time. **Hint:** split the array A into two arrays A_1 and A_2 of half the size. Does knowing the majority elements of A_1 and A_2 (if such elements exist) help you figure out the majority element of A? If so, can you use a divide-and-conquer approach? Be sure to prove that your algorithm is correct (via induction) and give a recurrence for the running time of your algorithm.

(b) Can you design a linear-time algorithm? **Hint:** arbitrarily pair up the elements of the array. For each pair, if the two elements are different, discard both of them; if they are the same, keep just one of them. Show that after this procedure executes, there are at most $n/2$ elements left, and that they have a majority element if A does.

Solution

Algorithm 1 A

```
function MAJORITY(A)
    if A has only one element then
        return (A[1],1)
    else
        A_1 ← first half of A
        A_2 ← second half of A
        majority_1 ← Majority(A_1)
        majority_2 ← Majority(A_2)
        if either subarray does not have a majority, then A does not have a majority
        for i in A_2 do
            if i == majority_1[0] then
                increase the count of majority_1
            end if
        end for
        for i in A_1 do
            if i == majority_2[0] then
                increase the count of majority_2
            end if
        end for
        if majority_1[0] == majority_2[0] then
            return majority_1[0] and sum of their counts
        else if both majorities have the same count then
            We do not have a majority
        else
            return the majority with highest count if count > n/2
        end if
    end if
end function
```

Proof. Base Case: If our array A has one element, then such element is more than half of its entries. Also, our algorithm will return this element as well. The algorithm holds for this case.
Inductive Hypothesis: Assume our algorithm works for an array of size i where $1 \leq i \leq k$ for some integer k.

Inductive Step: Let A be an array of size $k+1$. We must show our algorithm returns the correct majority for A. Based on our inductive hypothesis, the algorithm returns the correct majority (if one exists) of each half, A_1 and A_2, of A as the halves are of size less than $k+1$. If it is the case that either subarray does not have a majority, then A does not have a majority. Let us see the case where both A_1 and A_2 have majorities. We must see if A_1's majority is an element of A_2 and vice versa. If we see said elements in the other array then we update their count in A accordingly. Now, if A_1 and A_2 have the same majority, then it is the case that A has the same majority, so we aggregate their count. However, if both majorities are different but have the same count then either majority cannot be a majority of A as their count will not be over half of the elements of A. Now, we check if the majority with the larger count occurs more than twice the size of A. If so, we return the larger majority. □

$T(n) = 2T(\frac{n}{2}) + cn$

Algorithm 2 B

```
function MAJORITY(A)
    arbitrarily pair up the elements of the array
    for each pair (i, j) in A do
        if i ≠ j then
            Discard them both
        else
            Discard one of them
        end if
    end for
    Check if Majority of the rest of A is a majority in A.
    if so, return it
end function
```

Proof. Suppose A has a majority. This means that the majority m occurs more than $n/2$ times. If m is paired with itself, then there will be more pairs of m than of any other element. If m is paired with other elements, then minorities and some elements m will be thrown out but pairs of m will remain as there are more than $n/2$ m's. This subarrays has the same majority the original array. We must see if a candidate majority is the actual majority by running through the array and seeing if it occurs more than half of the time. Run Time: $O(n) + O(n) = O(n)$ □