

<p>Name: Harshit Jain Access ID: hmj5262 Recitation: 8</p>

Problem 0

Points:

Acknowledgements

- (a) I did not work in a group.
- (b) I did not consult without anyone my group members.
- (c) I did not consult any non-class materials.

Problem 1

Points:

Divide-and-Conquer

- (a) We will first have the function which returns the frequency of the element in the given list. This function will take $O(n)$ time.

```

1  def frequencyCalculator(Array, element):
2      count = 0
3      for ele in Array:
4          if (element == ele):
5              count += 1
6      return count
7

```

Now, to start, we will split the array A into 2 subarrays A_1 and A_2 of half the size. Then we will calculate the majority elements of A_1 and A_2 . The algorithm is as follows:

```

1  def majorityElement(Array, low, high):
2
3      subArray = Array[low:high+1]
4
5      # base case
6      if (len(subArray)==1):
7          return subArray[0]
8
9      mid = (low+high)//2
10     leftMajorityElement = majorityElement(Array, low, mid)
11     rightMajorityElement = majorityElement(Array, mid+1, high)
12
13     if (leftMajorityElement == rightMajorityElement):
14         return leftMajorityElement
15
16     leftFrequency = frequencyCalculator(subArray, leftMajorityElement)
17     rightFrequency = frequencyCalculator(subArray, rightMajorityElement)
18
19     if (leftFrequency > len(subArray)//2):
20         return leftMajorityElement
21     elif (rightFrequency > len(subArray)//2):
22         return rightMajorityElement
23     else:
24         return -1 # no majority element found
25

```

- (b)
- (c) Here, we are choosing the majority element of sub-arrays after dividing them in half. Moreover, frequency calculation is using $O(n)$ time as shown above. Therefore, the recurrence relation for this algorithm is given by:

$$T(n) = 2T(n/2) + O(n)$$

According to Master's Theorem, the time complexity: $O(n \log n)$

Problem 2

Points:

Reverse graph

(a)

(b)

Problem 3

Points:

Graph Basics

- (a)
- (b)
- (c)
- (d)
- (e)