Programming Assignment 2

1. The Problem

You are given an unsorted array A of n ≥ 1 integers and a positive integer k. An element is called

special if it occurs at least k times. If there is a special element in A, output it. Otherwise, output the string “None”

1. Correctness

Because the function uses merge sort the only additional code that contributes to correctness is the equality checking and frequency counting. The equality checking is trivial there is simply a comparison between the current elements from the left and right arrays. Because all elements will need to be compared for the algorithm to terminate every possible element will be checked for equality. The frequency counting is however, less trivial. I used a simple hashing function to identify the various frequencies among the array. Each element that is found to occur at least 2 times will have its modulo 100 calculated and this remainder will exactly correspond to its index in the frequency array. This method allows for many different numbers to be in the input array and have them all counted separately without having to iterate through an array, increasing the complexity. Because the array in the code is hardcoded to 100 individual values if there are input values greater than or equal to 10000 there will start to be hash collisions and the correct value may not be returned every time. This problem could easily be resolved by increasing the size of the input array or generating it dynamically using numpy or similar. However, all the inputs in testing are of small enough size to avoid collisions with this fixed array.

1. Running Time

Because the find\_special function uses the same splitting and merging as merge sort it has a worst-case performance of O(n log n). This worst case is when there is no special value in the array and as such the entire merge sort will run through resulting in n log n operations. The additional checks for equality, modulo calculations, and frequency assignment are negligible on this efficiency.

The Best case of this function would be given a sorted array with the special element in at least the first k positions. This will result in a performance of O(n). This can be demonstrated with the example array given by Khalid in Zulip; k = 2, n = [1, 1, 2, 3, 4]

This will require only 1 comparison after the splitting. Upon comparing and merging 1 and 1, the critical value k is met and thus the function exits. In this case it only takes 1 comparison, however as the value of k changes and thus the size of the array the number of comparisons needed to reach k will increase linearly with the size of the input, thus it cannot be O(1) but instead is bounded by O(n) in the best case.

1. Pseudocode

Global count <- array of 100 zeros

Global value <- 0

Func find\_special(array, k, count)

If(length(array) > 1)

Mid <- length(array) / 2

left\_half = array[0,mid]

right\_half = array[mid, end]

find\_special(left\_half)

find\_special(right\_half)

counter1, counter2, counter3 <- 0

while counter1 < length(left\_half) and counter2 < length(right\_half)

if left\_half[counter1] == right\_half[counter2]

count[left\_half[counter1] modulo 100] += 1

if count[left\_half[counter1] modulo 100] >= k

value <- left\_half[counter1]

return

if left\_half[counter1] < right\_half[counter2]

array[counter3] <- left\_half[counter1]

counter1 += 1

else

array[counter3] <- right\_half[counter2]

counter2 += 1

counter3 += 1

while counter1 < length(left\_half)

array[counter3] <- left\_half[counter1]

counter1 += 1

counter3 += 1

while counter2 < length(right\_half)

array[counter3] <- right\_half[counter2]

counter2 += 1

counter3 += 1

return