# Michael Chillemi

# 07/15/2023

# CS 590 - Algorithms

# M8.D1: Midterm Exam

**Problem 1.**

The definition of big omega notation is that if the > 0.

This means that there exists some and some such that for all , we must have .

In this particular case, and . We choose and . Then the statement follows because for all , we have .

**Problem 2.**

Claim:

Base Case: if then our formula is true for

For all values assume is true

Inductive Step:

By inductive hypothesis:

Therefore

Hence

is true.

Therefore, by the principle of mathematical induction, is true.

**Problem 3.**

A sorting method for in-place data is not a bucket sort. This is due to the fact that sorting array elements requires additional space for buckets. Therefore, sorting an array of n elements requires, in the worst case, additional O(n) memory. Bucket sort is not an in-place algorithm, to sum up.

**Problem 4.**

Algorithm countingNonZeroNums(array[x][x]):

Input: array[][]

Output: numCount

newCount = 0

numCount = 0

i =1

for i < x

middle = n / 2

numCount = newCount + BinarySearchAlgo(array, 0 , x)

return numCount

Algorithm BinarySearchAlgo(array, lowNum, highNum):

Input: array, lowNum, highNum

Output: middleNum - 1

middleNum = (lowNum + highNum) / 2

if array[middleNum] == 0 and array[middleNum - 1] == 1

return middleNum -1

else if array[middleNum] == 1

return BinarySearchAlgo(array, middleNum +1 ,highNum)

else

BinarySearchAlgo(array, lowNum ,middleNum -1)

**Additional Justification for Pseudo Code:**

The first function will use the binary search approach while looping over the array size. The array's non-zero elements will be counted via the binary search method, and on each loop, the previous count of those elements is added. This has an overall time complexity of . Since n is the complexity of a for loop and log n is the complexity of each binary search method, the total complexity of this algorithm is .

**Problem 5.**

Sort the data by non-decreasing characteristics. The extent of each subsequence of identical numbers in the sequence can then be determined by looking at the series as it proceeds. We may also keep track of the length of the longest succession we've witnessed so far. This is the mode when the output of the sequence has finished. When sorting the succession, which should be doable in O(n log n) time in the worst case scenario using the union sort calculation.

**Problem 6.**

Algorithm identifyDistinctChar(x):

Input: x

Output: C

charCount[256]

i =1

for i < x

charCount[x.charAt(i)]++

j = 0

for j < 256

if charCount[i] == 1

C.add(char)

return C

For this algorithm the time complexity it will take to scan the string is . Additionally the time complexity to add the characters in set C will take time. In total the time complexity for the identifyDistinctChar() algorithm will run is time.

**Problem 7.**

In a social network, N, which contains n people, m edges, and c connected components. If n represents a singleton set, the union() operation will execute times, makeSet() will execute n times. Lastly the find() function will run .

**Problem 8.**

**(Just to note I was not sure if this question meant to write pseudo code for the solution it did not specifically say it like in question 6. But I was pressed for time and found it more efficiently for me to write out a complete solution)**

The following steps can be used to implement the technique for finding the kth lowest element in a set of n different numbers in O(n+k log n) time:

**Step 1.** Create a minimum heap using the n different integers. This process requires O(n) time.

**Additional Justification for Building Min Heap:**

We must first create a min heap using the set of n unique numbers. A min heap is a binary tree-based data structure with the smallest value at the root node and the following smallest value at each of its children. By inserting each component of the input set one at a time, the min heap can be created in O(n) time.

**Step 2.** Take the root node k-1 times out of the min heap. As each extraction requires O(log n) time and we must complete this operation k-1 times, this step requires O(k log n) time.

**Additional Justification for Extraction Of The Root Node Times:**

The min heap's root node must then be extracted k-1 times. The input set's kth smallest element will be the kth smallest element taken from the min heap as a result of this action. Because the root node must be replaced with the last member of the min heap and the min heap must be rebalanced to maintain its min heap property, each extraction takes O(log n) time to complete. As k extractions must be carried out, the overall time complexity of this phase is O(k log n).

**Step 3.** As the kth smallest element, give back the kth extracted node.

**Additional Justification for Reintroducing The k Extracted Node:**

Finally, we return the input set's kth smallest element, which is the kth extracted node. This step's temporal complexity is O(1) because it just entails returning a single value.

This algorithm's total time complexity is O(n + k log n). The total time complexity is O(n + k log n), with the first step taking O(n) time and the second step taking O(k log n) time. It should be noted that the technique described above makes the assumption that the input set of n distinct numbers is free of duplicates. In the event of duplication, a new strategy would be required.