ALGORITHMS

# Section B

**Time Complexity Analysis**

**Submitted By**

**Vishad Upmanyu (45)**

**Yashvi Verma (47)**

**Question 6,**

**Since, the approach followed is divide and conquer.**

**Function divideInput(int \*, int, int)**

**Divides the input into equal halves.**

**Function check(int, int \*, int, int)**

**Checks whether the element is in majority or not. It takes O(n) time.**

Let us denote the running time of the algorithm on array of size ‘n’, by T(n).

Then,

T(n) is,

T(⌈n/2⌉) + T(⌊n/2⌋) +O(n), if n>2

O(1), if n≤2

The master theorem shows that T(n)=O(nlogn).

**Question 7,**

**We initialize minimum with first element of price array and the difference bewtween 2nd and 1st element to be the maximum difference.**

**After that, we iterate through the price array and update values of max\_diff if any difference is more than current max\_diff.**

**Similarly, the value of minimum is updated as we iterate across the array. The sell and buy date is recorded only when max\_diff is updated and not when the minimum is updated, because of real-life possibilities (buy date has to come before sell date). The final maximum difference between buying price and selling price is used to return the optimum buy and sell date.**

**ASSUMPTION: It is assumned that the merchant can do only one transaction in the given time period.**

**Complexity**

Function 1:

line 1 -> O(1)

loop 2-3 -> O(n) linear time

line 4 -> O(1)

Function 2:

line 1 -> O(1)

line 2-3 -> O(n)

line 4 -> O(1)

Time complexity = O(n)+O(n) =O(n)

**Question 8,**

**To minimize the cost, license with most expensive rate of growth must be bought first. The solution is to find maximum, then second maximum, then third maximum and so on. Since we are interested in immediate best choice, it is a greedy algorithm. It can be achieved by sorting the array which will take at best O(nlogn) [merge sort, heap sort]. We were given the flexibility to find solution in polynomial of n time. So, here is a more ituitive solution.**

**The approach is to find maximum of entire array, print the license number and swap it with the beginning of array.**

**Now, to find second maximum find the maximum of l\_arr[2..n], since the first maximum is stored at location l\_arr[1].**

**Similarly find subsequent maximum and print it.**

**Total no. of software license to buy = n**

**The max\_calc() function works in linear time,**

**line 1, 5-7 take constant time.**

**loop line 2-4 of function1 iterates n times.**

**print\_order() calls the function max\_calc() n times, function2 line 1-2.**

**Therefore, total time complexity is O(n2).**

**Time complexity: O(n2)**

**Question 9,**

The simple solution of the problem is, send the contestants in decreasing order of the sum of projected biking and running time.

Since, the solutions just adds the projected biking and running times, then sorts the total in decreasing order.

The algorithm takes T(n)=O(n2) time, as sorting is done using insertion sort.

**Question 10,**

**Initially, zero people are removed. Check for condition 1 and 2 for every person. Whenever a failure occurs remove that person and start all over again with remaining persons. Stop when no one will fail the conditions or everyone fails it.**

**Function 1:**

**line 1 -> O(1)**

**loop 2-4 -> O(n) linear time**

**line 5 -> O(1)**

**Function 2:**

**line 1 -> O(1)**

**line 2 -> j iterates from 1 to n,**

**line 3 -> O(n) it makes a call to count known in every iteration**

**line 4 -> O(1)**

**line 5 -> if the condition in this line is satisfied, the value of j is again set to 1.**

**which implies there will be more than n iterations in line 2. In worst case value of j is set to 1 by every value of j,**

**Therefore, the main loop runs at most n^2 times.**

**line 6-8 -> O(1)**

**line 9 -> if this condition is true, we will come out of the main loop. and line 10-11 will iterate n times but only once, hence it will not affect the degree of time complexity**

**--------------------------------------------------**

**Since, count known is called at most n2 time.**

**Time complexity = O(n3)**

**Question 11,**

**Consider all ways of partitioning n people in groups of atmost m people and atleast k people.**

**anti\_squad(n,m,k) gives this value.**

**If we have no upper bound of people in a group, we need to call anti\_squad(n,n,k)**

**Total no. of people = N**

**Minimum no. of people required in each group = K**

**Recurrence relation:**

**T(n,m) = T(n-m,m) + T(n,m-1) ; n, m <= N**

**T(0,m) = 1**

**T(n,0) = 0**

**T(n,m) = 0 ; n < K or m < K**

**The base cases take constant time - O(1).**

**Size of memoize\_table is N x N.**

**We will be calculating values of memoize\_table[n][m] for n,m <= N only once since we are**

**returning immediately if its value is already computed.**

**Therefore, total time complexity is O(N2).**

**Time complexity: O(N2)**