GTU

**DEPARTMENT OF COMPUTER ENGINEERING**

**CSE 321 – Autumn 2022**

**HOMEWORK 5  
REPORT**

SÜLEYMAN GÖLBOL 1801042656

# LONGEST COMMON SUBSTRING

For the first problem, I created a function called longest\_common\_str to divide array into 2. It calls a function called longest\_common\_str\_between\_2() and this function takes 2 string parameters for comparison. For parameters I give longest\_common\_str(half of array) to it so longest\_common\_str() becomes recursive. This way the problem is divided into 2. For the time complexity of this problem;

In each step algorithm divides strings array into 2 half parts. Longest\_cmmon\_str\_between\_2() takes O(n) time because of for loop.

So, T(n) = 2T(n/2) + n. This way time complexity is O(nlogn)

1. **MAXIMUM PROFIT**
2. To solve the profit problem with divide and conquer, I divided array into two halves and called the same function again for half arrays recursively. But maximum profit can be in left array right array or just the difference between max in right and min in left. This way time complexity is;  
   T(n) = 2T(n/2) + 1. Using master theorem time complexity is O(n)
3. To solve the problem without divide and conquer; I made minimum as first element then updated when occur to a smaller value. And to find the maximum profit, I found the difference with minimum so if it’s bigger than current profit; I updated the profit. Because of only one for loop time complexity is O(n).
4. In the worst case; both algorithms need to compute all the elements so both will be O(n).

**3) LONGEST INCREASING SUBARRAY**

# In this problem, I realized that I could create an array to store the values of lengths maximum. I made the name of the array as length\_storer\_array.

# Then in a for loop, I checked if element is bigger then previous array element. If so; I added the value to length\_storer\_array so we don’t have to compute subproblem again.

# Then from this array I returned the maximum one. So, time complexity is because of for loop or max function; it’s O(n).

# 

**4) HIGHEST POSSIBLE SCORE**

a) For dynamic programming approach; I created an array T for total score got until that point. This T has the same size with array. I initialized first element with first array value for first row and column so that it will start from there. Then, I added the sum of points of previous to the T array so that I can see the total points. Then for each I checked for the max and updated the array. To print path and points I also created 2 different arrays(lists) for them. Because of 2 nested for loops time complexity is O(n^2)

b) For greedy approach, I just checked for the local optimal not for the global optimal. So this makes the algorithm may not find the global optimum solution. Because of 1 while loop time complexity is O(n).

c) When we compare these and homework 4 brute force solution; brute force solution and dynamic programming solution gives the global optimum score. Because we can see all the maximum scores until the end. But greedy approach only looks for the local optimum so it may not be the global optimum always. But for the worst case time complexities brute force approach has 2^(n+m) time complexity which is bad since it’s exponential. Dynamic programming has O(n^2) which is better than brute force and greedy approach has linear O(n) complexity which is better than other 2.