

## National University of Computer and Emerging Sciences, Lahore Campus



<b>Course:</b>	<b>Design and Analysis of Algorithms</b>	<b>Course Code:</b>	<b>CS2009</b>
<b>Program:</b>	<b>BS(Computer Science)</b>	<b>Semester:</b>	<b>Spring 2022</b>
<b>Duration:</b>	<b>30 Minutes</b>	<b>Total Marks:</b>	<b>15</b>
<b>Paper Date:</b>	<b>28-April-2022</b>	<b>Weight</b>	<b>%</b>
<b>Section:</b>	<b>4A</b>	<b>Page(s):</b>	<b>4</b>
<b>Exam:</b>	<b>Quiz 3</b>		

**Q1)** The semester is over! You've rented a car and are ready to set out on a long drive on the Strange Highway. There are  $n$  tourist stores on the Strange Highway, numbered 1, 2, ...,  $n$  and you would want to stop at these and buy some souvenirs (only one souvenir may be bought from a store and all souvenirs everywhere have the same price). You are a greedy shopper and are most interested in maximizing the total discount you get on your shopping. You know that each store  $i$  offers a discount  $d_i$  on a souvenir. But it's a strange highway after all. It turns out that you can only buy a souvenir from a store  $i$  if you have not bought anything from the previous  $f_i$  stores. For example, if  $f_6 = 3$  then you can only buy a souvenir from store 6, and get the discount  $d_6$ , if you haven't bought anything from stores 3, 4, and 5. All the  $d_i$  and  $f_i$  are known to you in advance (passed as input). You have recently learnt the DP technique in your algorithms course and wish to apply it here in order to maximize your total discount under the given constraints. I will help you by defining the optimal sub-structure. [6 Marks]

$D[i] = \text{max total discount when store } i \text{ is the last store where you buy a souvenir.}$

Example

i	f(i)	d(i)	D(i)
1	0	2	2
2	0	4	6
3	2	2	2
4	2	5	7

(a) Provide recurrences for  $D[i]$ .

$$D[i] = \max_{1 \leq k < (i - f_i)} \{D[k]\} + d_i$$

- (b) Provide complete pseudo-code of the bottom-up DP algorithm based on the recurrence above.

```
int Function( char *d, char *f, int n )
{
    D[1] = d[1]
    for (i = 2 to n)
        max = -1
        for (k = 1 to (i-f[i]))
            if (D[k] > max)
                max = D[k]
        D[i] = max +d[i]

    max = -1
    for (k = n to (n-f[n]))
        if (D[k] > max)
            max = D[k]
    return max
}
```

Q2) Consider the following recursive algorithm. [2 + 5+ 2 = 9 Marks]

*/\* m and n are lengths of char arrays X and Y respectively \*/*

```
int Function( char *X, char *Y, int m, int n )
{
    if (m == 0 || n == 0)
        return 0;
    if (X[m-1] == Y[n-1])
        return 1 + Function(X, Y, m-1, n-1);
    else
        return max(Function(X, Y, m, n-1), Function(X, Y, m-1, n));
}
```

(a) What is time complexity of above algorithm?

$O(n^2)$  or  $O(m^2)$

(b) Convert the recursive code given above into bottom up iterative dynamic programming algorithm.

```
int Function( char *X, char *Y, int m, int n )
{
    for (i = 1 to n)
        A[i][0] = 0
    for (j = 1 to m)
        A[0][j] = 0

    for (i = 1 to n)
        for (j = 1 to m)
            if (X[m-1] == Y[n-1])
                A[i][j] = A[i-1][j-1] + 1
            Else
                A[i][j] = max (A[i][j-1] , A[i-1][j])

    return A[m][n]
}
```

(c) What is time complexity of your iterative algorithm?

$O(m*n)$

Name: \_\_\_\_\_

Reg #: \_\_\_\_\_

Section: \_\_\_\_\_