Design and Analysis of Algorithms

Q1) Consider the following recursive algorithm

```
K(n){
    IF (n = 1)
        RETURN 1
    ELSE
        SUM = 0
        FOR( $i = 1 \ to \ n - 1$ )
            SUM = SUM + (K($i$) + K($n - i$))/3 + n/2
    RETURN SUM
}
```

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

Q2) Consider the following recursive algorithm

```
P(n, k){
    IF (k > n)
        RETURN 0
    IF (k == n || k == 0)
        RETURN 1

    RETURN   P(n-1, k-1) + P(n-1,k)
}
```

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

Q3) Given a matrix M * N of integers where each cell has a cost associated with it, the cost can also be negative. Find the minimum cost to reach the last cell (M-1, N-1) of the matrix from its first cell (0,0). We can only move one unit right (Column No + 1), one unit down (Row No + 1), and one unit in bottom diagonal (Row No + 1, Column No + 1). For example from index (i , j) you can move to (i , j+1), (i+1 , j), and (i+1, j+1) where i = row no and j = column no.

Example

4	7	8	6	4
-6	7	3	9	2
3	8	1	-2	4
7	1	7	3	7
2	9	8	9	3

4	7	8	6	4
-6	**7**	3	9	2
3	8	**1**	**-2**	4
7	1	7	**3**	7
2	9	8	9	**3**

Path with minimum cost = 4 -> -6 -> 7 -> 1 -> -2-> 3 -> 3 = 10

Provide a Dynamic Programming solution for it.

a) Provide recurrence for sub-problem

a) Provide pseudo code for DP solution

Problem 4

You are going to Europe by road! Wow. And its one single road (wow!). You start on the road at distance d_1 =0. There are many stopovers on this roads and these are located at distances $d_1 < d_2 < \cdots < d_n$. You are driving the car yourself and will need to stop at some of these stopovers. In the end, you will definitely stop at the final stopover at distance d_n.

Your aim is to travel 500 kms every day (you make stopover at night), as this seems to be the ideal distance that balances the fatigue with the pleasure of travelling the most. However, it's not always possible to do so because the stopovers are not always 500 kms apart from each other. (If they were, you could travel 500 Kms and make a stop, and so on). Being of a quantitative bent of mind, you have formulated that if you travel α miles in a day, the amount of 'loss' (in terms of more fatigue and less joy) for that day is $(500 - \alpha)^2$.

You want to make stops such that this total loss is minimized — that is, the sum, over all travel days, of the daily losses is minimized. Give an efficient DP algorithm that determines the optimal sequence of stopovers at which to stop so as to minimize loss.

(a) Define a subproblem
(b) Write a recurrence
(c) Write a bottom up algorithm to compute the subproblem solutions
(d) Write a method to retrieve the optimal stopover sequence that produces the optimal (minimum) value of loss.

Problem 5

Now at this point you have become an algorithm expert and it’s the right time to increase the difficulty level. You have given an array M = {1 ,2, 3, 8, 6, 7, 9, 10, 2, 1}. You have to calculate the longest increasing with decreasing subsequence in the array. Let’s understand what a subsequence is through an example.

1, 3, 6, 10, 1 à Subsequence

1, 2, 3, 5 à Sequence

There is a subtle difference between subsequence and sequence. In Subsequence we can skip some elements but it should be done in increasing order, like 1 3 2 5 6 is not subsequence. Order of elements in parent array should remain same in subsequence. Like here 3 comes after 1 so there exist no subsequence in which 3 comes before 1.

In the problem you have to find such subsequence which has 2 parts, whose first part is **increasing** in order and second part is **decreasing** in order and it is the ***longest*** subsequence.

Here, (red is increasing subsequence and green is decreasing subsequence)

i) 1 2 3 8 9 10 2 1
ii) 1 2 3 6 7 9 10 2 1

are one of such subsequences but we can see that second subsequence is longest so we will choose this one

1) Define a subproblem for it.
2) Provide its recurrence.
3) Pseudo code for the algorithm devised.
4) Run time complexity

Problem 6

You are fascinated with stars and your friend challenges you with the algorithms question. In Problem you are given a m * m matrix containing only 1 and 0. Your challenge is to find biggest star in matrix in min amount of time and computations. For example in the following matrix the biggest start is highlighted, it is M[5][4] with length = 3. The star at M[5][4] entry with length 3 means all of the following entries have 1.

M[5][3] , M[5][2], M[5][1], //same row, previous columns

M[5][5], M[5][6], M[5][7], //same row, next columns

M[4][4], M[3][4], M[2][4], //same column, previous rows

M[6][4], M[7][4], M[8][4], //same column, next rows

M[4][3], M[3][2], M[2][1], //diagonal

M[6][5], M[7][6], M[8][7], // diagonal

M[6][3], M[7][2], M[8][1] // diagonal

M[4][5], M[3][6], M[2][7], // diagonal

1	1	0	1	1	0	1	1	0	1
1	1	1	0	1	1	1	0	1	0
0	1	0	1	1	1	0	0	0	1
0	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	0
1	0	1	0	1	1	1	0	0	1
0	1	0	0	1	0	1	1	1	1
0	1	1	1	1	0	0	1	0	1

A) DP Recurrence

B) Time Complexity for DP Solution

C) Time Complexity for Naive Solution

<u>**Greedy Algorithms**</u>

Q7) Consider the problem of making change from n cents using the fewest coins when the available coins are quarters (25 cents), dimes (10 cents), nickels (5 cents) and pennies (1 cent). Does the greedy strategy of outputting the largest coin that does not exceed the amount of change that must still be returned yield an optimal solution?

Q8) You are given a sequence of n songs where the *ith* song is l_i minutes long. You want to place all of the songs on an ordered series of CDs. (e.g. CD 1, CD 2, CD 3, ….., CD k) where each CD can hold m minutes. Furthermore,

1. The songs must be recorded in the given order, song 1, song 2, ... song n
2. All songs must be included
3. No song maybe split across CDs.

Your goal is to determine how to place them on the CDs as to minimize the number of CDs needed. Give the most efficient algorithm you can to find an optimal solution for this problem, analyze the time complexity.

Q9) The input consists of *n* skiers with heights $p_1, ..., p_n$, and *n* skies with heights $s_1, ..., s_n$. The problem is to assign each skier a ski to minimize the AVERAGE DIFFERENCE between the height of a skier and his/her assigned ski. That is, if the skier *i* is given the ski a_i, then you want to minimize:

$$\sum_{i=1}^{n} \frac{\left| p_i - s_{a_i} \right|}{n}$$

(a) Consider the following greedy algorithm. Find the skier and ski whose height difference is minimized. Assign this skier this ski. Repeat the process until every skier has a ski. Prove of disprove that this algorithm is correct.
(b) Consider another greedy algorithm. Give the shortest skier the shortest ski, give the second shortest skier the second shortest ski, give the third shortest skier the third shortest ski, etc. Prove of disprove that this algorithm is correct.
HINT: One of the above greedy algorithms is correct and one is incorrect for the other.