Question 1:

Write an efficient algorithm (to the best of your Knowledge) for the following problem, briefly describe why it is a correct algorithm, provide pseudocode, and analyze the time complexity

Problem:- Robot Walk

Input:- An array A[0...n, 0...n] of positive integers. A robot that starts at position A[0,0] and wants to reach A[n,n]. At each step, the robot can only move on step either rightward, along the diagonal, or downward. For example, if the current position is (i,j) then the next possible positions are (i,j+1), (i+1, j+1) or (i+1, j). The cost of a path is the sum of the entries that the robot visits.

Output:- The most expensive path that the robot can take

Example: Input A=

50 (robot Starts at A[0,0])	20	30
10	10	10
10	40	40 (robots end at A[n,n])

Output: 160 (because of the path 50 —> 20 —> 10 —> 40 —> 40

Solutions

For the Algorithm:-

Step 1:-

— Create a dynamic programming positive array of a n x n array and the array name is an.

Step II:-

— Will rotate the array A from bottom right to the top left: starting position of the robot (i,j) in A[0][0]

Step III:-
If the current position is (i,j) will store an[i,j+1] + A[i,j] cell of an array

Step IV:-
Will just use a for loop to loop through the rows and columns and will store the last column as an[i+1, j] + A[i,j] in (i,j) of an array and will return the most expensive path that the robot can take.

pseudo code:-

The pseudocode follows as:-

```
A[ ]{ }
an[n+1][n+1]            // positive array of n+1 using dynamic programming
for n ≥ i ≥ 0 will do
    for n ≥ j ≥ 0
    if (j==n & i==n)
        an[i][j] = A[i][j]
    else if (j < n & i==n){
        an[i][j] = an[i][j+1] + an[i][j]
    else if (j == n & i < n){
        an[i][j] = an[i+1][j] + A[i][j]
    else if (j < n & i < n)
        an[i][j] = max(an[i+1][j], an[i][j+1], an[i+1]
                       [j+1] + A[i][j])
        }
    }
}
will then return the an[0][0] }
```

Time analysis complexity:-
The time complexity is $O(n^2)$ due to the nested loop, it loops through (n+1) times and there is also an extra 2d array been used. $T(n) = O(n^2)$

Question 2:-

You have N dollars and you have a list of K items $L_1 \ldots L_K$ that you wish to purchase from an online store. If you purchase an item, then the online store gives you some reward points. For an item L_i, $1 <= i <= K$, the price and reward values are P_i and R_i respectively. Both P_i and R_i are integers. Unfortunately, there is a threshold T on the total reward you can earn. In other words, if the sum of reward values for the items you purchase is more than T, then any reward point beyond T will be wasted. Write an efficient algorithm (to the best of your knowledge) to find a list of items within N dollars such that purchasing them will maximize the sum of reward values but will keep the sum within the threshold T (Note: the sum can be equal to T). Briefly describe why it is a correct algorithm, provide pseudocode and analyze the time complexity.

Example: Input N=10, K=3, T=100, P=[4,6,5]
R=[40,70,50]
Output: L1, L3
Here is an explanation. Note that you have the following choices:
{L1} where price 4 and reward 40. {L2} where price 6 and reward 70. {L3} where price 5 and reward 50. {L1, L2} where price 10 and reward sum 110 (exceeding T). {L1, L3} where price 9 and reward 90. {L2, L3} where price 11 (exceeding N) and reward 120 (exceeding T).
Thus the best option that maximizes the reward sum is L1, L3

Solution

Algorithm:
- I will need to create a dynamic programming table

Step II:- Will use a for loop to determine/check ~~[illegible]~~ the price and rewards values

Step III:— I will look for the sum of reward values but will try keeping the sum within the threshold T.

Step IV:- will return the {L1, L3} of the reward sum

I will assume let $y(i,j)$ be the maximum sum of the first i of the item; $1 < i < K,\ K\ j \le N$

So there are multiple choices:- $y(i-1, j)$ and $y(i-1, j-ji)$

$$y(i,j) = \begin{cases} 0, & \text{if } j = 0 \\ 0, & \text{if } i = 0 \\ y(i-1,j). & \text{if } j > N \end{cases}$$

Pseudocode

```
(i)   for j = 0 to N
          y[0, j] = 0
(ii)  for i = 1 to K
          y[i, 0] = 0
(iii) for i = 1 to K
          for j = 0 to N
              if (ji > j)
                  y[i, j] = y[i-1, j]
              else y[i, j] = max{y[i-1, j], ji + y[i-1, N-ji]}
      for i = K to 0
          for j = N to 0
              if (y[i x j] ≤ T)
                  return y[i, j]
```

For Input: N = 10, K = 3, T = 100, P = [4, 6, 5], R = [40, 70, 50]

{L1} — price 4 → 40; {L2} = - price 6 → 70;

...mic Table:-

Price	1	2	3	4	5*	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	40	40	40	40	40	40	40	40
2	0	0	0	40	50	50	50	50	90	90	90
3	0	0	0	40	50	50	50	50	90	110 exceed	120 exceed

L1 = 40

{L1 L3} - 9 ⇒ 90

Scanned with CamScanner

Time Analysis Complexity:

The time complexity is O(N·K) items

Using through the dynamic programming table, the value does not exceed the threshold and the complexity is N·K Big O Notation and also the recursive is O(1) times

So, therefore the complexity analysis is O(N·K)

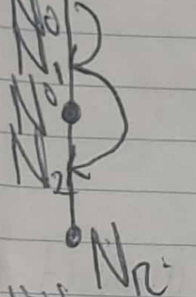
Question 3:-

Algorithm:-

In this aspect, there are two integers, will output the steps as follows:-

— I will want to use Greedy algorithm because is the safest choice. Will have to refill at the shortest reachable gas station that's nearby

— I will assume N^o to be the reachable gas station



Will assume N^o to be the first refilling station in the solution, and N_2 is the second and the nearest to N^o

— The algorithm will be best solution and will follow:-

Base case:- Will be only one gas station

Inductive hyp:- If the set of the gas station is less that n, $i<n$; then the algorithm will be the greedy algo and will be the best solution

Inductive step:- Can also refill at N_1 and ignore N^o; I can then focus on the rest of the remaining gas station. Here, we have $\leq$ than the n $i<n$; N and can just use say the greedy is safe and returns the best solution.

$n=6$
$C=40$
$D=[10,20,15,5,5,20]$
$10-10-4$
$20+15+5=44$
$5+20=46$

Pseudocode

for j = 0 to n
will add the distance
if distance ≤ n
else set distance to 0
~~end~~[j] = true;
j - 1 {decrease}

for j = 0 to n
if ~~end~~ end[j] is true
will print on the console j

Time analysis complexity

The time complexity is O(n)

the first and second for loop will return the n number of times and it takes O(1) and the T(n) = O(n).