VISWA NIHAR NUKALA

50414392

① Optimal path needs to be found from one position to another where p(x,y) is the cost to go to another square.

Following is the algorithm for the above problem:-

```
1.  Maximum Optimal Checker (P, n, i1, j1, i2, j2):
2.      for i = n-2 to 0:
3.          for j = n-1 to 0:
4.              max = -∞
5.              if (j == 0):
6.                  max = max { p[i+1, j], p[i+1, j+1] }
7.              else if (j == n-1):
8.                  max = max { p[i+1, j-1], p[i+1, j] }
9.              else:
10.a                max = max { p[i+1, j-1],
10.b                          p[i+1, j],
10.c                          p[i+1, j+1] }
11              p[i, j] = p[i, j] + max
12.     return P[i2, j2]
```

In the above algorithm i_1, j_1 and i_2, j_2 are the initial and final positions of the checker and P is the cost matrix.

We are the finding the cost from one box to another for all the possible values by finding the maximum of the 3 places from where the checker can be moved and are updating the value.

Finally we are returning the maximum cost of the final position we want our checker in.

Time complexity analysis:-

Since there are 2 nested for loops which one which runs for $n-2$ times and the other runs for $n-1$ times, the time complexity is $O(n^2)$.

③ To find:. fbt(n,h) where n is the number of vertices and h is the height.

Following is the algorithm for the given problem.

```
1.  fbt(n,h):
2.      dp = [[0]] ----> all 0's matrix of size n×n
3.      from n=1 to n:
4.          from h=1 to h:
5.              result=0
6.              from left=1 to n by 2:
7.                  right = n-1-left
8.                  from i=0 to h-1
9.                      result += dp[right][i] x dp[left][h-1]
10.                 from i=0 to h-1
11.                     result += dp[left][i] x dp[right][h-1]
12.             dp[n][h] = result
13.     return dp[n][h]
```

For the above sum we need to check the no: of nodes and height at both left sub tree and right sub tree

Initially we start with 1 node and 1 height, we keep adding nodes and store the number of trees and also leaf nodes for both right and left.

Time complexity analysis:-

Since, there are two for loops to iterate 'n' times and we iterate another two for loops, one for the right and one for the right, with the size h, the time complexity is n^2h^2.

∴ The time complexity is $O(n^2h^2)$.

④ Given an array of length of holes, need to find the maximum number of possible partitions such that the sum of the partitions should be in decreasing order.

Following is the algorithm for the given problem:-

```
1.  dp = [ 0 ] ----> list of size n x n
2.  Number_of_Partitions( A, index, prev_sum)
3.      if index == A.size
            return 0
4.
5.      if dp[index]
            return dp[index]
6.
7.      sum = 0, result = 0, temp
8.      from i = index to A.size :
            sum = sum + A[i]
9.
10.         if  sum <= prev_sum
11.             temp = 1 + Number_of_Partitions(A, i+1, sum)
12.             result = max(temp, result)
13.     dp[index] = result
14.     return result
```

In the above algorithm we keep iterating elements from one index to the end to find a sum greater than the previous. We also save the number of partitions in an array at every index and iterate every time.

Finally we get the total number of partitions.

The above algorithm is a recursive one which runs on the remaining array from one index.

Since at each index the size of the array decreases and the recursive function is called n times, the recursive function runs in a polynomial of k.

(7) (a) Editing distance problem :-

Following is the algorithm for the problem :-

```
1.  Editing Distance Problem (x, y)

2.      m = length(x)
3.      n = length(y)
4.      dp = [[0 for x in range(n+1)] for x in range(m+1)]
5.      for i = 0 to m
6.          for j = 0 to n
7.              if i == 0:
                    dp[i][j] = j
8.
9.              else if j == 0
                    dp[i][j] = i
10.
11.             else if x[i-1] == y[j-1]:
                    dp[i][j] = dp[i-1][j-1]
12.
                else:
13.                 dp[i][j] = 1 + min(dp[i][j-1],
14.a                                   dp[i-1][j],
14.b                                   dp[i-1][j-1])
14.c
15      return dp[m][n]
```

The following is a basic idea on how to solve the problem:-

1. If last characters are the same, we can ignore those characters and recur on the remaining characters as we don't need to do any of the 3 operations.

2. If the last characters are not same, we need to do one of the 3 operations to make the 3 strings same. Do the following for different operations on X:-

   1. Insert: Recursion on m and n-1.
   2. Remove: Recur for m-1 and n.
   3. Replace: Recur for m-1 and n-1.

   We compute the minimum of the above 3 and go to the next operation.

Time complexity of the above solution is $O(mn)$ as there are 2 for loops of sizes m and n.