1> **A>**

Let A[n][m] -> Let A be an array of size m+1 and n, where each rows represents of the profit obtained by assigning j programmers to project i, column represents the number of programmers assigned to project

n -> number of projects
m -> number of programmers

Algorithm Steps :

Step1 : Calculate the marginal benefit array for each project that is obtained by assigning one more programmer to the existing programmer set. Use an 2D array B of size m * n to hold the values. This is got by subtracting each column value by the previous column value i.e.
B[i][j-1] = A[i][j] - A[i][j-1]

Step 2: Maintain an array "countPrgmr" of size n to keep track of count of programmers assigned to each project. Initialize this array to 0 initially. This implies that zero programmers are assigned to all projects.

Step 3: Now loop through each project i and find the project whose marginal profit obtained by an addition of a programmer to the existing set is maximum.

Step5: Add a programmer to this project by incrementing the value by 1 for the corresponding project index i in array "countPrgmr". Continue this process until we have assigned all m programmers to at-least one project.

Step 4: loop through the array which holds the number of programmers assigned to each project and compute the total profit "maxTtlProfit" made by assigning m programmers to n projects.

Step 5: Return "countPrgmr" and "maxTtlProfit" i.e. count of programmers assigned to each project and max profit obtained.

Pseudocode :
Given : A 2D array A which hold the details of the profit obtained assigning j programmers to each project i , an integer m is the number of programmers and n is the total number of pojects

Returns : An array of size n where each row represents the number of programmers assigned to the project i where 0 <= i < n and an integer which represents maximum total profit.

```
Maximize-Profit(A[][],int m, int n)
{
0    B[][]  -> array of n* m to hold marginal benefit values which is
                 computed   by calculating the difference
1    for i = 0 to n-1
2          for j = 1 to m
3          B[i][j-1] = A[i][j] - A[i][j-1]

4     countPrgmr[n]  array of size n which holds the value of number of
      programmers assigned to  each project.

     // initially one person is assigned to each project
5     for  k =  0 to n-1
6          countPrgmr [k] = 0

7    ttlProgmers = 0

8    while ttlProgmers <= m-1
9      maxMarginalProfit =0
10     maxIndx =-1
11     for i = 0 to n-1
12           if   B[countPrgmr[i]][i]  > maxMarginalProfit
13              maxIndx = i
14              maxMarginalProfit  = B[countPrgmr [i]][i]
15     countPrgmr [maxIndx]++
16      ttlProgmers++

17   maxTtlProfit = 0
18   for  i = 0 to n-1
19   {
        maxTtlProfit += A[i][countPrgmr[i]] * countPrgmr[i]
20    }
17   return  (maxTtlProfit, countPrgmr)

}
```

1> B> Proof of correctness: TO BE COMPLETED

Let C = { $C_{1}, C_{2}, C_{3}, C_{4}, \ldots\ldots C_{n}$ } be the optimal solution generated by the above algorithm where C_{i} represents the count of number of programmers assigned to each project i where $0 <= i < n$. Let T be the total profit obtained for optimal solution

In this algorithm we greedily choose to add programmer to the project when the marginal profit obtained is greater than the marginal profit obtained by adding the programmers to other projects.

Let x be countPrgmr array used to store the number of programmers assigned.

In this algorithm the optimal substructure is

$$P(n) = \sum_{i=0}^{n} \max_{k \in x[i]}(B[k][i]) + 1$$

Let's assume that we have another arbitrary solution O = { $O_{1}, O_{2}, O_{3}, O_{4}, \ldots\ldots O_{n}$ } which is better than the optimal solution obtained by the algorithm and Ok ! = Ck for some k>=0 and k <n. Let this Ok be obtained by adding last programmer to the project $P_{k\ 1}$instead of P_{k2}. Let 0<=k1 <n and 0<= k2<n be two projects. Let T` be the total profit obtained for this solution.

In the above algorithm, We add programmers to a project if the marginal benefit obtained by adding an programmer to this project P_{i} is greater than the marginal benefit obtained by adding a programmer to any other project P_{j} where i != j

Now since we have added the programmer to a project P_{k1} rather than the one which gives the optimal marginal benefit i.e. project P_{k2} , the marginal benefit $b_{k1} >= b_{k2}$ Where b_{k1} represents the marginal benefit for project k1 and b_{k2} represents the marginal benefit for project k2.
We can have $b_{k1} = b_{k2}$ as it results in the same total profit. But if $b_{k1} > b_{k2}$, it will result in a contradiction as it results in a solution whose total profit T` > T which is considered to be optimal. Therefore there is no optimal solution other than C which can be obtained when $b_{k1} > b_{k2}$

Space Complexity:

This algorithm involves a m*n matrix to store the marginal-benefit details and an additional array of size n is used to keep track of count of programmers assigned to each project P.

Therefore, total space = mn + n
⇨ **Space Complexity = O(mn)**

Time complexity :

Steps 1 to 3 involves two for loops which runs for m and n times. Run time = C1 * m * n

Steps 5 and 6 involves a for loop which runs for n times. Run time = C2 * n

Steps 6 to 8 involves a while loop which runs for (m-1) times when n =1 and a for loop which runs for a max of n times .
Run time for all steps between 12 to 15 = (m-1) n = (mn – n) * C3

Steps 9 and 10 are executed for a maximum of m * c4 times .

Steps 18 to 20 execute for a max of n times. = n * c5
Remaining lines run for some constant amount of time = C

Total run time = (C1 * mn) + (C2 * n) + ((mn – n) * C3) + m * c4 + n *c5 + C

Time Complexity T(n) = O(mn), By ignoring all lower order terms