Everytime we are struct all possible subsekt we've to try then just think of DP

Post Title

Burst Balloon to maximize coins

→ MCM → Subarray taken at a time & Partitioning it
→ Burst Balloon → Subsequence (Pick any from) Partition also

We have been given N balloons, each with a number of coins associated with it. On bursting a balloon i, the number of coins gained is equal to A[i-1]*A[i]*A[i+1]. Also, balloons i-1 and i+1 now become adjacent. Find the maximum possible profit earned after bursting all the balloons. Assume an extra 1 at each boundary.

Examples:

```
Input : 5, 10
Output : 60
Explanation - First Burst 5, Coins = 1*5*10
              Then burst 10, Coins+= 1*10*1
              Total = 60

Input : 1, 2, 3, 4, 5
Output : 110
```

A recursive solution is discussed here. We can solve this problem using dynamic programming. First, consider a sub-array from indices Left to Right(inclusive).
If we assume the balloon at index Last to be the last balloon to be burst in this sub-array, we would say the coined gained to be-A[left-1]*A[last]*A[right+1]. dp[current] = dcurrent + last
Also, the total Coin Gained would be this value, plus dp[left][last - 1] + dp[last + 1][right], where dp[i][j] means maximum coin gained for sub-array with indices i, j.
Therefore, for each value of Left and Right, we need find and choose a value of Last with maximum coin gained, and update the dp array.
Our Answer is the value at dp[1][N].

C++ | Java | Python3 | C# | Javascript

```
// Java program to illustrate
// Burst balloon problem
import java.util.Arrays;

class GFG{

public static int getMax(int[] A, int N)
{

    // Add Bordering Balloons
    int[] B = new int[N + 2];
    B[0] = B[N + 1] = 1;

    for(int i = 1; i <= N; i++)
```

See in video for Approach in Tabulation

```
        B[i] = A[i - 1];

    // Declaring DP array
    int[][] dp = new int[N + 2][N + 2];

    for(int length = 1;
            length < N + 1; length++)
    {
        for(int left = 1;
                left < N - length + 2; left++)
        {
            int right = left + length -1;

            // For a sub-array from indices
            // left, right. This innermost
            // loop finds the last balloon burst
            for(int last = left;
                    last < right + 1; last++)
            {
                dp[left][right] = Math.max(
                            dp[left][right],
                            dp[left][last - 1] +
                            B[left - 1] * B[last] *
                            B[right + 1] +
                            dp[last + 1][right]);

            }
        }
    }
    return dp[1][N];
}

// Driver code
public static void main(String args[])
{
    int[] A = { 1, 2, 3, 4, 5 };

    // Size of the array
    int N = A.length;

    // Calling function
    System.out.println(getMax(A, N));
}
}

// This code is contributed by dadi madhav
```

So all understood now
↓
with strivers yt & code here

→ Last is like index 'k'
Last balloon to be burst, which partition into sub-array.

left subarray of k(last)

when (last=k) is bursted in last we will have what is left of left i.e left-1 & right of right i.e. right+1

Right subarray of k(last)

e.g.,

1 [5 1 3 8] 1

Left ← length → Right

last (let 1 be last)

→ last

last

→ last

→ last

Last can be one of any from this i to j or left to right range

Output:

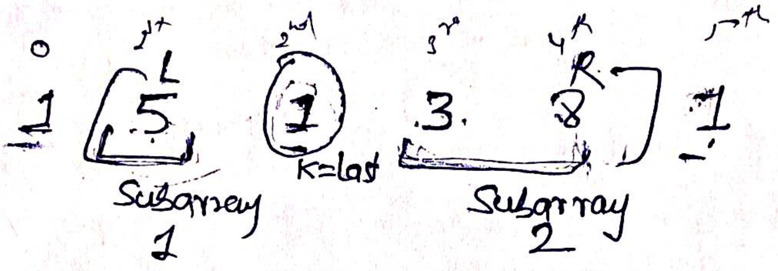
```
110
```

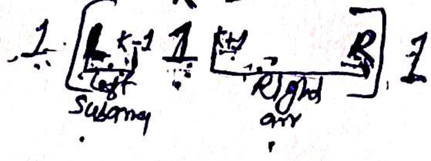
So it is acting like we had in MCM.

Partition DP only like MCM

[5 1 3 8]

How to make approach to bring this question to solve by getting subproblem.





DP[L][K-1] + $\boxed{1 \times 1 \times 1 = 1}$ ⟹ Array (last=K) * array (Left-1)
DP[K+1][R]
* array (Right+1)

⟹. Max = { DP[L][K-1] + DP[K+1][R] + arr(K) * arr (Left-1)
* arr(Right+1)