Question: https://leetcode.com/problems/burst-balloons/

Intuition-  
1) Clearly the brute force solution would be to try all possible combinations of popping the balloons and so First we have n choices to select 1 to burst and then in the next step n-1 choices and in the end we are left with the only balloon ..So clearly it's a n! solution which will work for n<=10  
2) So clearly we have to find a way to break this problem into smaller parts. A natural approach would be to separate the array on the basis of what if we pop a particular balloon first and then separate the array into 2 parts. SO if i have 3,1,4,8,9 and i guess that a balloon 4 should be popped first ,would it make the 2 parts [3,1] &[8,9] results independent of each other.I mean can we forget while calculating [3,1] that [8,9] exists. Clearly Not,  
Why?  
because even if we knew in [3,1] the correct order we have to pop these balloons we still dont know the exact right neighbor of these elements ,rigght neighbor could be 8 or 9 depending on which one is popped earlier.  
So what if ,we could find a definition of dp that could help us make these 2 parts independent of each other.

1. So instead of picking or guessing which balloon to pop first we guess which balloon is popped last. So it means if [1,2,3,4,5] if in this array we knew that element 3 was the last one to be popped and then we would know that subparts[1,2] would always have right border as 3 because it's the last one to be popped and similarly the right part[4,5] would always know it's left border

Now onto the solution->  
we define ****dp[left][right]to be maximum value we can get on bursting thee balloons in the subarray nums**[left,right](https://leetcode.com/problems/burst-balloons/discuss/1524408/inclusive)**considering the fact that other balloons are still not burst.****

Code:  
class Solution {

public int maxCoins(int[] iNums) {

int[] nums = new int[iNums.length + 2];

int n = 1;

for (int x : iNums) if (x > 0) nums[n++] = x;

nums[0] = nums[n++] = 1;

int[][] dp = new int[n][n];

for (int k = 2; k < n; ++k)

for (int left = 0; left < n - k; ++left) {

int right = left + k;

for (int i = left + 1; i < right; ++i)

dp[left][right] = Math.max(dp[left][right],

nums[left] \* nums[i] \* nums[right] + dp[left][i] + dp[i][right]);

}

return dp[0][n - 1];

}

}

Github Link :<https://lnkd.in/ecwtJeaz>