**STEPS TO CHANGE TOP DOWN CODE INTO BOTTOM UP CODE-**

Step - 1-> Write the base case -> If it is generic then initialise the dp matrix accrodingly or if it is not that way like here, it is dealt

separately

Step -2 -> Write the changing variables of recursive code as nested loop in bottom up approach and revert the way values of the variables

change like in top down apporach i changes from 1 to n-2 so here it will go from n-2 to 1 and same for others.

Step - 3 -> Copy the recursive code inside the loop and change the recursive call with dp values.

Note-> We may need to increase size of dp in bottom up for corner cases.

We may also need to change the default value in the dp matrix in the bottom up apporoach

Example –

//Problem link - https://leetcode.com/problems/burst-balloons/

//Video link - https://www.youtube.com/watch?v=Yz4LlDSlkns&t=1700s

//Time complexity - O(n^3)

class Solution {

public:

int helper(vector<int> &nums,vector<vector<int>> &dp,int i,int j)

{

if(i>j) return 0;//there are no balloons to consider

if(dp[i][j] != -1) return dp[i][j];

dp[i][j] = INT\_MIN;

for(int k=i;k<=j;k++)//if kth balloon is left in the last to burst

{

dp[i][j] = max(dp[i][j],nums[i-1]\*nums[k]\*nums[j+1]+helper(nums,dp,i,k-1)+helper(nums,dp,k+1,j));//to write the indices just

//think of base case.For first time,i gets 1 and j gets (n-2)

//so in this range if kth balloon is bursted in last, it's

//reward will be nums[i-1]\*nums[k]\*nums[j+1] and not

//nums[i]\*nums[k]\*nums[j] because balloons at ith and jth

//index would have already been bursted before balloon at k

//as it is bursting last and hence remaining balloons on its

//immediate left and right will be at index (i-1) and (j+1)

}

return dp[i][j];

}

int maxCoins(vector<int>& nums)

{

nums.push\_back(1);

nums.insert(nums.begin(),1);

int n = nums.size();

vector<vector<int>> dp(n,vector<int>(n,-1));

return helper(nums,dp,1,n-2);

}

};

**Changed code to bottom up**

class Solution {

public:

int maxCoins(vector<int>& nums) {

nums.push\_back(1);

nums.insert(nums.begin(),1);

int n = nums.size();

vector<vector<int>> dp(n+1,vector<int>(n+1,0));

for(int i=n-2;i>=1;i--)

{

for(int j=1;j<=n-2;j++)

{

if(i>j) continue;//This base condition was not generic as we couldn't initialise few values in dp with it hence it is dealt

//in this way

dp[i][j] = INT\_MIN;

for(int k=i;k<=j;k++)

dp[i][j] = max(dp[i][j],nums[i-1]\*nums[k]\*nums[j+1]+dp[i][k-1]+dp[k+1][j]);

}

}

return dp[1][n-2];

}

};