**Optimal binary search tree**

**Medium**Accuracy: 60.65% Submissions: 902 Points: 4

Given a sorted array **keys[0.. n-1]** of search keys and an array **freq[0.. n-1]** of frequency counts, where freq[i] is the number of searches to keys[i]. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.  
Let us first define the cost of a BST. The cost of a BST node is level of that node multiplied by its frequency. Level of root is 1.

**Example 1:**

**Input:**

n = 2

keys = {10, 12}

freq = {34, 50}

**Output:** 118

**Explaination:**

There can be following two possible BSTs

10 12

\ /

12 10

*The cost of tree I is 34\*1 + 50\*2 = 134*

*The cost of tree II is 50\*1 + 34\*2 = 118*

***Example 2:***

***Input:***

*N = 3*

*keys = {10, 12, 20}*

*freq = {34, 8, 50}*

***Output:*** *142*

***Explaination:*** *There can be many possible BSTs*

*20*

*/*

*10*

*\*

*12*

*Among all possible BSTs,*

*cost of this BST is minimum.*

*Cost of this BST is 1\*50 + 2\*34 + 3\*8 = 142*

***Your Task:*** *You don't need to read input or print anything. Your task is to complete the function****optimalSearchTree()****which takes the array****keys[], freq[]****and their size****n****as input parameters and returns the total cost of all the searches is as small as possible.*

***Expected Time Complexity:****O(n3)****Expected Auxiliary Space:****O(n2)*

***Constraints:*** *1 ≤ N ≤ 100*

class Solution{

    public:

    int dp[101][101][101];

    int solve(int l, int r, int level, int keys[], int freq[]) {

        if (l==r) return freq[l]\*level;

        if (l>r) return 0;

        if (dp[l][r][level]!=-1) return dp[l][r][level];

        int ans=INT\_MAX;

        for (int i=l; i<=r; i++) {

            int a=solve(l, i-1, level+1, keys, freq);

            int b=solve(i+1, r, level+1, keys, freq);

            int value=a+b+(freq[i]\*level);

            ans=min(ans, value);

        }

        return dp[l][r][level]=ans;

    }

    int optimalSearchTree(int keys[], int freq[], int n) {

        // code here

        memset(dp, -1, sizeof(dp));

        return solve(0, n-1, 1, keys, freq);

    }

};