

Name: Pranjal Singh
Sec: FL_IOT-601/ A

UID: 22BCS13041
Sub: AP Lab -II

Climbing Stairs

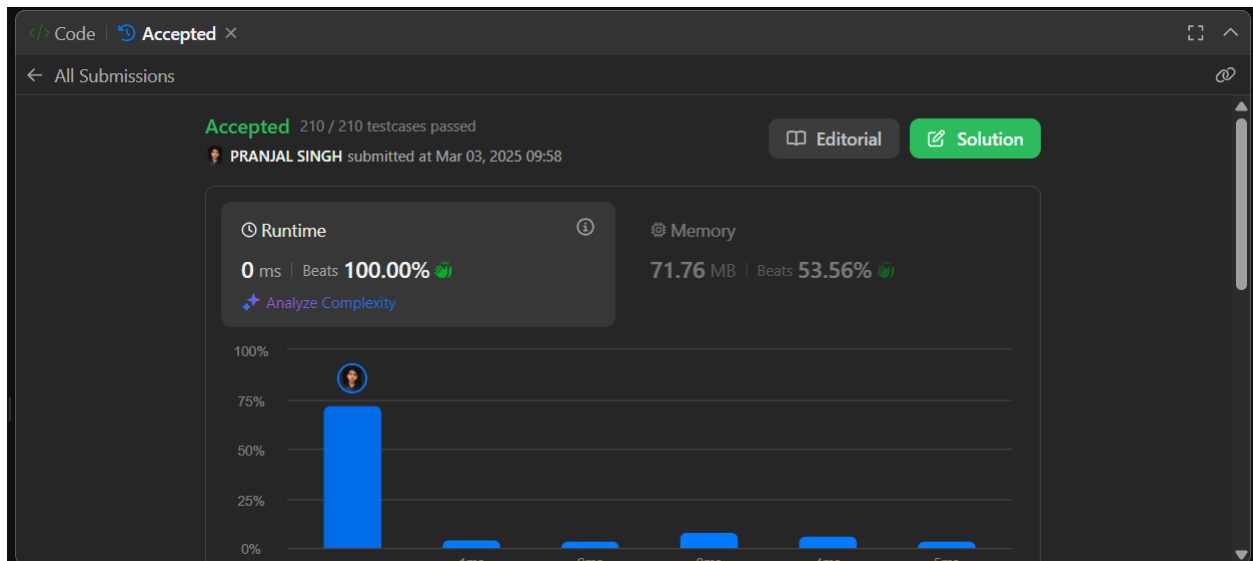
```
class Solution {  
public:  
    int climbStairs(int n) {  
        if (n <= 2) return n;  
  
        vector<int> dp(n + 1, 0);  
        dp[1] = 1;  
        dp[2] = 2;  
  
        for (int i = 3; i <= n; i++) {  
            dp[i] = dp[i - 1] + dp[i - 2];  
        }  
  
        return dp[n];  
    }  
};
```

The screenshot displays a coding platform interface for the 'Climbing Stairs' problem. The left sidebar shows submission details: 'Accepted 45 / 45 testcases passed', '22bcs13041 submitted at Apr 05, 2025 12:50', 'Runtime: 0 ms | Beats 100.00%', and 'Memory: 8.48 MB | Beats 48.50%'. A bar chart indicates the user's performance is top 1%. The main area shows the C++ code for the solution, which uses a dynamic programming array 'dp' to calculate the number of ways to climb stairs. The bottom panel shows the 'Testcase' section with 'Case 1' selected, where 'n' is set to 2.

```
1 class Solution {  
2 public:  
3     int climbStairs(int n) {  
4         if (n <= 2) return n;  
5  
6         vector<int> dp(n + 1, 0);  
7         dp[1] = 1;  
8         dp[2] = 2;  
9  
10        for (int i = 3; i <= n; i++) {  
11            dp[i] = dp[i - 1] + dp[i - 2];  
12        }  
13  
14        return dp[n];  
15    }  
16 };
```

Maximum Subarray

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        // Kadane's Algo...
        int maximumsum=INT_MIN;
        int currentsum=0;
        for(int i =0;i<nums.size();i++){
            currentsum=currentsum+nums[i];
            maximumsum=max(currentsum,maximumsum);
            if(currentsum<0){
                currentsum=0;
            }
        }
        return maximumsum;
    }
};
```



House Robber

```
#include <vector>
```

```
#include <algorithm>
```

```
using namespace std;
```

```
class Solution {
```

```
private:
```

```
    int rec(int i, vector<int>& nums, vector<int>& dp) {
```

```
        if (i >= nums.size()) return 0;
```

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

```
        int take = nums[i] + rec(i + 2, nums, dp);
```

```
        int dont = rec(i + 1, nums, dp);
```

```
        return dp[i] = max(take, dont);
```

```
    }
```

```
public:
```

```
    int rob(vector<int>& nums) {
```

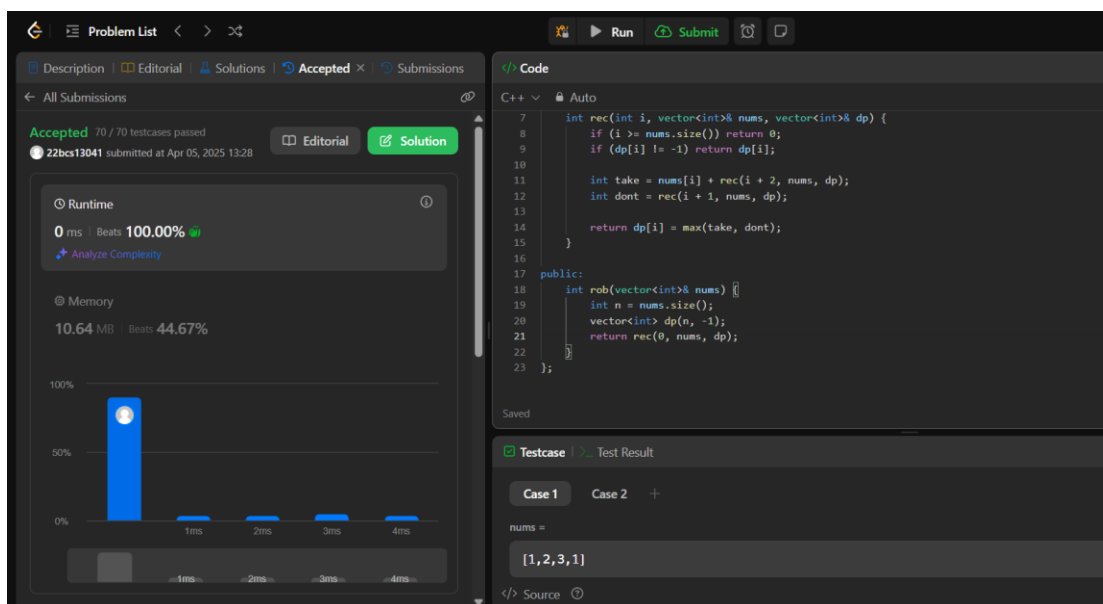
```
        int n = nums.size();
```

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

```
        return rec(0, nums, dp);
```

```
    }
```

```
};
```



Jump Game

```
class Solution
{
public:
    bool canJump(vector<int> &nums){
        if(nums.size() == 1) return true;
        int prevGreatestNum = nums[0];
        for (int i = 0; i < nums.size() - 1; i++){
            if (nums[i] != 0){
                if (nums[i] + i + 1 >= nums.size()) {
                    return true;
                }
            }
            else {
                if (prevGreatestNum <= 0) {
                    break;
                }
            }
            prevGreatestNum = max(prevGreatestNum, nums[i]);
            prevGreatestNum--;
        }
        return false;
    }
};
```

