Experiment 7

Student Name: PRATEEK **UID: 22BCS13864**

Branch: UIE CSE 3rd Year **Section/Group: 22BCS_KPIT-901-'B'**

Semester: 6th **Date of Performance: 27th March 2025**

Subject Name: Advanced Programming – II **Subject Code: 22CSP-351**

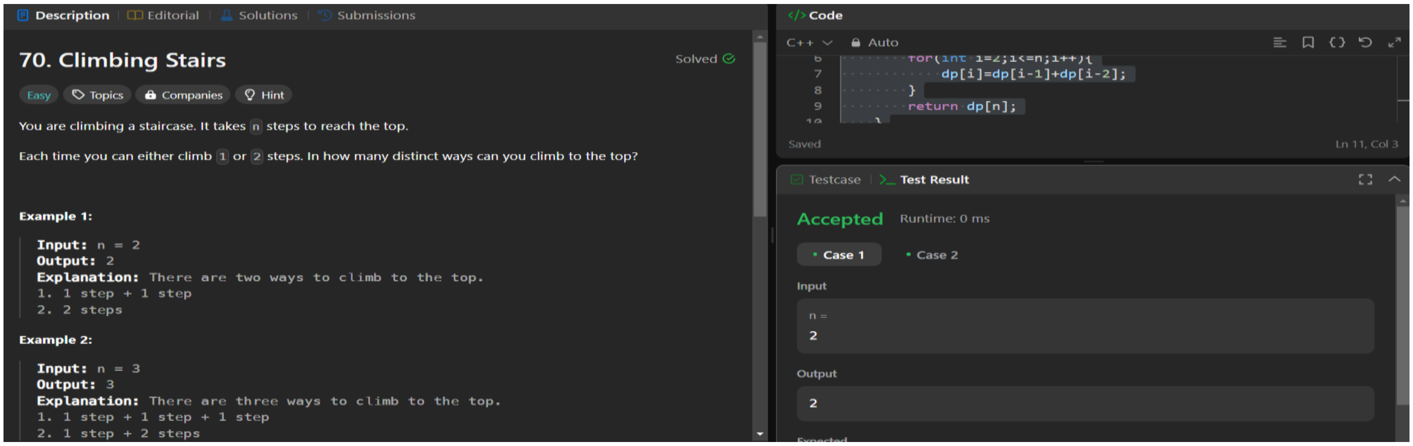
1. Aim:

You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

2. Implementation/Code:

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

3. Output:



70. Climbing Stairs

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

Example 2:

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
```

```
        for(int i=2;i<=n;i++){
            dp[i]=dp[i-1]+dp[i-2];
        }
        return dp[n];
    }
```

Saved Ln 11, Col 3

Accepted Runtime: 0 ms

Input

n =
2

Output

2

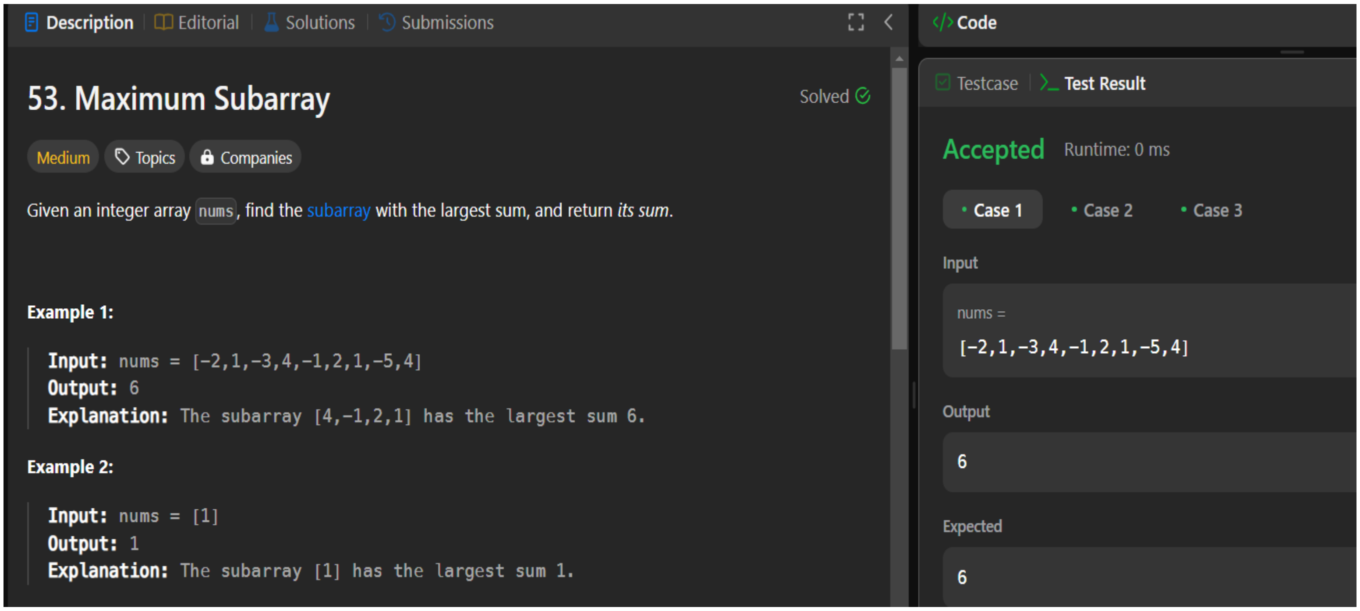
Expected

QUES:2

1. **Aim:** Given an integer array nums, find the subarray with the largest sum, and return *its sum*.
2. **Implementation/Code:**

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int sum=0;
        int maxi=nums[0];
        for(int i=0;i<nums.size();i++){
            sum+=nums[i];
            maxi=max(maxi,sum);
            if(sum<0)sum=0;
        }
        return maxi;
    }
};
```

3. **Output:**



53. Maximum Subarray

Given an integer array `nums`, find the subarray with the largest sum, and return *its sum*.

Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.
```

Accepted Runtime: 0 ms

Input

nums =
[-2,1,-3,4,-1,2,1,-5,4]

Output

6

Expected

6

QUESTION:3

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array nums representing the amount of money of each house, return *the maximum amount of money you can rob tonight* ***without alerting the police***.

CODE:

```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n=nums.size();
        if(n==1) return nums[0];
        if(n==2) return (nums[1]>nums[0])?nums[1]:nums[0];
        int m_money=0;
        vector<int>max_money(n);
        max_money[0]=nums[0];
        max_money[1]=(nums[1]>nums[0])?nums[1]:nums[0];
        for(int i=2;i<nums.size();i++){
            max_money[i]=max(max_money[i-1],nums[i]+max_money[i-2]);
        }
        return max_money[n-1];
    }
};
```

QUESTION:4

You are given an integer array nums. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return true *if you can reach the last index, or* false *otherwise*.

CODE:

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

QUESTION:5

There is a robot on an m x n grid. The robot is initially located at the **top-left corner** (i.e., grid[0][0]). The robot tries to move to the **bottom-right corner** (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.

Given the two integers m and n, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to 2 * 10^9.

CODE:

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        vector<int> cur(n, 1);
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                cur[j] += cur[j - 1];
            }
        }
        return cur[n - 1];
    }
};
```