



WORKSHEET 7

Student Name: Shubham

UID:22BCS15701

Branch: CSE

Section/Group: NTPP 603/B

Semester: 06

Date of Performance: 13/03/2025

Subject Name: AP Lab II

Subject Code: 22CSP-351

1. Aim:

- a) Climbing Stairs
- b) House Robber
- c) Maximum Subarray

2. Source Code:

a.

```
class Solution {
public:
    int climbStairs(int n) {
        // dp[i] := the number of ways to climb to the i-th stair
        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];
    }
};
```

b.

```
class Solution {
public:
    int rob(vector<int>& nums) {
        if (nums.empty())
```

```
        return 0;
    if (nums.size() == 1)
        return nums[0];

    // dp[i] := the maximum money of robbing nums[0..i]
    vector<int> dp(nums.size());
    dp[0] = nums[0];
    dp[1] = max(nums[0], nums[1]);

    for (int i = 2; i < nums.size(); ++i)
        dp[i] = max(dp[i - 1], dp[i - 2] + nums[i]);

    return dp.back();
}
};
```

C.

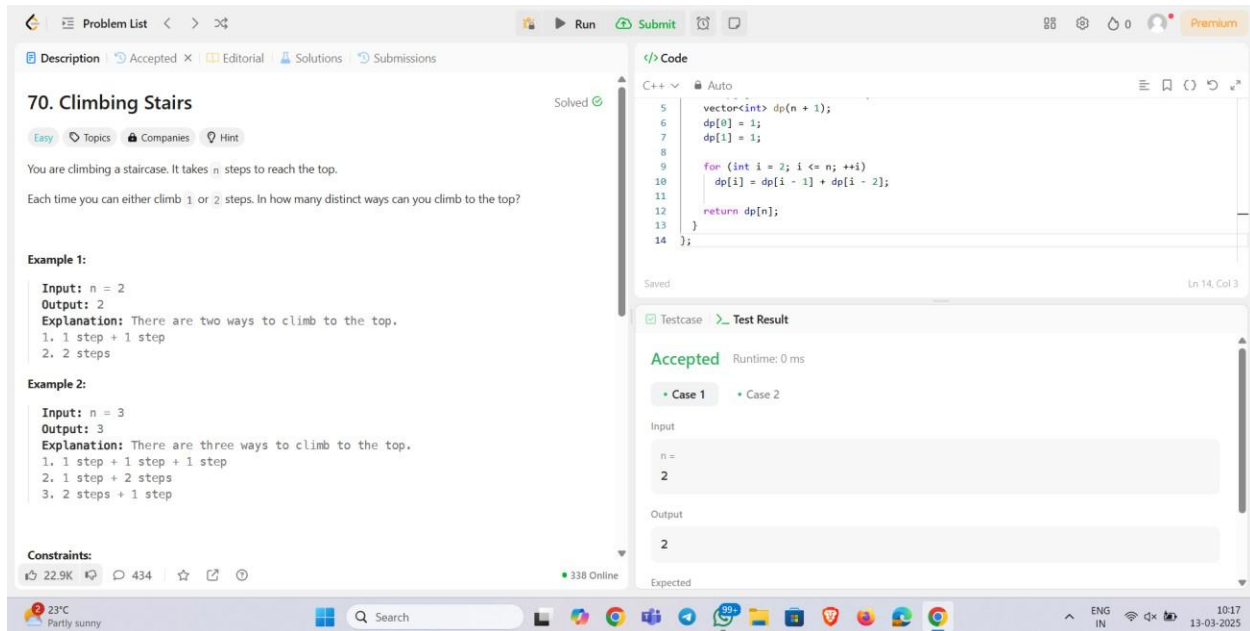
```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        // dp[i] := the maximum sum subarray ending in i
        vector<int> dp(nums.size());

        dp[0] = nums[0];
        for (int i = 1; i < nums.size(); ++i)
            dp[i] = max(nums[i], dp[i - 1] + nums[i]);

        return ranges::max(dp);
    }
};
```

Screenshot of Outputs:

a.



70. Climbing Stairs Solved

Easy Topics Companies Hint

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
3. 2 steps + 1 step

Constraints:
22.9K 434 338 Online

```

C++
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];

```

Accepted Runtime: 0 ms

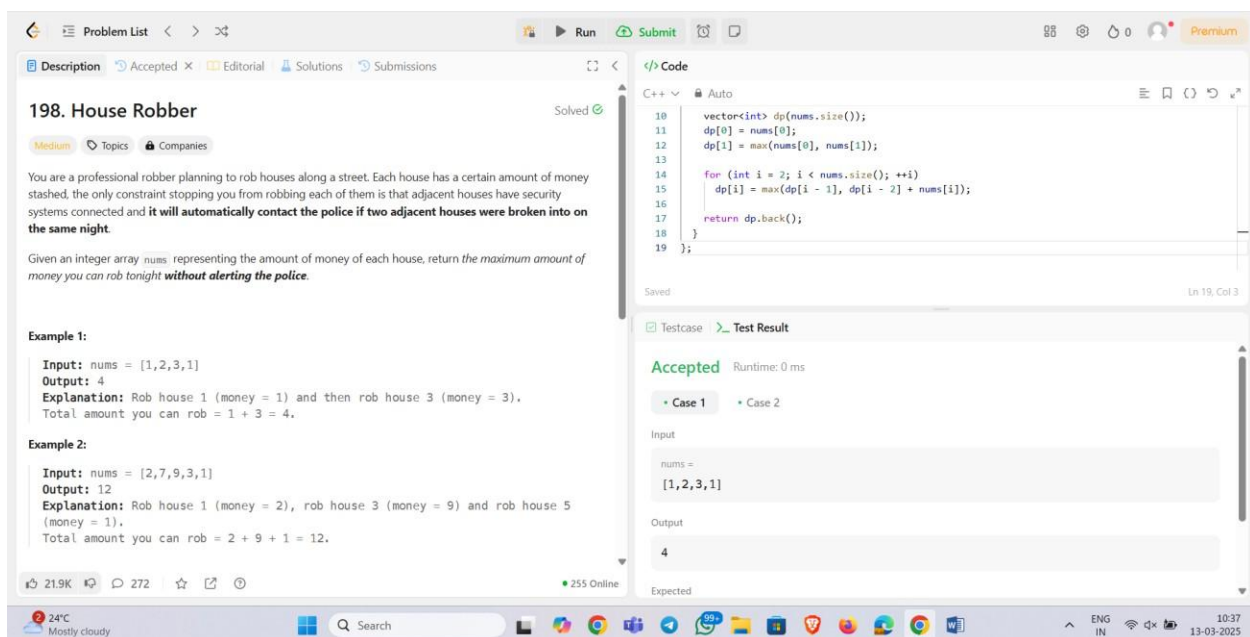
Case 1 Case 2

Input: $n = 2$

Output: 2

Expected

b.



198. House Robber Solved

Medium Topics Companies

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.

Example 1:
Input: `nums = [1,2,3,1]`
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3). Total amount you can rob = 1 + 3 = 4.

Example 2:
Input: `nums = [2,7,9,3,1]`
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1). Total amount you can rob = 2 + 9 + 1 = 12.

21.9K 272 255 Online

```

C++
vector<int> dp(nums.size());
dp[0] = nums[0];
dp[1] = max(nums[0], nums[1]);
for (int i = 2; i < nums.size(); ++i)
    dp[i] = max(dp[i - 1], dp[i - 2] + nums[i]);
return dp.back();

```

Accepted Runtime: 0 ms

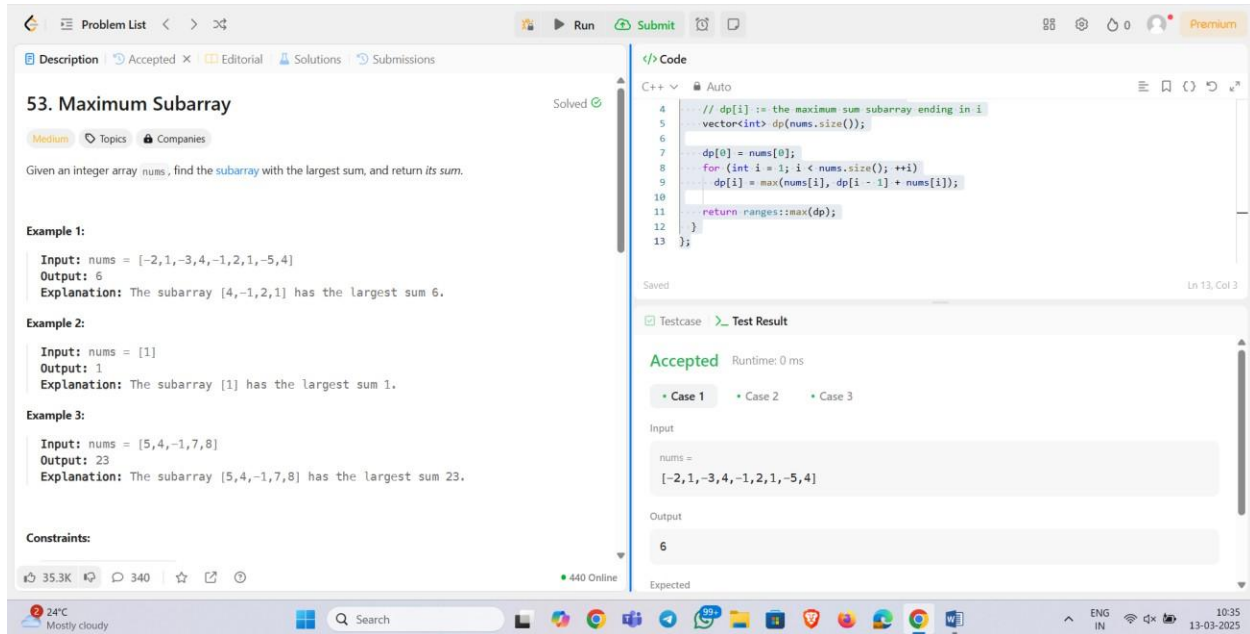
Case 1 Case 2

Input: `nums = [1,2,3,1]`

Output: 4

Expected

C.



The screenshot displays a coding problem titled "53. Maximum Subarray" on a platform. The problem description states: "Given an integer array `nums`, find the subarray with the largest sum, and return its sum." It includes three examples: Example 1 with input `nums = [-2,1,-3,4,-1,2,1,-5,4]` and output 6; Example 2 with input `nums = [1]` and output 1; and Example 3 with input `nums = [5,4,-1,7,8]` and output 23. The constraints are also listed. The solution is written in C++ using a dynamic programming approach. The code initializes a `dp` array of size `nums.size()`, sets `dp[0] = nums[0]`, and then iterates from `i = 1` to `nums.size() - 1`, updating `dp[i] = max(nums[i], dp[i - 1] + nums[i])`. Finally, it returns `ranges::max(dp)`. The test result shows "Accepted" with a runtime of 0 ms for Case 1, using the same input as Example 1, resulting in an output of 6.

```
4 // dp[i] := the maximum sum subarray ending in i
5 vector<int> dp(nums.size());
6
7 dp[0] = nums[0];
8 for (int i = 1; i < nums.size(); ++i)
9     dp[i] = max(nums[i], dp[i - 1] + nums[i]);
10
11 return ranges::max(dp);
12 }
13
```

3. Learning Outcomes

- (i) Learned about Dynamic Programming.
- (ii) Learned about top down approach.
- (iii) Learned about bottom up approach.