**Assignment 7 (Advance Programming)**

[**70. Climbing Stairs**](https://leetcode.com/problems/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

**Solution:**

class Solution {

public:

    int climbStairs(int n) {

           if (n <= 2) return n;

        int first = 1, second = 2;

        for (int i = 3; i <= n; i++) {

            int third = first + second;

            first = second;

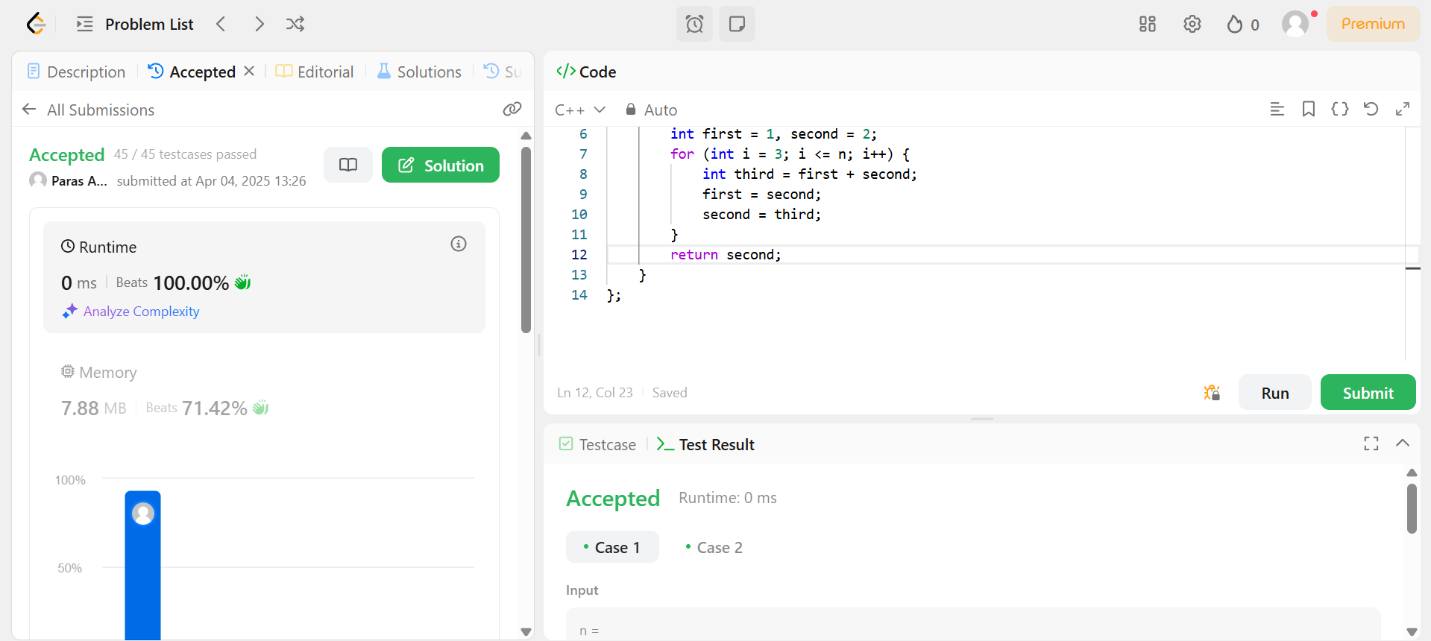
            second = third;

        }

        return second;

    }

};



**55. Jump Game**

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.

Example 1:

Input: nums = [2,3,1,1,4]

Output: true

Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

**Solution:**

class Solution {

public:

    bool canJump(vector<int>& nums) {

        int farthest = 0;

        int n = nums.size();

        for (int i = 0; i < n; i++) {

            if (i > farthest) return false;

            farthest = max(farthest, i + nums[i]);

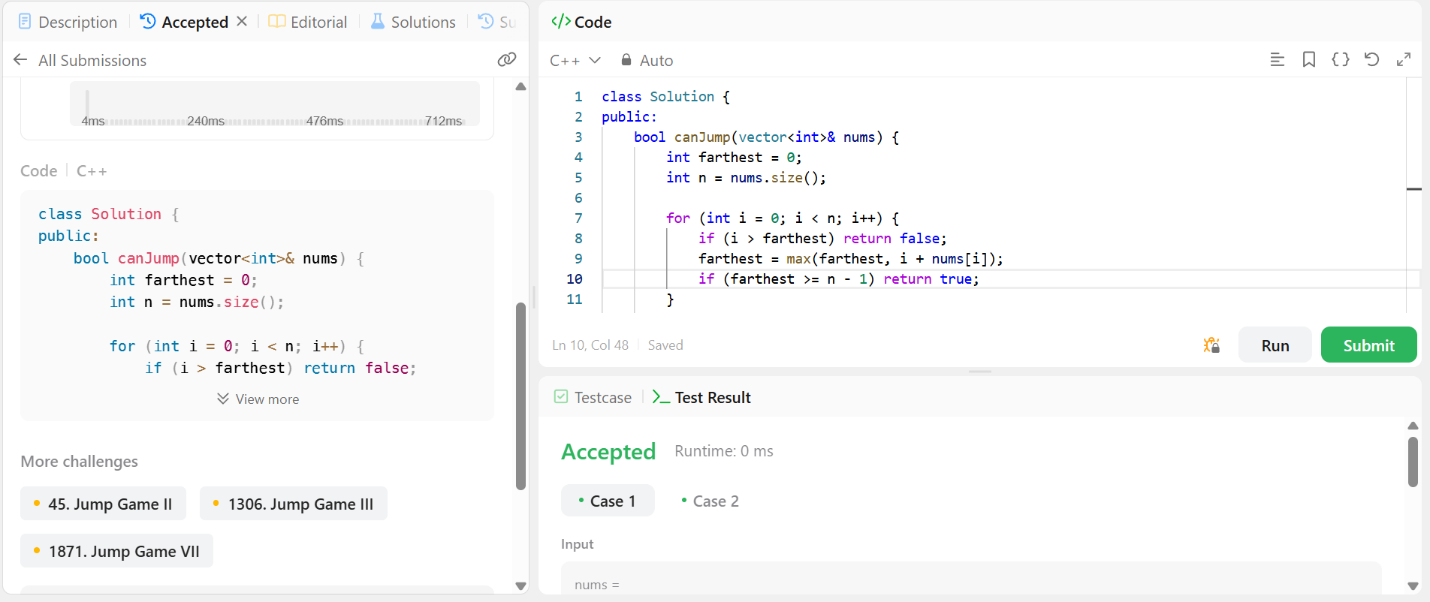
            if (farthest >= n - 1) return true;

        }

        return false;

    }

};



**152. Maximum Product Subarray**

Given an integer array nums, find a subarray that has the largest product, and return the product.

The test cases are generated so that the answer will fit in a 32-bit integer.

Example 1:

Input: nums = [2,3,-2,4]

Output: 6

Explanation: [2,3] has the largest product 6.

Example 2:

Input: nums = [-2,0,-1]

Output: 0

Explanation: The result cannot be 2, because [-2,-1] is not a subarray

**Solution:**

class Solution {

public:

    int maxProduct(vector<int>& nums) {

         int n = nums.size();

        int maxProd = nums[0];

        int minProd = nums[0];

        int result = nums[0];

        for (int i = 1; i < n; i++) {

            if (nums[i] < 0) swap(maxProd, minProd);

            maxProd = max(nums[i], maxProd \* nums[i]);

            minProd = min(nums[i], minProd \* nums[i]);

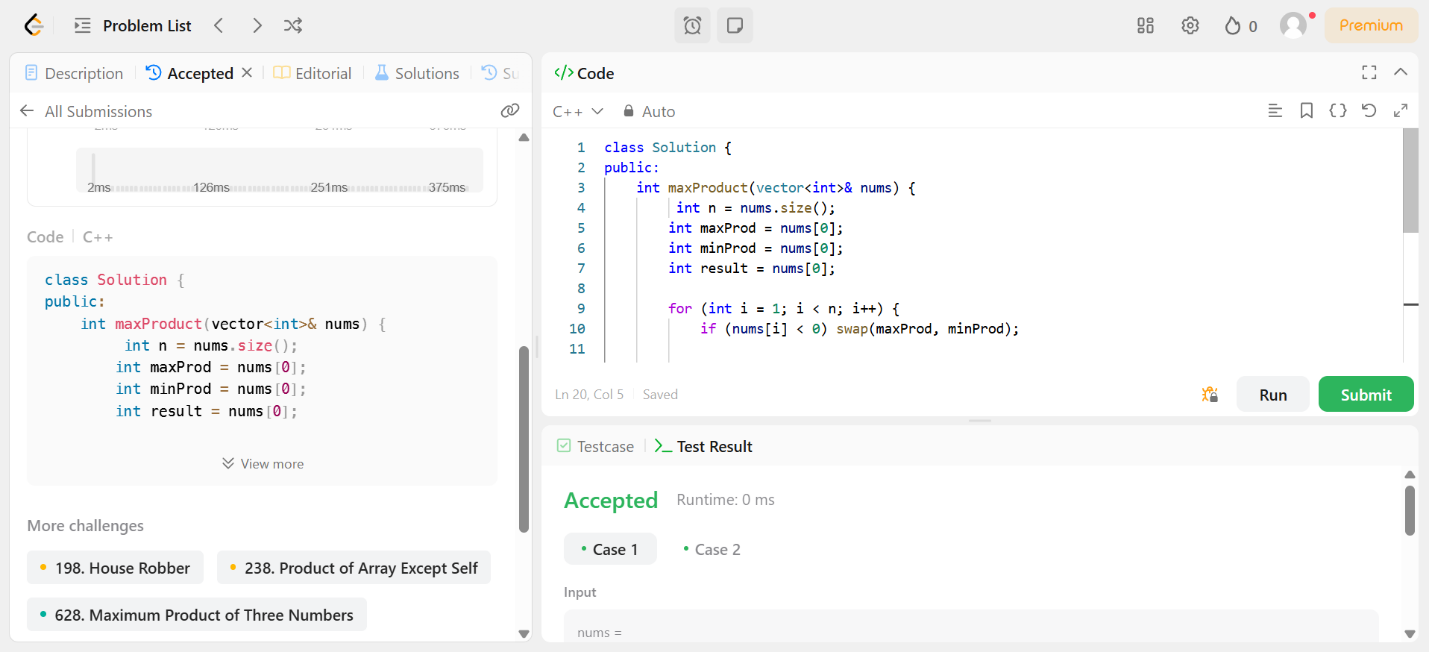
            result = max(result, maxProd);

        }

        return result;

    }

};



**279. Perfect Squares**

Given an integer n, return the least number of perfect square numbers that sum to n.

A perfect square is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not.

Example 1:

Input: n = 12

Output: 3

Explanation: 12 = 4 + 4 + 4.

Example 2:

Input: n = 13

Output: 2

Explanation: 13 = 4 + 9.

**Solution:**

class Solution {

public:

    int numSquares(int n) {

        vector<int> dp(n + 1, INT\_MAX);

        dp[0] = 0;

        for (int i = 1; i <= n; i++) {

            for (int j = 1; j \* j <= i; j++) {

                dp[i] = min(dp[i], dp[i - j \* j] + 1);

            }

        }

        return dp[n];

    }

};

