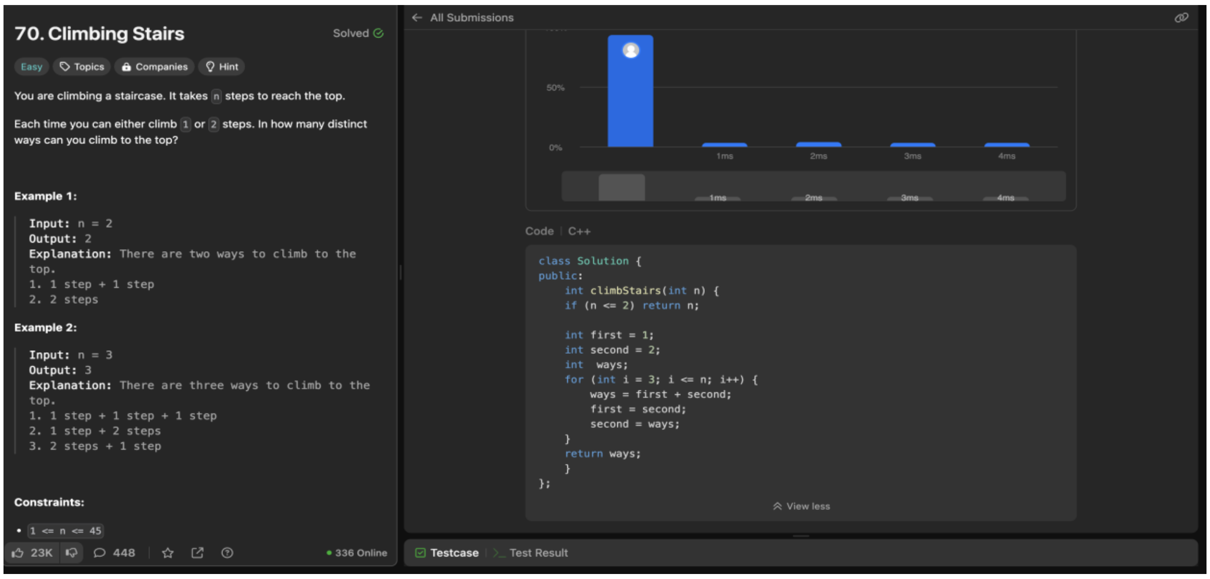
Name: Harmandeep Singh.   UID:22BCS14975

Ap Assignment 7

1. Staircase Problem (Climbing Stairs)

```
class Solution {
public:
    int climbStairs(int n) {
    if (n <= 2) return n;

    int first = 1;
    int second = 2;
    int  ways;
    for (int i = 3; i <= n; i++) {
        ways = first + second;
        first = second;
        second = ways;
    }
    return ways;
    }
};
```


70. Climbing Stairs

Solved

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:

- `1 <= n <= 45`

```
class Solution {
public:
    int climbStairs(int n) {
    if (n <= 2) return n;

    int first = 1;
    int second = 2;
    int  ways;
    for (int i = 3; i <= n; i++) {
        ways = first + second;
        first = second;
        second = ways;
    }
    return ways;
    }
};
```

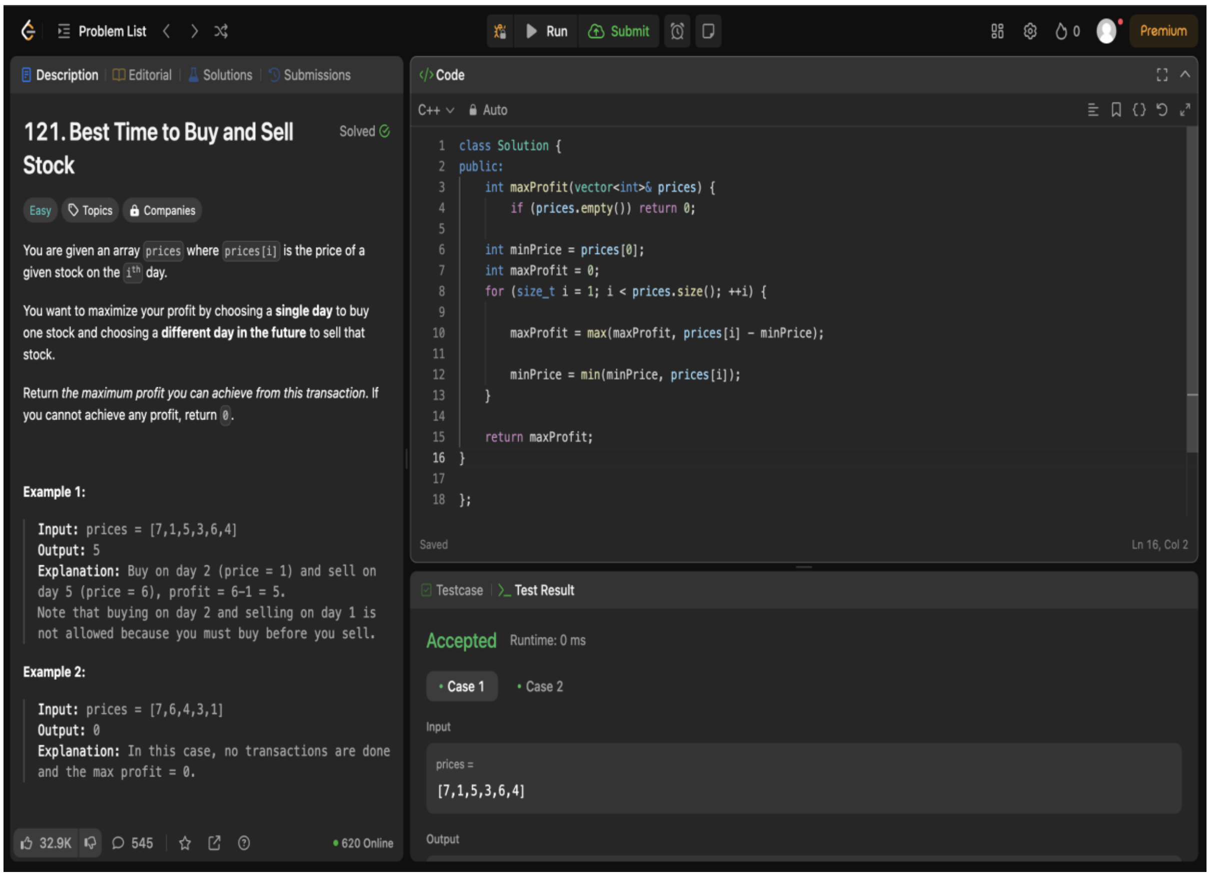
2. Best Time to Buy and Sell Stock

```
class Solution {
public:
```

```
int maxProfit(vector<int>& prices) {
if (prices.empty()) return 0;

int minPrice = prices[0];
int maxProfit = 0;
for (size_t i = 1; i < prices.size(); ++i) {
maxProfit = max(maxProfit, prices[i] - minPrice);
minPrice = min(minPrice, prices[i]);
}

return maxProfit;
}
};
```



121. Best Time to Buy and Sell Stock

Solved

Easy Topics Companies

You are given an array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on
day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is
not allowed because you must buy before you sell.
```

Example 2:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done
and the max profit = 0.
```

32.9K 545 620 Online

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        if (prices.empty()) return 0;

    int minPrice = prices[0];
    int maxProfit = 0;
    for (size_t i = 1; i < prices.size(); ++i) {

        maxProfit = max(maxProfit, prices[i] - minPrice);

        minPrice = min(minPrice, prices[i]);
    }

    return maxProfit;
}

};
```

Saved Ln 16, Col 2

Accepted Runtime: 0 ms

Case 1 Case 2

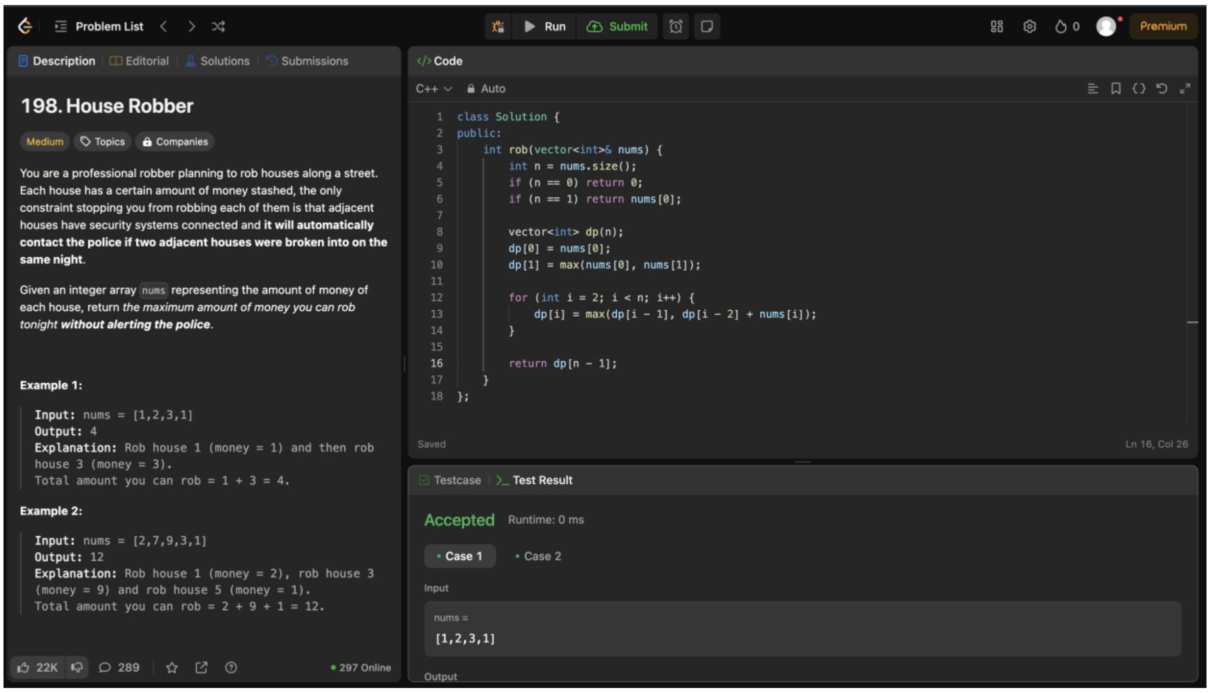
Input

prices =
[7,1,5,3,6,4]

Output

3. House Robber Problem

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



198. House Robber

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.
```

```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
        if (n == 0) return 0;
        if (n == 1) return nums[0];

        vector<int> dp(n);
        dp[0] = nums[0];
        dp[1] = max(nums[0], nums[1]);

        for (int i = 2; i < n; i++) {
            dp[i] = max(dp[i - 1], dp[i - 2] + nums[i]);
        }

        return dp[n - 1];
    }
};
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

nums =
[1,2,3,1]

Output

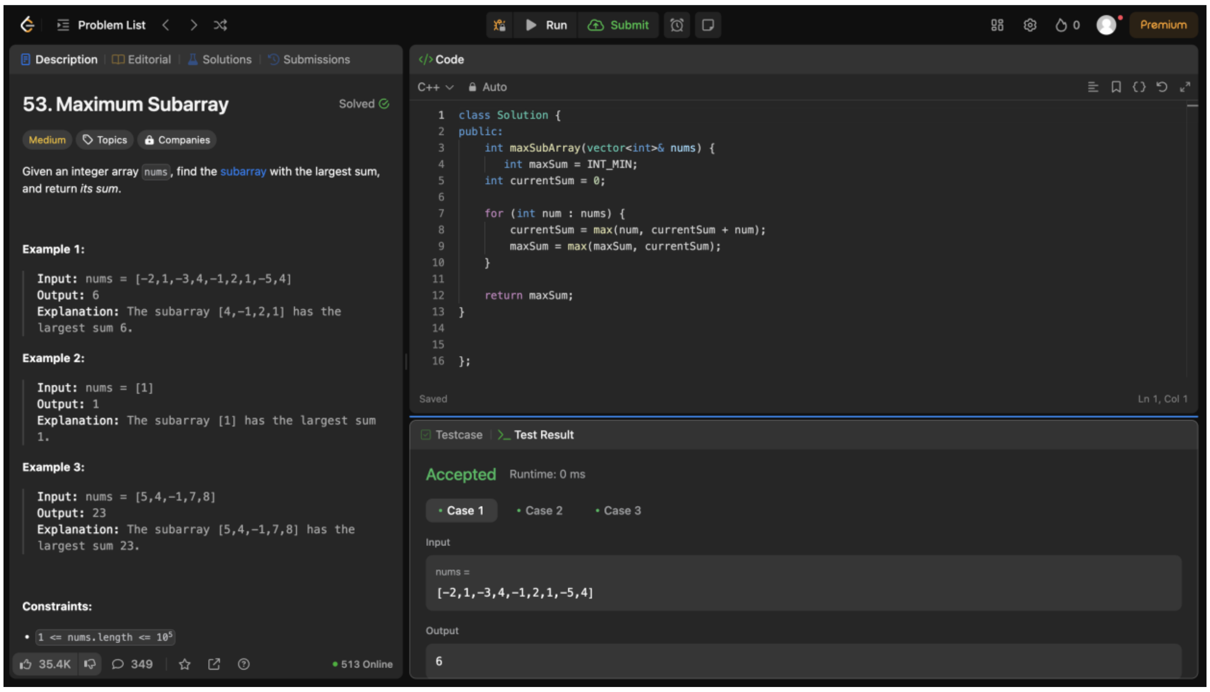
4.Maximum Subarray

```
class Solution {
public:
int maxSubArray(vector<int>& nums) {
int maxSum = INT_MIN;
int currentSum = 0;

for (int num : nums) {
currentSum = max(num, currentSum + num);
maxSum = max(maxSum, currentSum);
}

return maxSum;
}


};
```



53. Maximum Subarray

Solved

Medium · Topics · Companies

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.
```

Example 3:

```
Input: nums = [5,4,-1,7,8]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the
largest sum 23.
```

Constraints:

- `1 <= nums.length <=` 10^5

35.4K · 349 · 513 Online

C++ · Auto

```cpp
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
       int maxSum = INT_MIN;
    int currentSum = 0;

    for (int num : nums) {
        currentSum = max(num, currentSum + num);
        maxSum = max(maxSum, currentSum);
    }

    return maxSum;
}


};
```

Saved · Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

Input

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

Output

```
6
```