UNIVERSITY INSTITUTE OF ENGINEERING

Department of Computer Science & Engineering

(BE-CSE/IT-5th Sem)

Subject Name: AP LAB-II

Subject Code: 22CSP-351

Submitted to:

Faculty name

Anjali Thakur

Submitted by:

Name: Ayush Kumar Mayank

UID: 22BCS11193

Section: 22BCS-IOT-614

Group: B

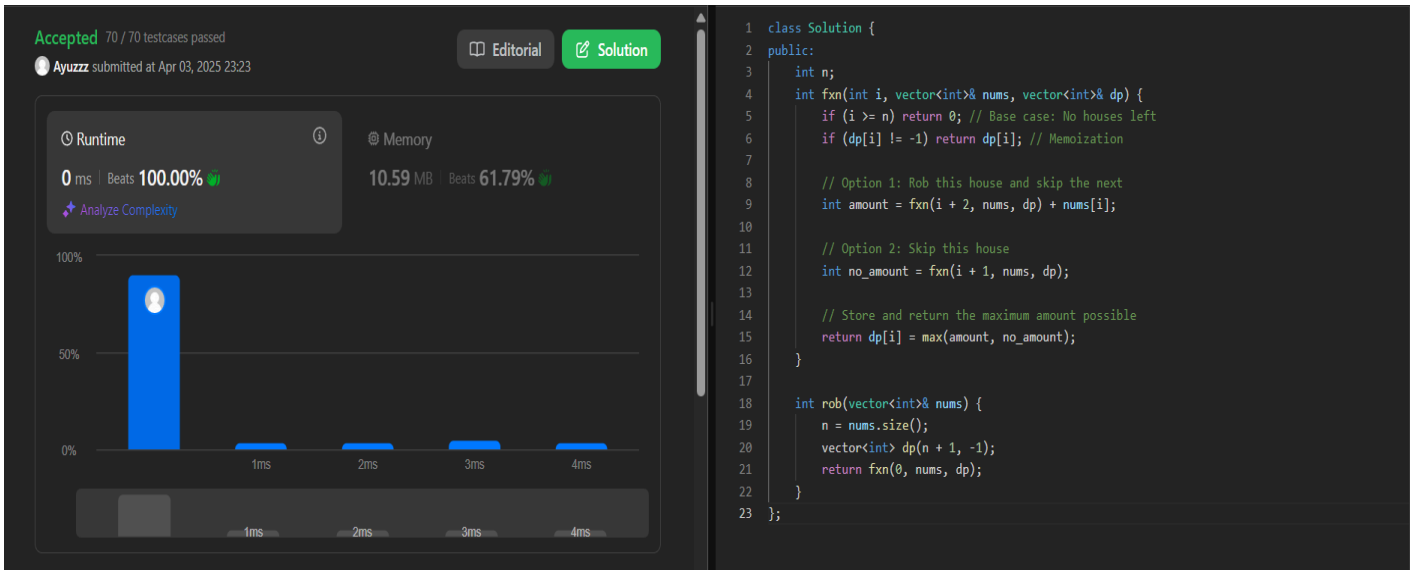
1. Problem - House RObber:

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 n;
    int fxn(int i, vector<int>& nums, vector<int>& dp) {
        if (i >= n) return 0; // Base case: No houses left
        if (dp[i] != -1) return dp[i]; // Memoization

        // Option 1: Rob this house and skip the next
        int amount = fxn(i + 2, nums, dp) + nums[i];

        // Option 2: Skip this house
        int no_amount = fxn(i + 1, nums, dp);

        // Store and return the maximum amount possible
        return dp[i] = max(amount, no_amount);
    }

    int rob(vector<int>& nums) {
        n = nums.size();
        vector<int> dp(n + 1, -1);
        return fxn(0, nums, dp);
    }
};
```

Output:



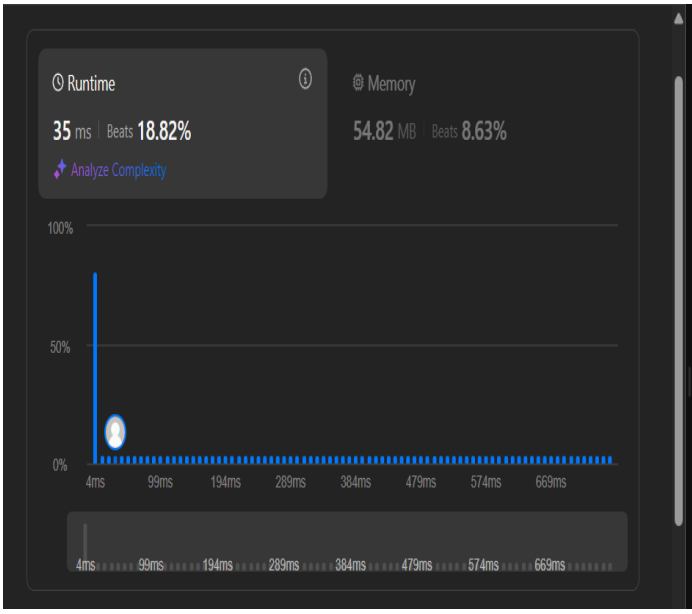
2.Problem -: Jump Game

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.

Code:

```
class Solution {
    vector<int> memo;
    public:
    bool canJump(vector<int>& nums) {
        int n=nums.size();
        vector<int> dp(n,0);
        dp[0]=true;

        for(int i=1;i<n;i++){
            for(int j=i-1;j>=0;j--){
                if(dp[j] && j+nums[j]>=i){
                    dp[i]=true;
                    break;
                }
            }
        }
        return dp[n-1];
    }
};
```




```
class Solution {
    vector<int> memo;
    public:
    bool canJump(vector<int>& nums) {
        int n=nums.size();
        vector<int> dp(n,0);
        dp[0]=true;

        for(int i=1;i<n;i++){
            for(int j=i-1;j>=0;j--){
                if(dp[j] && j+nums[j]>=i){
                    dp[i]=true;
                    break;
                }
            }
        }
        return dp[n-1];
    }
};
```

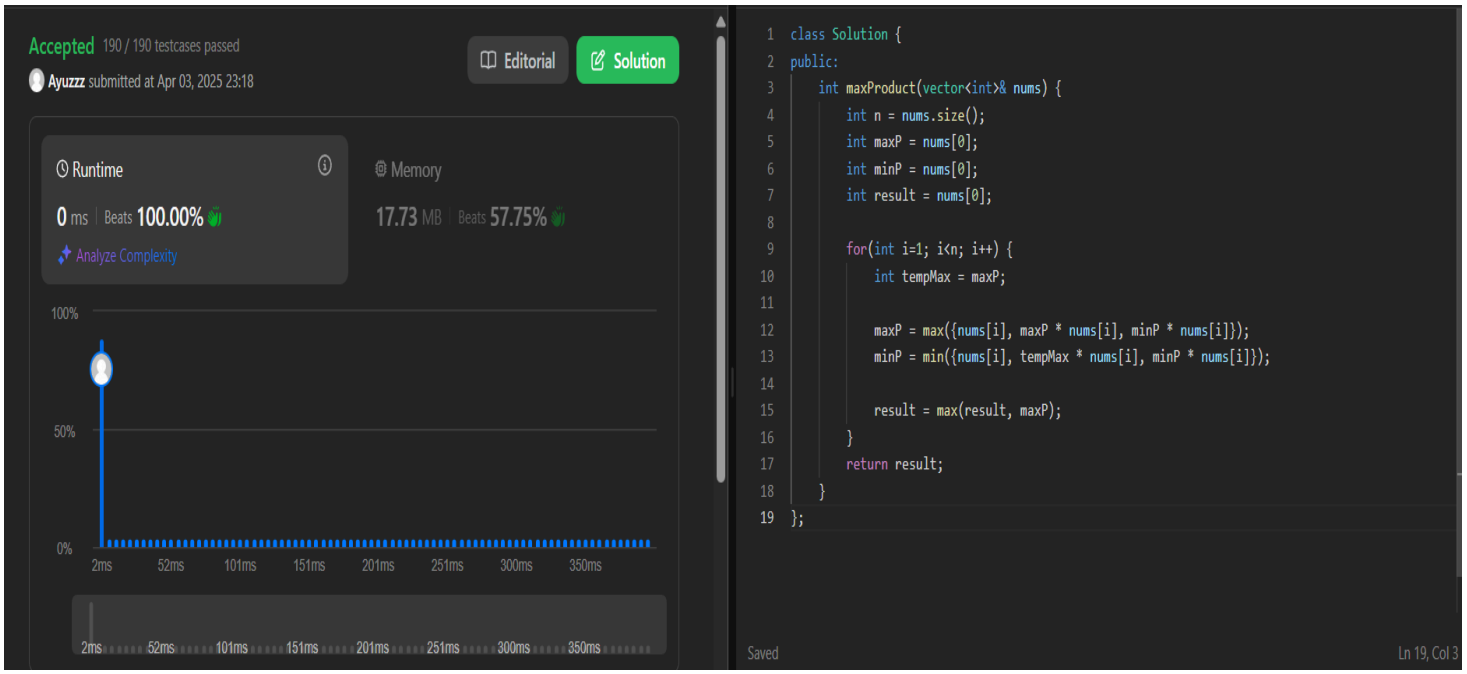
206.Problem - Maximum Product Subarray:

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

Code:

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int n = nums.size();
        int maxP = nums[0];
        int minP = nums[0];
        int result = nums[0];

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

            maxP = max({nums[i], maxP * nums[i], minP * nums[i]});
            minP = min({nums[i], tempMax * nums[i], minP * nums[i]});

            result = max(result, maxP);
        }
        return result;
    }
};
```

Output:



2095.Problem - Coin Change:

Code:

```
class Solution {
public:
    int coinChange(vector<int>& coins, int n) {
        // creating the base dp array, with first value set to 0
        int dp[++n];
        dp[0] = 0;
        // more convenient to have the coins sorted
        sort(begin(coins), end(coins));
        // populating our dp array
        for (int i = 1; i < n; i++) {
            // setting dp[0] base value to 1, 0 for all the rest
            dp[i] = INT_MAX;
            for (int c: coins) {
                if (i - c < 0) break;
                // if it was a previously not reached cell, we do not add use it
                if (dp[i - c] != INT_MAX) dp[i] = min(dp[i], 1 + dp[i - c]);
            }
        }
        return dp[--n] == INT_MAX ? -1 : dp[n];
    }
}
```

Output: