Experiment-7

Student Name: Gaurav Saini
UID: 22BCS11085
Branch: BE-CSE
Section/Group: NTPP_IOT_603_B
Semester: 06
Date of Performance: 28-03-2025

Subject Name: AP LAB-II
Subject Code: 22CSP-351

1. **Aim:**
   a. **Jump Game.**
   b. **Maximum Subarray**
   c. **House Robber**

2. **Introduction to Dynamic Programming:**
   Dynamic Programming (DP) is an optimization technique used to solve complex problems by breaking them into smaller overlapping subproblems, solving each subproblem only once, and storing the results to avoid redundant calculations. It is mainly applied to optimization problems where we need to find the minimum, maximum, shortest, or longest solution.

3. **Implementation/Code:**

A. Jump Game

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
         int maxReach = 0;
        int n = nums.size();

        for (int i = 0; i < n; i++) {
            if (i > maxReach) {
                return false;  }
            maxReach = max(maxReach, i + nums[i]);
            if (maxReach >= n - 1) {
                return true;
            }}
        return false;
    }
};
```

B. Maximum Subarray

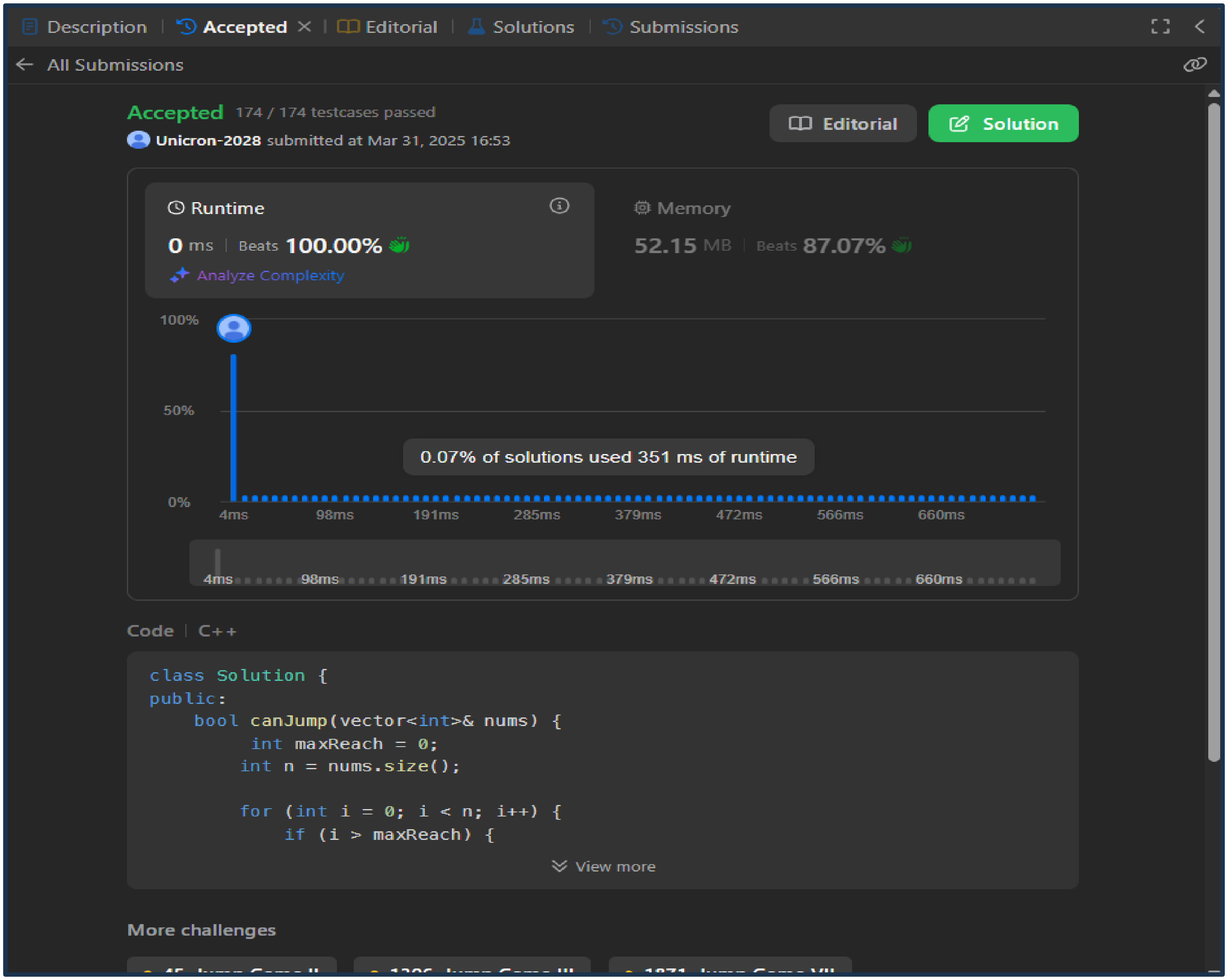
```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = nums[0]; // Initialize max sum as the first element
        int currentSum = nums[0]; // Current subarray sum

        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]); // Extend or restart
subarray
            maxSum = max(maxSum, currentSum); // Update max sum
        }

        return maxSum;
    }
};
```

C. House Robber

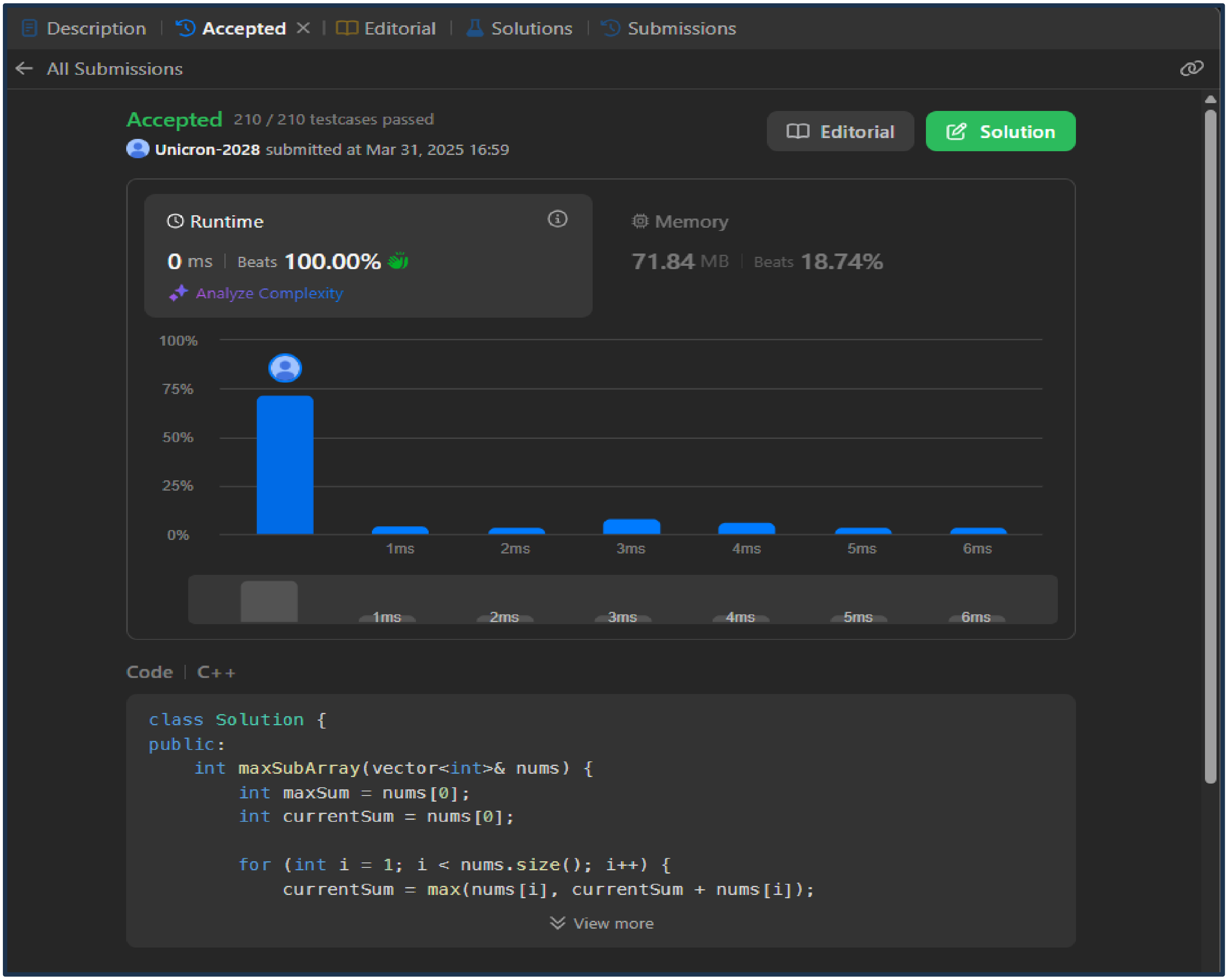
```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
        if (n == 0) return 0;
        if (n == 1) return nums[0];

        vector<int> dp(n, 0);
        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];
    }
};
```

4. Output

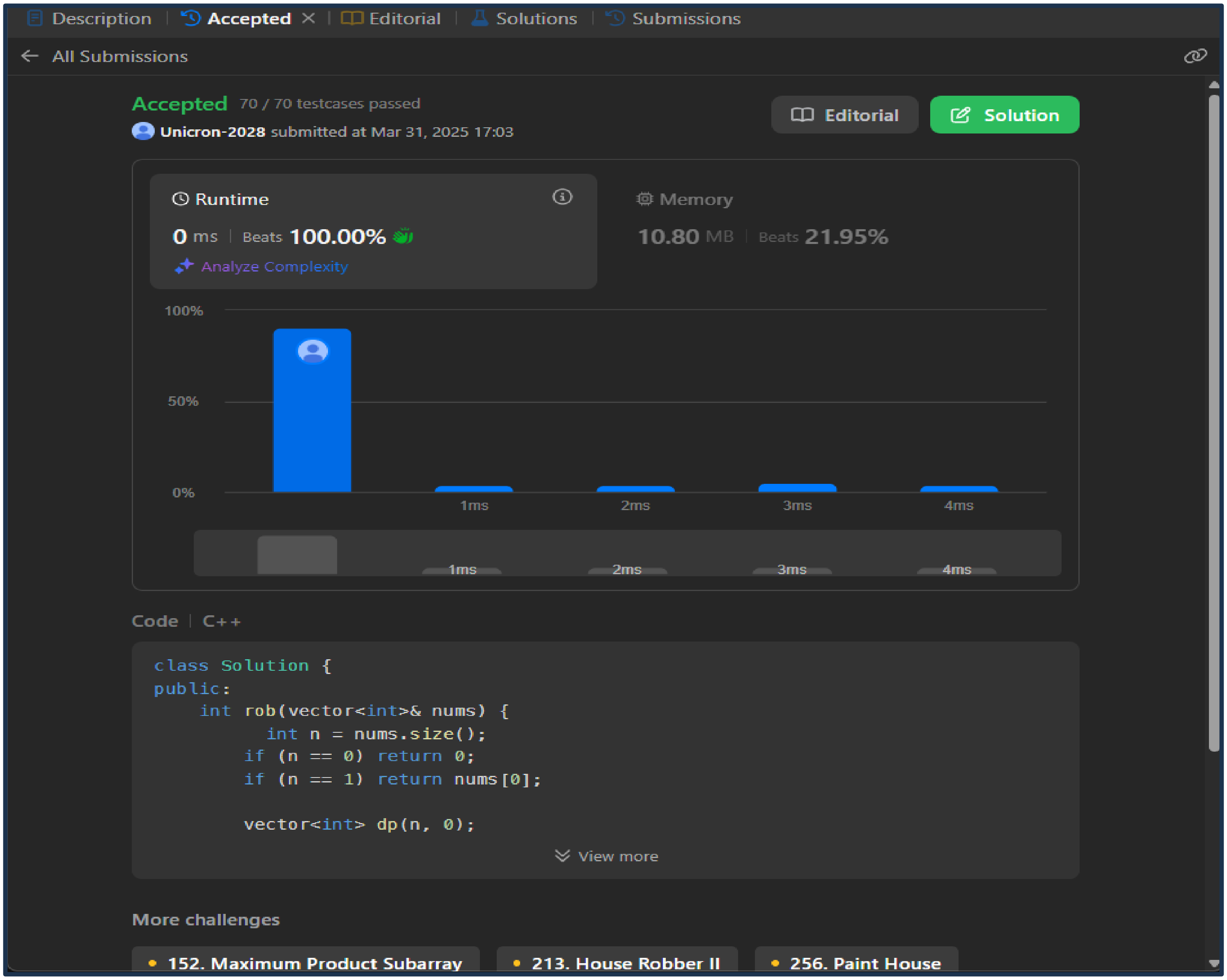
A. Jump Game



B. Maximum Subarray



C. House Robber



5. Learning Outcomes:

□ **Dynamic Programming (DP) Concepts**

- Understanding how to break a problem into subproblems and store intermediate results.
- Identifying overlapping subproblems and optimal substructure in recursive problems.

□ **Optimal Substructure & Recurrence Relation**

- Learning how to formulate the **recurrence relation**:

dp[i]=max⁡(dp[i−1],dp[i−2]+nums[i])dp[i] = \max(dp[i-1], dp[i-2] + nums[i])dp[i]=max(dp[i−1],dp[i−2]+nums[i])

- Understanding how each step builds on previous solutions.

□ **Time & Space Complexity Analysis**

- The DP approach runs in **O(n) time complexity** since we iterate through the array once.
- Using an array for DP results in **O(n) space complexity**, which can be optimized to **O(1) space** with two variables.

□ **Alternative Approaches**

- Learning how to optimize the DP approach by **reducing space complexity**.
- Exploring **recursive with memoization** vs. **iterative DP** solutions.

□ **Problem-Solving Techniques**

- How to **transform real-world constraints** (no adjacent houses robbed) into a computational model.
- Developing a **step-by-step approach** to solving problems using mathematical reasoning.