Experiment 2

Student Name: Rishikesh Raj
UID: 22BCS17196
Branch: CSE
Section/Group: 627-A
Semester: 6th
Date of Performance: 24/01/25
Subject Name: Advanced Programming Lab-2
Subject Code: 22CSP-351

1. **Aim:**
   - **a)** Given an array of integers nums and an integer target, return the indices of the two numbers such that they add up to target. Each input has exactly one solution, and you cannot use the same element twice.
   - **b)** You are given a 0-indexed array nums of length n. You are initially positioned at nums[0]. Each element nums[i] represents the maximum length of a forward jump from index i. Return the minimum number of jumps to reach nums[n - 1].

2. **Objective:**
   - **a)** Develop an efficient algorithm to identify two numbers in an array that sum up to a given target using techniques like hashing for optimal performance.
   - **b)** Implement a greedy approach to minimize the number of jumps required to reach the last index, ensuring an optimal path selection for efficiency.

3. **Algorithm:**
   - **a) Algorithm for Two Sum problem:-**

1) Initialize jumps as 0 (the number of jumps needed), current_end as 0 (the farthest index you can reach in the current jump), and farthest as 0 (the farthest index you can reach so far).
2) Loop through each index in nums (except the last index):
3) Update farthest to the maximum of farthest and i + nums[i].
4) If i == current_end, increment jumps and set current_end = farthest.
5) Return jumps when current_end reaches or exceeds the last index (n - 1).

   - **b) Algorithm for Jump Game II:-**

1) Initialize jumps = 0 (to count the number of jumps), current_end = 0 (the farthest index reachable in the current jump), and farthest = 0 (the farthest index reachable so far).
2) Loop through each index i in nums (excluding the last index).
3) Update farthest to max(farthest, i + nums[i]).
4) If i reaches current_end, increment jumps and set current_end = farthest.
5) Return jumps when current_end reaches or exceeds n - 1.

4. **Implementation/Code:**

a) **Code for Two Sum Problem:-**

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int n = nums.size();
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                if (nums[i] + nums[j] == target) {
                    return {i, j};
                }
            }
        }
        return {}; // No solution found
    }
};
```

b) **Code for Jump Game II problem:-**

```
class Solution {
public:
    int jump(vector<int>& nums) {
        int jumps = 0, currentEnd = 0, farthest = 0;
        for (int i = 0; i < nums.size() - 1; i++) {
            farthest = max(farthest, i + nums[i]);
            if (i == currentEnd) {
                jumps++;
                currentEnd = farthest;
            }
        }
        return jumps;
    }
};
```

5. Output:

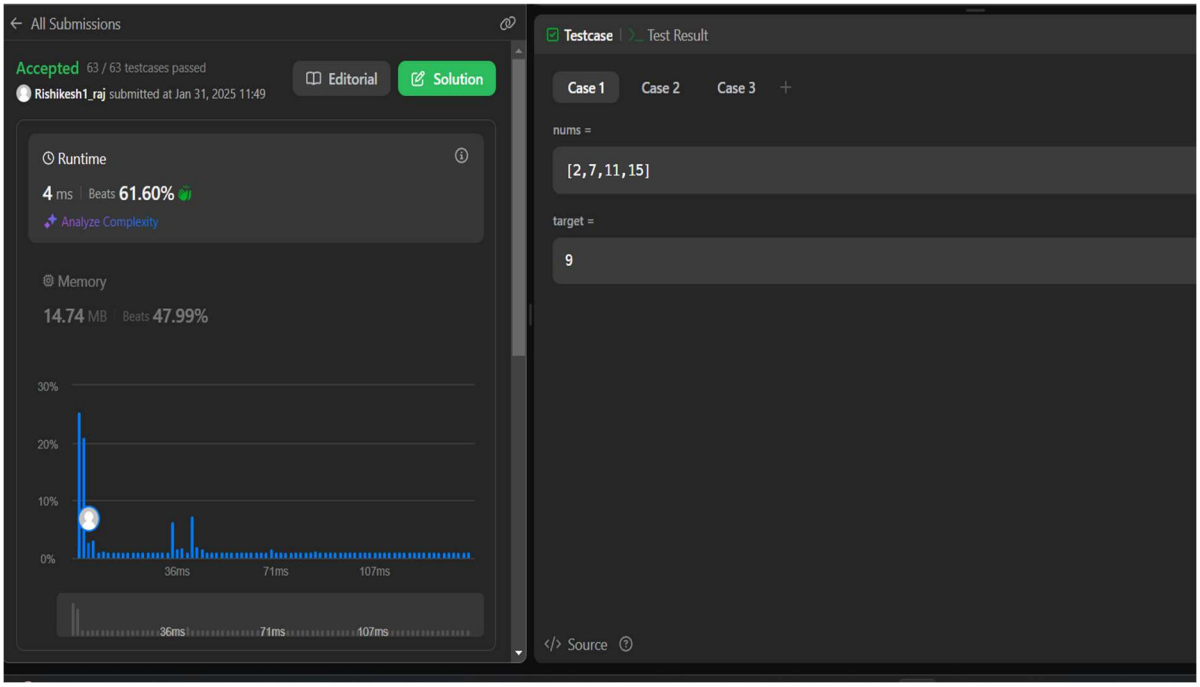

Fig 1:- Output for Two Sum

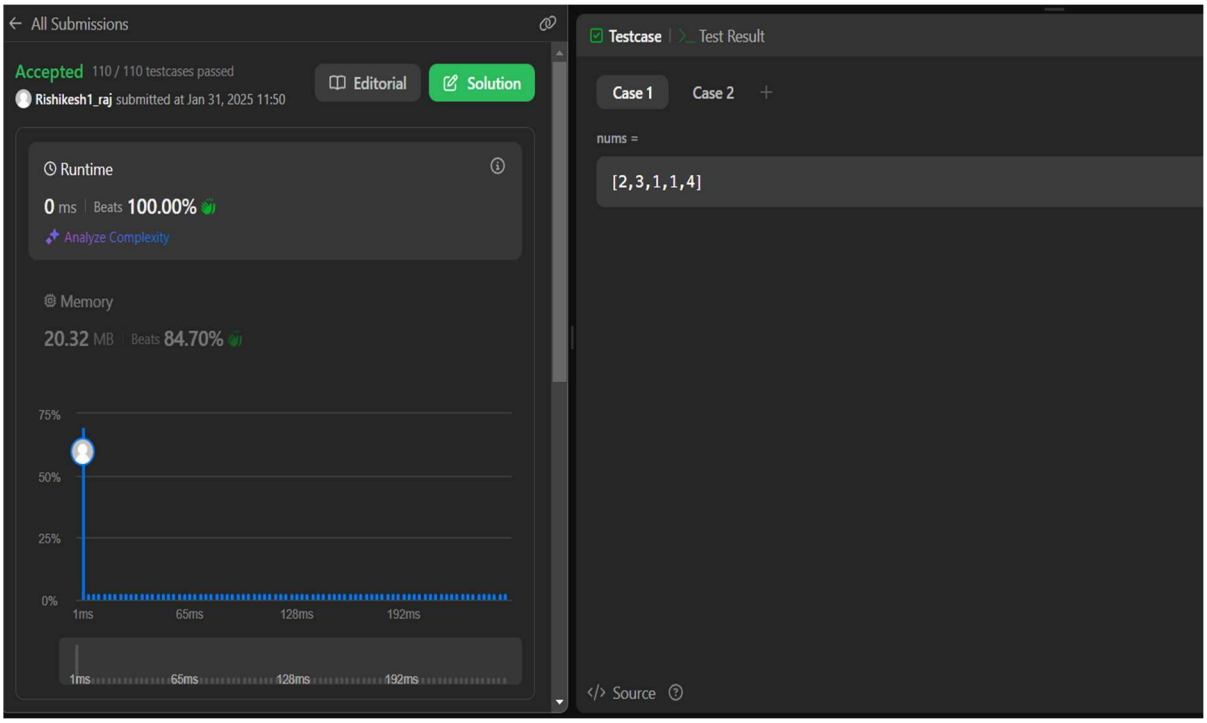

Fig 2:- Output for Jump Game

6. Learning Outcomes:

1) Understanding of array traversal and index-based operations to solve problems efficiently.
2) Application of hash maps for optimizing search operations in the Two Sum problem.
3) Implementation of a greedy approach to determine the minimum jumps needed in the Jump Game problem.
4) Enhanced problem-solving skills by breaking down problems into smaller logical steps.
5) Improved time complexity analysis by comparing brute-force, optimized, and greedy approaches.