DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment 1.2

Student Name: Masud Alom
UID: 22BCS16095
Branch: BE-CSE
Section/Group: NTPP-602-A
Semester: 6th
Date of Performance:20-01-25
Subject Name: AP LAB-II
Subject Code: 22CSP-351

1.Aim:

Problem 1.2.1: Two Sum

- **Problem Statement:** 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.

Problem 1.2.2: Jump Game II

- **Problem Statement:** 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].

Problem 1.2.3: Simplify Path

- **Problem Statement**: Given a string path, which is an absolute path to a file or directory in a Unix-style file system, convert it to the simplified canonical path.

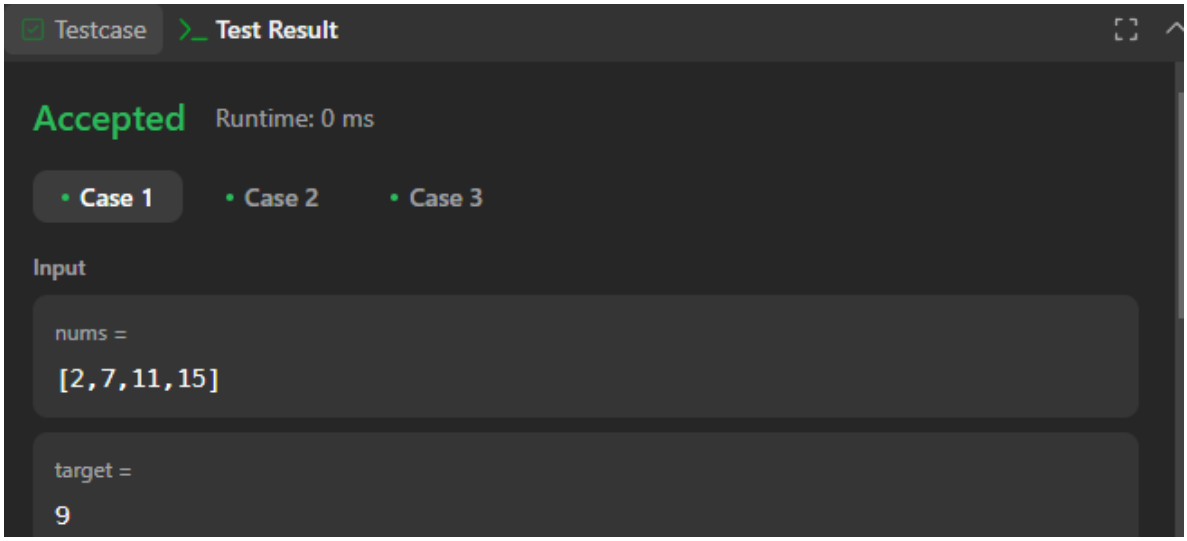
2.Algorithm:

1. Initialize an empty hash map (dict).
2. Iterate through the nums array:
3. For each element num, calculate the complement: complement = target - num.
4. Check if the complement exists in the hash map:
5. If it does, return the indices of the complement and the current number.
6. If it doesn't, add the current number and its index to the hash map. 7. Return the indices of the two numbers that add up to the target.

7. **Code:**

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];

            if (map.containsKey(complement)) {
                return new int[]{map.get(complement), i}; // Found the
pair
            }

            map.put(nums[i], i); // Store the number and its index
        }

        return new int[]{}; // Should never reach here as per the
problem statement
    }
}
```

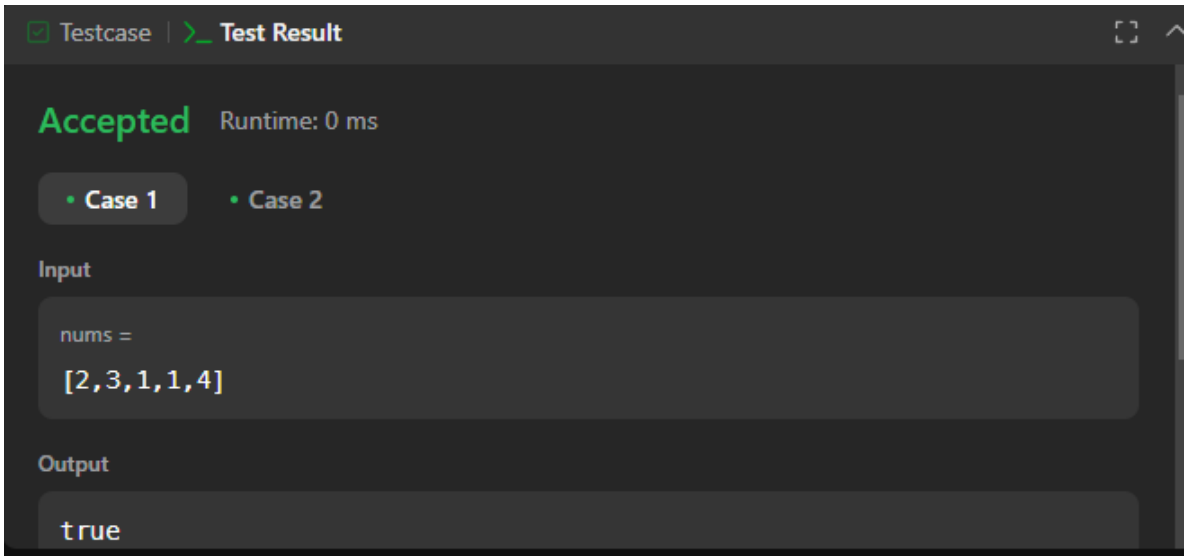
Output:



CODE: 1.2.2

```
class Solution {
    public boolean canJump(int[] nums) {
        int maxReach = 0; // Tracks the farthest index we can reach


        for (int i = 0; i < nums.length; i++) {
            if (i > maxReach) return false; // If current index is unreachable, return false
            maxReach = Math.max(maxReach, i + nums[i]); // Update max reachable index
            if (maxReach >= nums.length - 1) return true; // If last index is reachable, return
true
        }
        return false;
    }
}
```

OUTPUT:



5. Learning Outcomes:-

1. **Array Manipulation** – Solve problems using arrays and index-based operations.
2. **Efficient Algorithms** – Apply hashing and greedy methods for optimization.
3. **Stack Usage** – Use stacks for directory path simplification.
4. **Edge Case Handling** – Manage constraints like duplicates and empty inputs. **Problem-Solving Skills** – Break down problems and debug efficiently.