Q41: Two Sum

1. Understand the Problem

- **Read & Identify:** Given an array of integers nums and a target integer target, find indices of two numbers in nums that sum up to target.
- **Paraphrase:** Find exactly one pair (i, j) such that nums[i] + nums[j] = target

2. Constraints

- 2 <= nums.length <= 10^5
- -10^4 <= nums[i] <= 10^4
- -10^5 <= target <= 10^5
- Exactly one solution exists
- Each element used at most once

3. Examples & Edge Cases

Example 1 (Normal Case): Input:

```
[1,6,2,10,3], target 7
```

Output:

```
[0,1]
```

Example 2 (Normal Case): Input:

```
[1,3,5,-7,6,-3], target 0
```

Output:

```
[1,5]
```

4. Approaches

Approach 1: Brute Force

- **Idea:**
  - Check all possible pairs (i,j) where i<j to see if nums[i]+nums[j] == target.

Java Code:

```java
public int[] twoSumBrute(int[] nums, int target) {
    int n = nums.length;
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            if (nums[i] + nums[j] == target) {
                return new int[]{i, j};
            }
        }
    }
    return new int[]{-1, -1}; // should never reach here
}
```

Complexity:

- Time: O(n^2) → two nested loops.
- Space: O(1) → no extra space.

Approach 2: Optimal (HashMap)

- **Idea:**
  - Traverse array and store value → index in a HashMap.
  - For each element num, check if target - num exists in the map.
  - Return indices immediately when found.

Java Code:

```java
import java.util.*;

public int[] twoSumOptimal(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};
        }
        map.put(nums[i], i);
    }
    return new int[]{-1, -1}; // should never reach here
}
```

Complexity:

- Time: O(n) → single traversal.
- Space: O(n) → for the HashMap.

---

5. Justification / Proof of Optimality

- Brute Force: Simple, but inefficient for large arrays (n^2).
- HashMap Approach: Efficient, single pass, handles negative numbers and duplicates, optimal solution.

---

6. Variants / Follow-Ups

- Find all pairs summing to target (multiple solutions).
- Array is sorted → can use two-pointer approach instead of HashMap.
- Return values instead of indices.
- Target sum for more than two numbers → generalizes to 3-sum, 4-sum problems.