# **Leetcode 1 – Two Sum**

## Problem Understanding

You're given an array of integers nums and a target sum target.  
**Goal**: Return the **indices of two numbers** such that they **add up to the target**.

* Each input has exactly **one solution**.
* You **cannot use the same element twice**.
* Return indices in **any order**.

### Example

Input: nums = [2, 7, 11, 15], target = 9

Output: [0, 1]

Explanation: nums[0] + nums[1] = 2 + 7 = 9

## Optimized Java Solution (Using HashMap)

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 };

}

map.put(nums[i], i);

}

return new int[0]; // Not expected, as one solution is guaranteed

}

}

## Dry Run Using Table

### Input:

nums = [2, 11, 7, 15], target = 9

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | nums[i] | complement = target - nums[i] | map before | match? | map after |
| 0 | 2 | 7 | {} | ❌ | {2 → 0} |
| 1 | 11 | -2 | {2 → 0} | ❌ | {2 → 0, 11 → 1} |
| 2 | 7 | 2 | {2 → 0, 11 → 1} | ✅ (found 2) | — return [0, 2] |

✅ Output: [0, 2]

## Time / Space Complexity

|  |  |
| --- | --- |
| Metric | Value |
| Time | O(n) |
| Space | O(n) |

Because we use a HashMap to store indices as we traverse.

## Alternate Approaches

|  |  |  |  |
| --- | --- | --- | --- |
| Approach | Time | Space | Notes |
| ✅ HashMap | O(n) | O(n) | Best solution |
| Brute Force | O(n²) | O(1) | Check all pairs, inefficient |
| Two Pointers | O(n log n) | O(1) | Sort + pointers (loses original indices) |