LeetCode 1 - Two Sum

1. Problem Title & Link

1. Two Sum

🔗 https://leetcode.com/problems/two-sum/

2. Problem Summary

Given an array nums and an integer target,
return **indices** of the **two numbers** such that:
nums[i] + nums[j] = target
Rules:

- Each input has **exactly one solution**
- Cannot use the same element twice
- Return indices in **any order**

3. Examples

Example 1

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: 2 + 7 = 9
```

Example 2

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

4. Constraints

- $2 <= nums.length <= 10^4$
- $-10^9 <= nums[i], target <= 10^9$
- Exactly one valid answer exists

5. Thought Process (Step-by-Step)

For each number x, we want another number y = target - x.
We store previously seen numbers in a **hashmap**:
value → index
At each index i:

- Let complement = target - nums[i]
- If complement already in map → answer found
- Otherwise add nums[i] → index

This guarantees **O(1)** lookup per element → overall **O(n)**.

6. Pseudocode

```
map = {}

for i from 0..n-1:
    complement = target - nums[i]
    if complement in map:
        return [map[complement], i]
    map[nums[i]] = i
```

7. Code Implementation (Python)

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        seen = {}   # value → index

        for i, num in enumerate(nums):
            complement = target - num

            # If complement found earlier, return indices
            if complement in seen:
                return [seen[complement], i]

            # Store current number's index
            seen[num] = i

        # Problem guarantees exactly one solution
        return []
```

8. Time & Space Complexity

Metric	Complexity	Why
Time	O(n)	One pass + O(1) lookup
Space	O(n)	Hashmap storing numbers

9. Dry Run

Input:

nums = [3,2,4], target = 6

i	num	complement	seen	Action
0	3	3	{}	store 3:0
1	2	4	{3:0}	store 2:1
2	4	2	{3:0,2:1}	complement 2 found → return [1,2]

Output:

[1,2]

10. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds
Hash Map Lookups (value → index)	Two-sum, complement pairs, frequency lookups	Using same element twice, not storing before checking	LC 167, LC 170, LC 560

11. Common Mistakes

- Doing nested loops → $O(n^2)$
- Adding current number to map **before** checking complement (wrong order)
- Forgetting that we must return **indices**, not values

12. Variations / Follow-Ups

- **LC 167:** Two Sum II (sorted array → two pointers)
- **LC 170:** Two Sum III (data structure design)
- **LC 653:** Two Sum IV (BST)
- **LC 560:** Subarray Sum Equals K (prefix-sum)