LeetCode 704 — Binary Search

1. Problem Title & Link

- **Title:** LeetCode 704 — Binary Search
- **Link:** https://leetcode.com/problems/binary-search/

2. Problem Statement (Short Summary)

You are given a **sorted array** nums and a target.
Return the **index** of target if it exists.
Otherwise return **-1**.
You MUST use **O(log n)** binary search.

3. Examples (Input → Output)

Example 1

Input: nums = [-1,0,3,5,9,12], target = 9

Output: 4

Example 2

Input: nums = [-1,0,3,5,9,12], target = 2

Output: -1

Example 3

Input: nums = [5], target = 5

Output: 0

4. Constraints

- 1 <= nums.length <= 10^4
- -10^4 <= nums[i], target <= 10^4
- Array is **sorted in ascending order**
- Must use **O(log n)** time

5. Core Concept (Pattern / Topic)

Binary Search

Technique to search in a sorted array by dividing the search space into half each time.
Key idea:

- Compute mid = (left + right) // 2
- If nums[mid] == target → return mid
- If target < nums[mid] → search left half
- Else → search right half

6. Thought Process (Step-by-Step Explanation)

1. Start with left = 0, right = n-1
2. While left <= right:
    - Compute middle index
    - Compare with target
3. If mid value > target → reduce right
4. If mid value < target → increase left
5. Else return mid
6. If loop ends → return -1

Binary search guarantees O(log n) time.

7. Visual / Intuition Diagram

Example:

nums = [-1,0,3,5,9,12], target = 9

Step 1:

left=0, right=5

mid=2 → nums[2]=3 < 9 → search right

Step 2:

left=3, right=5

mid=4 → nums[4]=9 FOUND

Diagram:

```
[-1   0   3   5   9   12]
  L           M           R

[-1   0   3   5   9   12]
              L   M       R
```

8. Pseudocode

```
left = 0
right = n-1

while left <= right:
    mid = (left + right) // 2
    if nums[mid] == target: return mid
    else if target < nums[mid]: right = mid - 1
    else: left = mid + 1
```

```
return -1
```

9. Code Implementation

✅ **Python**

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums) - 1

        while left <= right:
            mid = (left + right) // 2

            if nums[mid] == target:
                return mid
            elif nums[mid] < target:
                left = mid + 1
            else:
                right = mid - 1

        return -1
```

✅ **Java**

```java
class Solution {
    public int search(int[] nums, int target) {
        int left = 0, right = nums.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] == target)
                return mid;
            else if (nums[mid] < target)
                left = mid + 1;
            else
                right = mid - 1;
        }

        return -1;
    }
}
```

10. Time & Space Complexity

Metric	Complexity
Time	O(log n)
Space	O(1)

11. Common Mistakes / Edge Cases

- Forgetting left <= right (should NOT be <)
- Wrong mid formula causing overflow in Java (left + (right-left)/2 is safe)
- Infinite loop due to wrong pointer movement
- Not returning -1 for missing target

Edge cases:

- target at index 0
- target at last index
- array size 1

12. Detailed Dry Run (Step-by-Step Table)

Input:

nums = [-1,0,3,5,9,12]

target = 9

Step	left	right	mid	nums[mid]	Action
1	0	5	2	3	3 < 9 → left = 3
2	3	5	4	9	MATCH → return 4

Answer = **4**

13. Common Use Cases

- Searching in sorted arrays
- Peak finding
- First/last occurrence finding
- Lower bound / upper bound problems
- Binary search on answer space

14. Common Traps

- Using while left < right (wrong here)

- Updating left/right incorrectly
- Returning mid without checking

15. Builds To (Related Problems)

- LC 35 — Search Insert Position
- LC 278 — First Bad Version
- LC 374 — Guess Number
- LC 34 — First and Last Position
- LC 875 — Koko Eating Bananas (Binary search on answer)

16. Alternate Approaches + Comparison

Approach	Time	Space	Notes
Linear Search	O(n)	O(1)	Too slow
Binary Search	O(log n)	O(1)	Optimal ✔

17. Why This Solution Works (Short Intuition)

Each comparison removes half of the search space,
ensuring logarithmic time complexity.

18. Variations / Follow-Up Questions

- What if array is rotated? → LC 33
- Binary search on descending array
- Binary search on infinite array
- Find first/last position of target