LeetCode 33 — Search in Rotated Sorted Array

1. Problem Title & Link

- **Title:** LeetCode 33 — Search in Rotated Sorted Array
- **Link:** https://leetcode.com/problems/search-in-rotated-sorted-array/

2. Problem Statement (Short Summary)

You are given a **sorted array**, but it has been **rotated at some pivot** you don't know.
You must **return the index of target** if it exists; otherwise return -1.

Must run in **O(log n)** time → binary search.

Example of rotated array:

Original: [0,1,2,4,5,6,7]

Rotated: [4,5,6,7,0,1,2]

3. Examples (Input → Output)

Example 1

Input: nums = [4,5,6,7,0,1,2], target = 0

Output: 4

Example 2

Input: nums = [4,5,6,7,0,1,2], target = 3

Output: -1

Example 3

Input: nums = [1], target = 0

Output: -1

4. Constraints

- 1 <= nums.length <= 5000
- Values in array are **unique**
- Array is sorted but rotated
- Must use binary search (O(log n))

5. Core Concept (Pattern / Topic)

Binary Search on Rotated Sorted Array

At any moment:

- **One half is always sorted**
- Decide which half to discard based on the target's range

This is a **modified binary search**.

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

Key Observation:

For any mid:

- Either **left half** (nums[left] → nums[mid]) is sorted
- Or **right half** (nums[mid] → nums[right]) is sorted

Steps:

1. Compute mid
2. Check if nums[mid] == target
3. If left half is sorted:
   - Check if target lies between left and mid
4. Else right half is sorted:
   - Check if target lies between mid and right
5. Narrow search accordingly

Binary search continues until found or exhausted.

7. Visual / Intuition Diagram

```
Array:
[4,5,6,7,0,1,2]
Diagram:
Left sorted    Pivot      Right sorted
[4 5 6 7]   |    [0 1 2]
When mid = 7:
4 5 6 [7] 0 1 2
Left side sorted → check if target in [4..7]
When mid = 0:
4 5 6 7 [0] 1 2
Right side sorted → check if target in [0..2]
```

8. Pseudocode

```
left = 0
right = n-1

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

    if nums[mid] == target:
        return mid

    if nums[left] <= nums[mid]:   # left sorted
```

```
        if nums[left] <= target < nums[mid]:
            right = mid - 1
        else:
            left = mid + 1
    else:                         # right sorted
        if nums[mid] < target <= nums[right]:
            left = mid + 1
        else:
            right = mid - 1

return -1
```

9. Code Implementation

✅ 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

            # Left sorted
            if nums[left] <= nums[mid]:
                if nums[left] <= target < nums[mid]:
                    right = mid - 1
                else:
                    left = mid + 1
            # Right sorted
            else:
                if nums[mid] < target <= nums[right]:
                    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;

            // Left half sorted
            if (nums[left] <= nums[mid]) {
                if (nums[left] <= target && target < nums[mid]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            }
            // Right half sorted
            else {
                if (nums[mid] < target && target <= nums[right]) {
                    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

❌ Misidentifying which half is sorted

❌ Forgetting <= boundary

❌ Infinite loops because of wrong pointer moves

❌ Missing case where nums[left] <= nums[mid] even if pivot nearby

Edge cases:

- Array not rotated (normal binary search)
- Pivot at index 0
- Pivot at last index
- Single element array

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

Input:

nums = [4,5,6,7,0,1,2]

target = 0

Step	left	mid	right	nums[mid]	Sorted Half	Action
1	0	3	6	7	left sorted	target NOT in [4..7] → left = 4
2	4	5	6	1	right sorted	target in [0..2] → right = 4
3	4	4	4	0	match	return 4

Answer: **4**

13. Common Use Cases

- Rotated arrays
- Searching in circular buffers
- Searching in rotated sorted logs
- Clockwise/anti-clockwise indexing

14. Common Traps

- Checking sorted half incorrectly
- Using < instead of <=
- Missing target in boundary cases
- Wrong movement of left/right pointers

15. Builds To (Related Problems)

- **LC 81** — Search in Rotated Sorted Array II (duplicates)
- **LC 153** — Find Minimum in Rotated Array
- **LC 154** — Min in rotated with duplicates
- **LC 34** — First/Last position of target

16. Alternate Approaches + Comparison

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

17. Why This Solution Works (Short Intuition)

A rotated sorted array always has **one sorted side**.
Binary search determines which side is sorted and eliminates half the search space each time.

18. Variations / Follow-Up Questions

- How to handle duplicates? (LC 81)
- How to find the pivot index?
- How to rotate + search multiple times efficiently?
- What if the array is rotated k times dynamically?