Searching – Search in Rotated Sorted Array

1. Introduction

A **rotated sorted array** is an array that was originally sorted but then **rotated (shifted) at some pivot point**.

Searching in such an array requires a **modified Binary Search** approach.

This problem is important because it tests a learner's understanding of:

- Binary search logic
- Array properties
- Decision making using conditions

---

2. What is a Rotated Sorted Array?

A rotated sorted array is created by taking a sorted array and rotating it around a pivot.

Example:

Sorted array:

```
[1, 2, 3, 4, 5, 6, 7]
```

Rotated array:

```
[4, 5, 6, 7, 1, 2, 3]
```

Here, the array is rotated at index 3.

---

3. Why is This Problem Important?

Searching in a rotated array is important because:

- It combines sorting and searching concepts
- It is a common **interview question**
- It strengthens conditional logic skills
- It introduces problem-solving using patterns

---

4. Key Observation

In a rotated sorted array:

- **At least one half of the array is always sorted**

This observation is the foundation of the solution.

---

5. Basic Idea Behind the Search

Instead of applying normal binary search directly:

- Identify which half (left or right) is sorted
- Check whether the target lies in the sorted half
- If yes, search there
- Otherwise, search the other half

---

6. Logic for Search in Rotated Array (Plain English)

1. Set start and end pointers
2. Find the middle element
3. If middle equals target, return index
4. Check which half is sorted:
    - If left half is sorted:
        - If target lies in left half → search left
        - Else → search right
    - If right half is sorted:

- If target lies in right half → search right
- Else → search left

5. Repeat until element is found or search ends

---

7. Visualization Example

Rotated array:

```
[6, 7, 1, 2, 3, 4, 5]
```

Target:

```
3
```

Search flow:

```
Mid = 2 → right half sorted
Target in right half → search right
Mid = 3 → found
```

---

8. Time and Space Complexity

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

This makes the approach efficient even for large arrays.

---

9. Advantages

- Efficient searching
- Uses binary search principles
- Handles rotated arrays correctly

- Suitable for large datasets

10. Limitations

- Requires careful condition handling
- Slightly more complex than normal binary search
- Assumes no duplicate elements (basic version)

11. Real-World Applications

- Searching cyclically shifted logs
- Circular data buffers
- Rotated schedules
- Interview and competitive programming problems

12. Comparison with Normal Binary Search

Feature	Binary Search	Rotated Array Search
Array Type	Sorted	Rotated sorted
Complexity	O(log n)	O(log n)
Logic	Simple	Conditional
Difficulty	Easy	Medium

13. Summary

- A rotated array is a shifted sorted array
- At least one half is always sorted
- Modified binary search is used
- Time complexity is O(log n)
- Important interview-level problem