Search in Rotated Sorted Array

The problem involves searching for a target in a rotated sorted array with unique elements, achieving O(log n) time complexity using modified binary search.

---

1. Problem Understanding

Key Characteristics

- Array was **originally sorted in ascending order** (e.g., [0,1,2,3,4,5,6,7])
- Array was **rotated** at some pivot (e.g., [4,5,6,7,0,1,2,3])
- **All elements are unique** (distinct values)
- Find **index of target** or return -1 if not found

Brute Force Approach

- **Linear search**: Check each index one by one
- **Time Complexity**: $$O(n)$$

Key Insight: Array is **sorted** → Think **binary search** for optimization

---

2. Why Normal Binary Search Fails

```
Example: [4,5,6,7,0,1,2,3], target = 0
start=0, end=7, mid=3 (arr[3]=7)
```

Normal binary search logic:

- If **target < arr[mid]**: Search **left half**
- If **target > arr[mid]**: Search **right half**

Problem: target=0 < arr[mid]=7, but 0 is in **RIGHT half**! Normal logic **fails**

---

3. Core Insight: Always One Sorted Half

🔑 **Critical Observation**: In any rotated sorted array, **either left half OR right half is ALWAYS sorted**

Examples

Array	Left Half	Right Half	Sorted Half
[4,5,6,7,0,1,2,3]	[4,5,6,7]	[0,1,2,3]	**Left**
[6,7,0,1,2,3,4,5]	[6,7,0]	[1,2,3,4,5]	**Right**

Strategy:

1. Identify **which half is sorted**
2. Apply **binary search conditions** on that sorted half

---

4. Algorithm Steps

Initialization

```
start = 0
end = n-1
while start <= end:
    mid = start + (end - start) / 2  // Avoid overflow
```

Step-by-Step Logic

5. Detailed Conditions

Check Which Half is Sorted

```
if arr[start] ≤ arr[mid]:
    // Left half is sorted
else:
    // Right half is sorted
```

Left Half Sorted - Binary Search Conditions

```
if target ≥ arr[start] AND target ≤ arr[mid]:
    end = mid - 1  // Search left
else:
    start = mid + 1  // Search right
```

Right Half Sorted - Binary Search Conditions

```
if target ≥ arr[mid] AND target ≤ arr[end]:
    start = mid + 1  // Search right
else:
    end = mid - 1  // Search left
```

6. Trace Example 1: [4,5,6,7,0,1,2,3], target=0

Iteration	start	end	mid	arr[mid]	Left Sorted?	Decision	New Range
1	0	7	3	7	Yes (4≤7)	0∉[4,7]	[4,7]
2	4	7	5	1	No	0∉[1,3]	[4,4]
3	4	4	4	0	-	**Found!**	Return 4

7. Trace Example 2: [6,7,0,1,2,3,4,5], target=0

Iteration	start	end	mid	arr[mid]	Left Sorted?	Decision	New Range
1	0	7	3	1	No (6>1)	0∉[1,5]	[0,2]
2	0	2	1	7	Yes (6≤7)	0∉[6,7]	[2,2]
3	2	2	2	0	-	**Found!**	Return 2

8. Complete Pseudocode

```
function search(nums, target):
    start = 0
```

```
end = nums.length - 1

while start <= end:
    mid = start + (end - start) / 2

    if nums[mid] == target:
        return mid

    // Check if left half is sorted
    if nums[start] <= nums[mid]:
        // Apply binary search on left half
        if target >= nums[start] AND target < nums[mid]:
            end = mid - 1
        else:
            start = mid + 1
    else:
        // Right half is sorted
        if target > nums[mid] AND target <= nums[end]:
            start = mid + 1
        else:
            end = mid - 1

return -1
```

9. Time & Space Complexity

Approach	Time Complexity	Space Complexity
Linear Search	O(n)	O(1)
Modified Binary Search	**O(log n)**	**O(1)**