Cheat Sheet: Search in Rotated Sorted Array

Foundational Array Concept

A **rotated sorted array** is an array that was **initially sorted in ascending order**, but then rotated at some pivot (unknown to us). For example:

Original sorted array: `[0, 1, 2, 4, 5, 6, 7]`
After rotation: `[4, 5, 6, 7, 0, 1, 2]`

Even though the array is no longer fully sorted, **each rotation results in two sorted subarrays**. This is the key property leveraged in binary search.

Problem Statement

Given: An integer array `nums`, sorted in ascending order, then rotated at some pivot.
Task: Search for a given `target` in `nums`.
Goal: Return the index of target if found, else `-1`.
Constraint: Must run in **O(log n)** time.

Programming Approach

The problem is solved using a **modified binary search**:

1. Calculate `mid` as the midpoint between `low` and `high`.
2. Determine which **half is sorted**:
   - If `nums[low] <= nums[mid]`, left half is sorted.
   - Else, right half is sorted.
3. Use the sorted half to decide where to search:
   - If the target lies in the sorted half, continue there.
   - Else, move to the other half.
4. Repeat until the element is found or the range is exhausted.

Time Complexity: `O(log n)`
Space Complexity: `O(1)` (no extra space used)**Java Implementation**

```
public int search(int[] nums, int target) {
    int low = 0, high = nums.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;

        if (nums[mid] == target)
            return mid;

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

    return -1;
}
```

Example Walkthrough

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

- `mid = 3`, `nums[mid] = 7`
  Left half `[4,5,6,7]` is sorted but does not contain target
- Move to right half: `low = 4`, `high = 6`
- `mid = 5`, `nums[5] = 1`
  Right half `[0,1,2]` is sorted, target `0` found at index `4`

Edge Cases

- Not rotated: `[1, 2, 3, 4]`
- Pivot is at index `0`: No change
- Duplicates (advanced): Need alternate logic (see LeetCode 81)

Related Problems

- LeetCode 33: Search in Rotated Sorted Array
- LeetCode 81: Search in Rotated Sorted Array II (with duplicates)
- GFG: Search in Rotated Sorted Array

Suggested Points:

Use this technique when:

- You're given a **rotated version** of a **sorted array**.
- You must maintain **O(log n)** performance.
- You can leverage the **sorted structure** within segments of the array.