Search in Rotated Sorted Array

This problem requires O(log n) runtime, which implies a modified binary search.

Key Idea: Even though the array is rotated, one of the halves is always sorted:
- Either nums[left... mid] is sorted, or
- nums[mid... right] is sorted.

We determine which half is sorted and check if the target lies in that half. If yes, search that half; else search the other half.

Algorithm Steps:

1. Initialize two pointers:
    - left = 0, right = nums.size() - 1
2. While left <= right:
    - Find mid = (left + right) / 2
    - If nums[mid] == target → return mid
    - Determine which half is sorted:
        - If nums[left] <= nums[mid] → Left half sorted
            - If target in [nums[left]], nums[mid]) search left half; else search right half
        - Else → Right half sorted
            - If target in (nums[mid], nums[right]], search right half; else search left half
3. Return -1 if not found.

Complexity:
- Time: O(log n) (binary search)
- Space: O(1)