Strivers-A2Z-DSA-Sheet-main\02.Binary Search\1D Arrays\09.Search_in_rotated_sorted_array.cpp

```
/*
QUESTION:
There is an integer array nums sorted in ascending order (with distinct values).

Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums after the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums.

You must write an algorithm with O(log n) runtime complexity.

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
*/

/*
APPROACH:
We can use the binary search approach to find the target element in the rotated sorted array.
1. Initialize low = 0 and high = nums.size() - 1, where nums is the input array.
2. Perform binary search using the while loop until low <= high.
3. Calculate mid = low + (high - low) / 2.
4. If nums[mid] is equal to the target, return mid.
5. Check if the left part of the array (nums[low] to nums[mid]) is sorted or the right part (nums[mid] to nums[high]) is sorted.
   - If the left part is sorted:
     - If the target is within the range of nums[low] and nums[mid], update high = mid - 1.
     - Otherwise, update low = mid + 1.
   - If the right part is sorted:
     - If the target is within the range of nums[mid] and nums[high], update low = mid + 1.
     - Otherwise, update high = mid - 1.
6. If the target is not found after the while loop, return -1.

CODE:
*/

int search(vector<int>& nums, int target) {
    int low = 0, high = nums.size() - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (nums[mid] == target)
            return mid;
        if (nums[low] <= nums[mid]) {
            if (nums[low] <= target && target <= nums[mid])
                high = mid - 1;
```

```
48              else
49                  low = mid + 1;
50          } else {
51              if (nums[mid] <= target && target <= nums[high])
52                  low = mid + 1;
53              else
54                  high = mid - 1;
55          }
56      }
57      return -1;
58  }
59
60  // TIME COMPLEXITY: O(log n)
61  // SPACE COMPLEXITY: O(1)
62
```