33. search in rotated sorted array

// Iterative solution

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int leftIdx = 0;
        int rightIdx = nums.size() - 1;

        while (leftIdx <= rightIdx) {
            int midIdx = (leftIdx + rightIdx) / 2;
            if (nums[midIdx] == target) {
                return midIdx;
            }
            if (nums[leftIdx] <= nums[midIdx]) { // left is sorted
                if (target >= nums[leftIdx] && target < nums[midIdx]) {
                    rightIdx = midIdx - 1;
                } else {
                    leftIdx = midIdx + 1;
                }
            } else { // right is sorted
                if (target > nums[midIdx] && target <= nums[rightIdx]) {
                    leftIdx = midIdx + 1;
                } else {
                    rightIdx = midIdx - 1;
                }
            }
        }
        return -1;
    }
};
```

```
// Recursive solution
class Solution {
public:
    int shiftedBinarySearch(vector<int>&nums, int target, int leftIdx, int rightIdx) {
        if (leftIdx > rightIdx) return -1;
        int mid = (leftIdx + rightIdx) / 2;
        int leftNum = nums[leftIdx];
        int rightNum = nums[rightIdx];
        int midNum = nums[mid];
        if (midNum == target) {
            return mid;
        }
        if (leftNum <= midNum) { // left is sorted
            if (target >= leftNum && target < midNum) {
                return shiftedBinarySearch(nums, target, leftIdx, mid - 1);
            } else {
                return shiftedBinarySearch(nums, target, mid + 1, rightIdx);
            }
        } else { // right is sorted
            if (target > midNum && target <= rightNum) {
                return shiftedBinarySearch(nums, target, mid + 1, rightIdx);
            } else {
                return shiftedBinarySearch(nums, target, leftIdx, mid - 1);
            }
        }
    }

    int search(vector<int>& nums, int target) {
        return shiftedBinarySearch(nums, target, 0, nums.size() - 1);
    }
```

};

34. find first and last position of element in sorted array

```
public class FindFirstAndLastPositionOfElementInSortedArray {

    public int[] searchRange(int[] nums, int target) {
        return new int[]{findFirstOccurrence(nums, target),
findLastOccurrence(nums, target)};
    }

    private int findFirstOccurrence(int[] nums, int target) {
        // Left and right pointers
        int left = 0;
        int right = nums.length - 1;
        // Index of first occurrence
        int firstOccurrence = -1;
        // Loop until the two pointers meet
        while (left <= right) {
            // Middle index
            int middle = left + (right - left) / 2;
            // Check if we have found the value
            if (nums[middle] == target) {
                firstOccurrence = middle;
                right = middle - 1;
            }
            // If the target is less than the element
            // at the middle index
            else if (target < nums[middle]) {
                right = middle - 1;
            }
            // If the target is greater than the element
            // at the middle index
            else {
                left = middle + 1;
            }
        }
        return firstOccurrence;
    }

    private int findLastOccurrence(int[] nums, int target) {
        // Left and right pointers
        int left = 0;
        int right = nums.length - 1;
        // Index of first occurrence
        int lastOccurrence = -1;
        // Loop until the two pointers meet
        while (left <= right) {
            // Middle index
            int middle = left + (right - left) / 2;
            // Check if we have found the value
            if (nums[middle] == target) {
                lastOccurrence = middle;
                left = middle + 1;
            }
            // If the target is less than the element
            // at the middle index
            else if (target < nums[middle]) {
                right = middle - 1;
            }
            // If the target is greater than the element
            // at the middle index
            else {
                left = middle + 1;
            }
        }
        return lastOccurrence;
    }
}
```