33. Search Rotated Sorted Array

Hard

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
```

Example 3:

```
Input: nums = [1], target = 0
Output: -1
```

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums) - 1

        while left <= right:
            mid = left + ((right - left) // 2)
            if nums[mid] == target:
                return mid
            elif nums[left] > nums[mid]:
                if target < nums[mid] or target > nums[right]:
                    right = mid - 1
                else:
                    left = mid + 1
            else:
                if target > nums[mid] or target < nums[left]:
                    left = mid + 1
                else:
                    right = mid - 1
        return -1
```

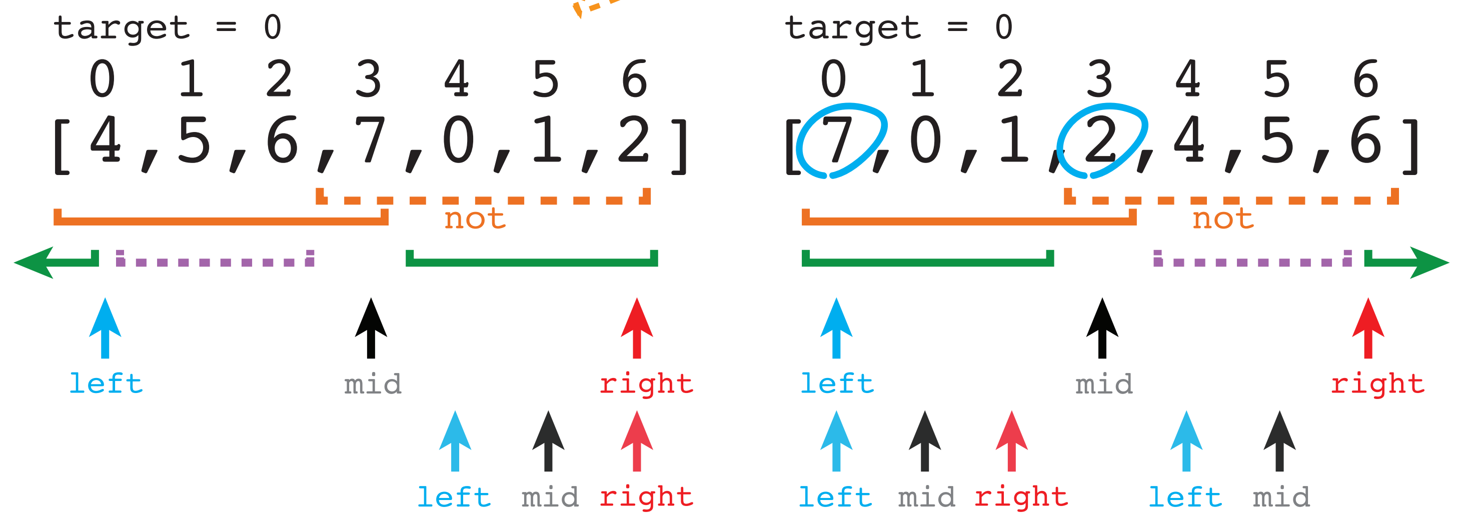
1. Set up a typical binary search and define the mid, left, and right values.
2. Identify the location of the lowest number by comparing the middle number with the left or right numbers:

```
        nums[left] > nums[mid] ## lowest number is on left of mid

        ## ALTERNATIVELY
        nums[right] < nums[mid] ## lowest number is on right of mid
```

3. Based on which half contains the lowest number, further determine how left or right pointers should be shifted, by checking:

```
        target < nums[mid] or target > nums[right]
            ## target number is on left of mid, therefore
            ## move the pointer towards the left (right = mid - 1)
```



target = 0
0 1 2 3 4 5 6
[4,5,6,7,0,1,2]
not
left
mid
right
left
mid
right
target = 0
0 1 2 3 4 5 6
[7,0,1,2,4,5,6]
not
left
mid
right
left
mid
right
left
mid