33. Search in Rotated Sorted Array

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
```

Constraints:

- `1 <= nums.length <= 5000`
- `-10<sup>4</sup> <= nums[i] <= 10<sup>4</sup>`
- All values of `nums` are **unique**.
- `nums` is an ascending array that is possibly rotated.
- `-10<sup>4</sup> <= target <= 10<sup>4</sup>`

```
def search(self, nums: List[int], target: int) -> int:

        lo = 0
        hi = len(nums)-1
```

```python
        while lo<=hi:
            mid = (lo+hi)//2
            if nums[mid]==target:
                return mid
            elif nums[lo]<=nums[mid]:
                if target>=nums[lo] and target<nums[mid]:
                    hi = mid-1
                else:
                    lo = mid+1
            elif nums[hi]>=nums[mid]:
                if target>nums[mid] and target<=nums[hi]:
                    lo = mid+1
                else:
                    hi = mid-1
        return -1
```

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        if len(nums)==1:
            if nums[0]==target:
                return 0
            else:
                return -1
        minIdx = self.searchMin(nums)
        # print(minIdx)
        if nums[minIdx]==target:
            return minIdx
        if minIdx==0:
            return self.binarySearch(nums,0,len(nums)-1,target)
        left=self.binarySearch(nums,0,minIdx-1,target)
        right=self.binarySearch(nums,minIdx,len(nums)-1,target)
        # print(left,right)
        if left==-1 and right==-1:
            return -1
        else:
            return left if right==-1 else right


    def binarySearch(self,arr,low,high,target):
        if low==high and arr[low]==target:
            return low
        while low<=high:
            mid = low+(high-low)//2
            if arr[mid]==target:
                return mid
            elif arr[mid]>target:
                high = mid-1
            elif arr[mid]<target:
                low = mid+1
        return -1


    def searchMin(self,arr):
        start = 0
        end = len(arr)-1
        n = len(arr)

        while start <= end:
            if arr[start]<arr[end]:
                return start
            mid = start + (end - start) // 2
            prev = (mid+n-1)%n
            nextt = (mid+1)%n
            if arr[mid] < arr[nextt] and arr[mid] < arr[prev]:
                return mid
            elif arr[mid] <= arr[end]:
                end = mid - 1
            elif arr[mid] >= arr[start]:
                start = mid + 1
```