33. Search in Rotated Sorted Array (Medium)

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`
- `104 <= nums[i] <= 104`
- All values of `nums` are **unique**.

- `nums` is an ascending array that is possibly rotated.
- `104 <= target <= 104`

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int l=0,r=nums.size()-1,m;
        while(l<r){
            m = (l+r)/2;
            if(nums[m] > nums[r]){
                if(target > nums[r] && target <= nums[m]){
                    r = m;
                }
                else{
                    l=m+1;
                }
            }
            else{
                if(target > nums[m] && target <= nums[r]){
                    l=m+1;
                }
                else{
                    r=m;
                }
            }
        }
        return (target == nums[l])? l:-1;
    }
};
```