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`
- `-10^4 <= nums[i] <= 10^4`
- All values of `nums` are **unique**.
- `nums` is an ascending array that is possibly rotated.
- `-10^4 <= target <= 10^4`

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

Find Minimum in Rotated Sorted Array

Medium

Suppose an array of length `n` sorted in ascending order is **rotated** between `1` and `n` times. For example, the array `nums = [0,1,2,4,5,6,7]` might become:

- `[4,5,6,7,0,1,2]` if it was rotated `4` times.
- `[0,1,2,4,5,6,7]` if it was rotated `7` times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in `O(log n) time.`

Example 1:

```
Input: nums = [3,4,5,1,2]

Output: 1
```

Explanation: The original array was [1,2,3,4,5] rotated 3 times.

Example 2:

Input: nums = [4,5,6,7,0,1,2]

Output: 0

Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.

Example 3:

Input: nums = [11,13,15,17]

Output: 11

Explanation: The original array was [11,13,15,17] and it was rotated 4 times.

Constraints:

- `n == nums.length`
- `1 <= n <= 5000`
- `-5000 <= nums[i] <= 5000`
- All the integers of `nums` are **unique**.
- `nums` is sorted and rotated between `1` and `n` times.

```
class Solution {
public:
    int findMin(vector<int>& a) {
        int l=0, r=a.size()-1, mid;
        while (r-l>1) {
            mid=(l+r)/2;
            if (a[mid]>a[l] and a[mid]<a[r]) return a[l];
            else if (a[mid]>a[l]) l=mid;
            else r=mid;
        }
        if (a[l]<a[r]) return a[l];
        else return a[r];
    }
};
```