Search in Rotated Sorted Array

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

Solution 1

```
class Solution {
public:
    int search(int A[], int n, int target) {
        int lo=0,hi=n-1;
        // find the index of the smallest value using binary search.
        // Loop will terminate since mid < hi, and lo or hi will shrink by at leas
t 1.
        // Proof by contradiction that mid < hi: if mid==hi, then lo==hi and loop
would have been terminated.
        while(lo<hi){
            int mid=(lo+hi)/2;
            if(A[mid]>A[hi]) lo=mid+1;
            else hi=mid;
        }
        // lo==hi is the index of the smallest value and also the number of place
s rotated.
        int rot=lo;
        lo=0;hi=n-1;
        // The usual binary search and accounting for rotation.
        while(lo<=hi){
            int mid=(lo+hi)/2;
            int realmid=(mid+rot)%n;
            if(A[realmid]==target)return realmid;
            if(A[realmid]<target)lo=mid+1;
            else hi=mid-1;
        }
        return -1;
    }
};
```

written by lucastan original link here

Solution 2

```
public class Solution {
public int search(int[] A, int target) {
    int lo = 0;
    int hi = A.length - 1;
    while (lo < hi) {
        int mid = (lo + hi) / 2;
        if (A[mid] == target) return mid;

        if (A[lo] <= A[mid]) {
            if (target >= A[lo] && target < A[mid]) {
                hi = mid - 1;
            } else {
                lo = mid + 1;
            }
        } else {
            if (target > A[mid] && target <= A[hi]) {
                lo = mid + 1;
            } else {
                hi = mid - 1;
            }
        }
    }
    return A[lo] == target ? lo : -1;
}
```

}

written by jerry13466 original link here

Solution 3

The idea is that when rotating the array, there must be one half of the array that is still in sorted order. For example, 6 7 1 2 3 4 5, the order is disrupted from the point between 7 and 1. So when doing binary search, we can make a judgement that which part is ordered and whether the target is in that range, if yes, continue the search in that half, if not continue in the other half.

```
public class Solution {
    public int search(int[] nums, int target) {
        int start = 0;
        int end = nums.length - 1;
        while (start <= end){
            int mid = (start + end) / 2;
            if (nums[mid] == target)
                return mid;

            if (nums[start] <= nums[mid]){
                 if (target < nums[mid] && target >= nums[start])
                    end = mid - 1;
                 else
                    start = mid + 1;
            }

            if (nums[mid] <= nums[end]){
                if (target > nums[mid] && target <= nums[end])
                    start = mid + 1;
                 else
                    end = mid - 1;
            }
        }
        return -1;
    }
}
```

written by flyinghx61 original link here