

# o81. Search in Rotated Sorted Array II

## o81 Search in Rotated Sorted Array II

- Binary Search+Array

### Description

Follow up for "Search in Rotated Sorted Array":

What if *duplicates* are allowed?

Would this affect the run-time complexity? How and why?

Suppose an array sorted in ascending order 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`).

Write a function to determine if a given target is in the array.

The array may contain duplicates.

### 1. Thought line

### 2. Binary Search+Array

```
1 class Solution {
2 private:
3     void binarySearch(vector<int>& nums, int target, int st, int ed, bool& res){
4         // finish process condition
5         if (st>ed) return;
6         if (target<nums[st]||target>nums[ed]) return;
7
8         // no target
9         if (st==ed && nums[st]!=target) return;
10
11        // find target
12        if (st==ed && nums[st]==target) res = true;
13
14        // keep finding process
15        else{
16            int midSt = (st+ed)/2, midEd = (st+ed)/2;
17            if (nums[midSt]==target) {
18                res = true;
19                return;
20            }
21            // find the bottle and top elements as same value of nums[mid];
22            while (midSt-1>=0 && nums[midSt]==nums[midSt-1])
23                --midSt;
24            while (midEd + 1<=nums.size()-1 && nums[midEd]==nums[midEd+1])
25                ++midEd;
26            if (target<nums[midSt])
27                binarySearch(nums, target, st, midSt-1, res);
28            else
29                binarySearch(nums, target, midEd+1, ed, res);
30        }
31    }
```

```

31     }
32
33 public:
34     bool search(vector<int>& nums, int target) {
35         bool res = false;
36         int pivot = 0;
37         if (nums.empty()) return false;
38         // find pivot
39         for (int i = 1; !nums.empty() && i<=nums.size()-1; ++i){
40             if (nums[i-1]>nums[i]){
41                 pivot = i;
42                 break;
43             }
44         }
45         // process binary search on left half
46         binarySearch(nums, target, 0, pivot-1, res);
47         // process binary search on right half
48         binarySearch(nums, target, pivot, nums.size()-1, res);
49         return res;
50     }
51 };

```