My 8ms C++ solution (o(logn) on average, o(n) worst case)

The idea is the same as the previous one without duplicates

1) everytime **check** **if** targe == nums[**mid**], **if** so, we find it.

2) otherwise, we **check** **if** the **first** half **is** **in** **order** (i.e. nums[**left**]<=nums[**mid**])

**and** **if** so, **go** **to** step 3), otherwise, the **second** half **is** **in** **order**, **go** **to** step 4)

3) **check** **if** target **in** the **range** **of** [**left**, **mid**-1] (i.e. nums[**left**]<=target < nums[**mid**]), **if** so, **do** **search** **in** the **first** half, i.e. **right** = **mid**-1; otherwise, search in the second half left = mid+1;

4) **check** **if** target **in** the **range** **of** [**mid**+1, **right**] (i.e. nums[**mid**]<target <= nums[**right**]), **if** so, **do** **search** **in** the **second** half, i.e. **left** = **mid**+1; otherwise search in the first half right = mid-1;

The only difference is that due to the existence of duplicates, we can have nums[left] == nums[mid] and in that case, the first half could be out of order (i.e. NOT in the ascending order, e.g. [3 1 2 3 3 3 3]) and we have to deal this case separately. In that case, it is guaranteed that nums[right] also equals to nums[mid], so what we can do is to check if nums[mid]== nums[left] == nums[right] before the original logic, and if so, we can move left and right both towards the middle by 1. and repeat.

**class** **Solution** {

**public**:

bool search(vector<int>& nums, int target) {

int **left** = 0, **right** = nums.size()-1, mid;

**while**(**left**<=**right**)

{

mid = (**left** + **right**) >> 1;

**if**(nums[mid] == target) **return** true;

// the only difference from the first one, trickly case, just updat left and right

**if**( (nums[**left**] == nums[mid]) && (nums[**right**] == nums[mid]) ) {++**left**; --**right**;}

**else** **if**(nums[**left**] <= nums[mid])

{

**if**( (nums[**left**]<=target) && (nums[mid] > target) ) **right** = mid-1;

**else** **left** = mid + 1;

}

**else**

{

**if**((nums[mid] < target) && (nums[**right**] >= target) ) **left** = mid+1;

**else** **right** = mid-1;

}

}

**return** false;

}

};