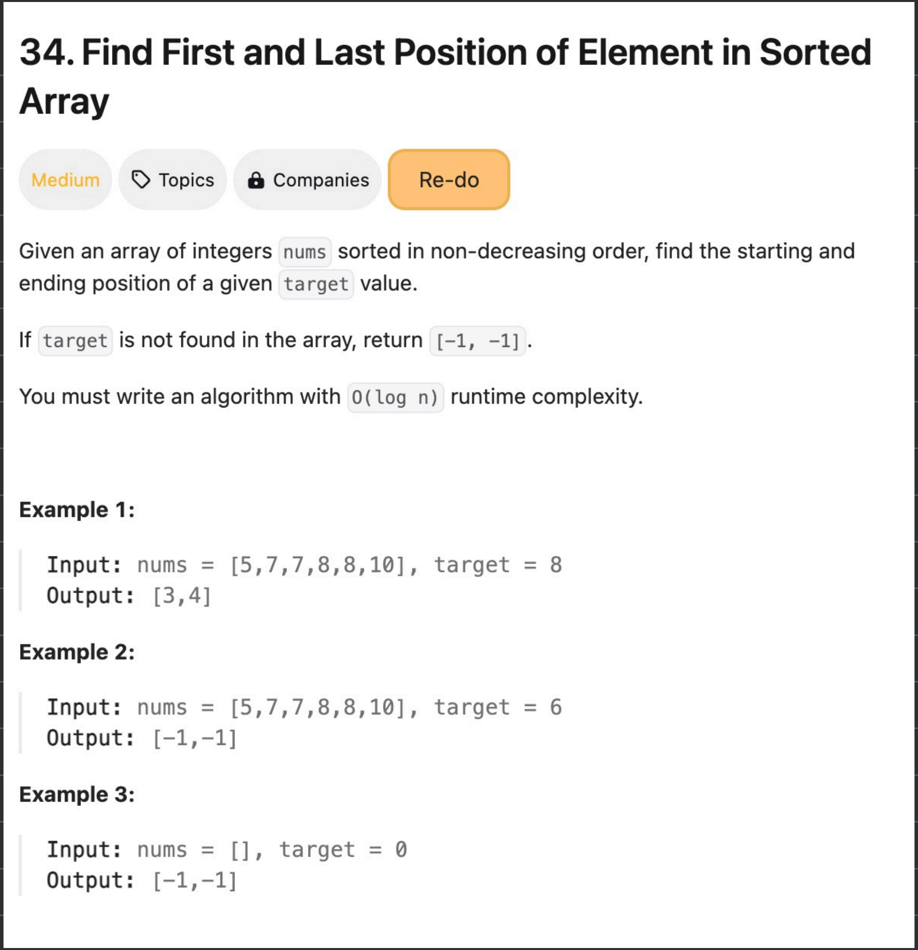
In the end, it's YOU vs YOU. Be the best today, to defeat your yesterday!!



# 34. Find First and Last Position of Element in Sorted Array

Medium | Topics | Companies | Re-do

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

You must write an algorithm with `O(log n)` runtime complexity.

**Example 1:**

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

**Example 2:**

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

**Example 3:**

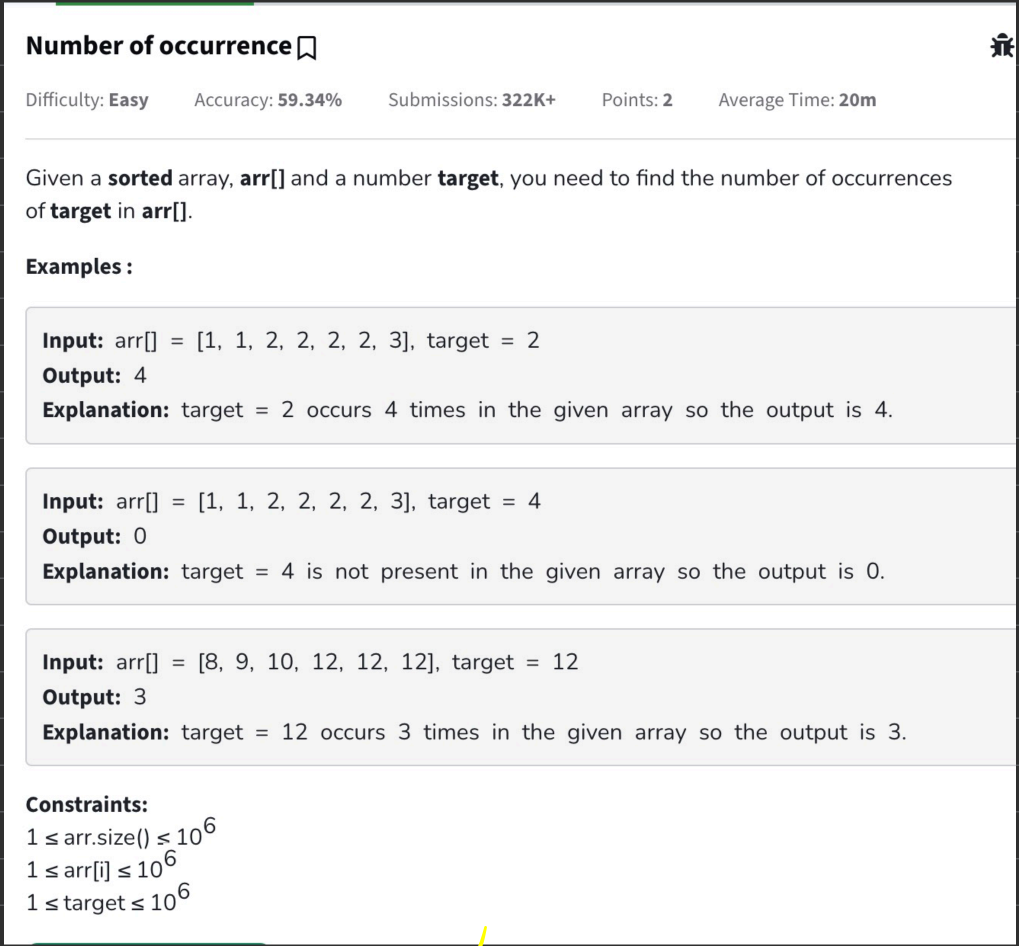
```
Input: nums = [], target = 0
Output: [-1,-1]
```

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
|  | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 6 |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

$x=3$

```
int startPosition ( arr, n, x )
{
    s=0 , e=n-1
    ans = -1
    while ( s ≤ e)
    {
        m= (s+e)/2
        if (a(mid) ==x)
        {
            ans= mid
            e= mid -1
        }
        else if ( a(mid) < x)
            s= mid+1
        else end=mid-1
```

```
int endPosition ( arr, n, x )
{
    s=0 , e=n-1
    ans = -1
    while ( s ≤e)
    {
        m= (s+e)/2
        if (a(mid) ==x)
        {
            ans= mid
            s=mid+1
        }
        else if ( a(mid) < x)
            s= mid+1
        else end=mid-1
}
```



## Number of occurrence

Difficulty: **Easy** Accuracy: **59.34%** Submissions: **322K+** Points: **2** Average Time: **20m**

Given a **sorted** array, **arr[]** and a number **target**, you need to find the number of occurrences of **target** in **arr[]**.

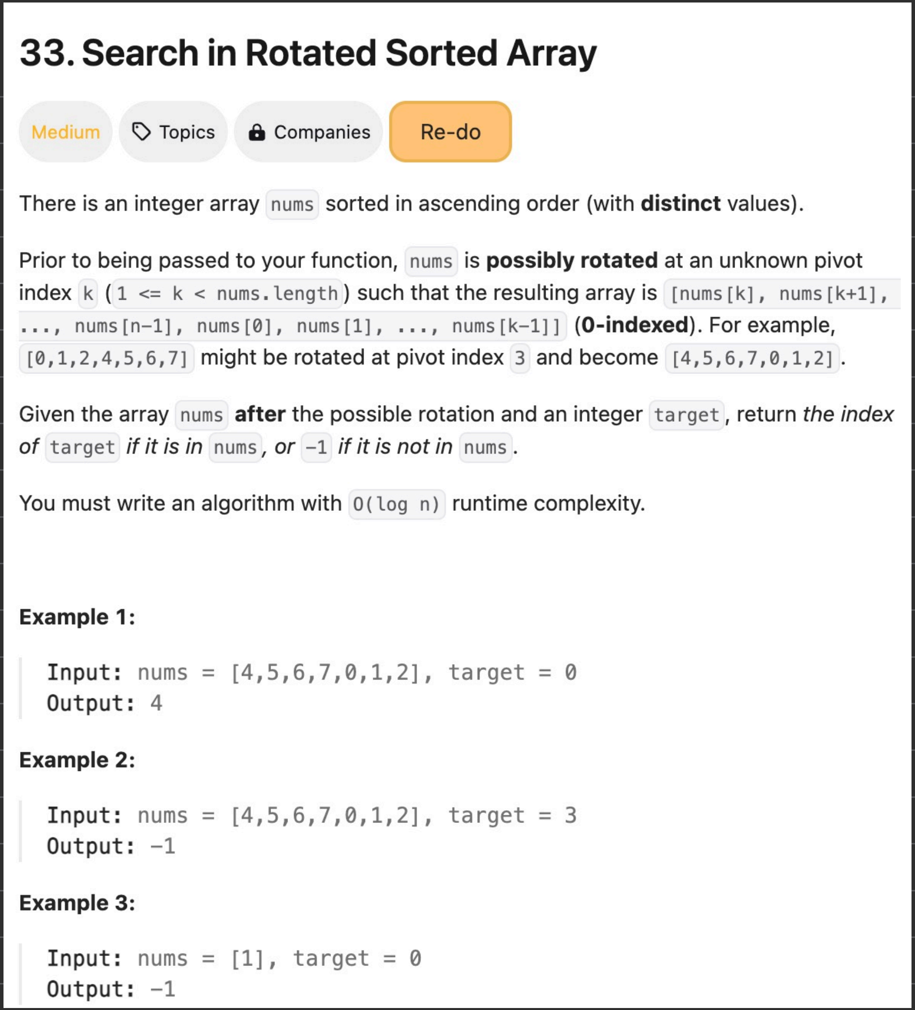
**Examples :**

```
Input: arr[] = [1, 1, 2, 2, 2, 2, 3], target = 2
Output: 4
Explanation: target = 2 occurs 4 times in the given array so the output is 4.
```

```
Input: arr[] = [1, 1, 2, 2, 2, 2, 3], target = 4
Output: 0
Explanation: target = 4 is not present in the given array so the output is 0.
```

```
Input: arr[] = [8, 9, 10, 12, 12, 12], target = 12
Output: 3
Explanation: target = 12 occurs 3 times in the given array so the output is 3.
```

**Constraints:**
$1 \le arr.size() \le 10^6$
$1 \le arr[i] \le 10^6$
$1 \le target \le 10^6$



# 33. Search in Rotated Sorted Array

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:**

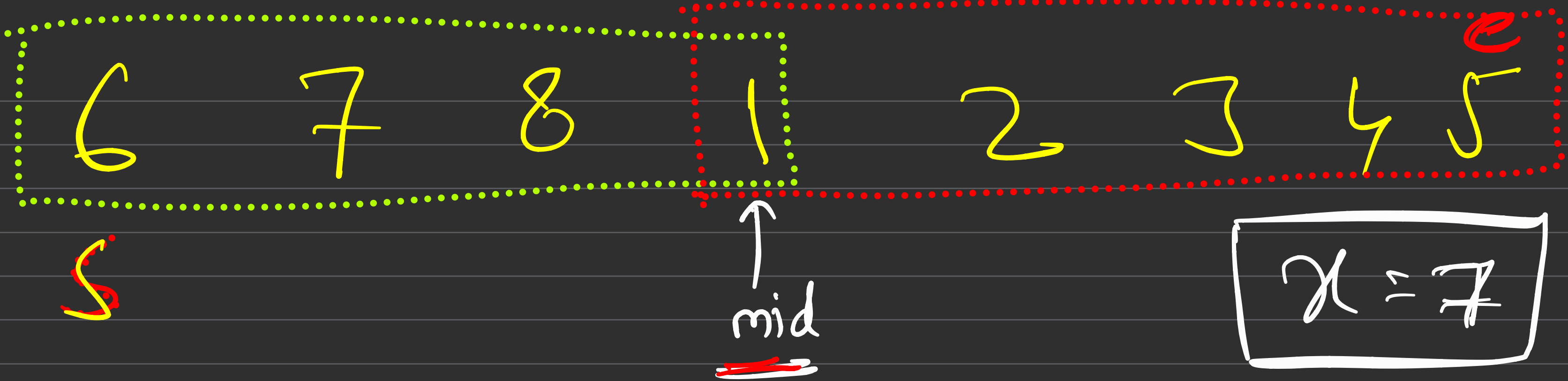
```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

**Example 2:**

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

**Example 3:**

```
Input: nums = [1], target = 0
Output: -1
```



$\boxed{x = 7}$

$\boxed{-\ \text{sorted}}$

$a(m) \le a(e)$

① mid find

② $a(mid) == x \longrightarrow$ return mid

③ Check which part is sorted $\boxed{x}$ target

1) left: $a[s] \leq a[mid]$

2) right: $a(m) \leq a(e)$

```
if (a[s] ≤ a[mid])                    // left sorted
{
    if (a[s] ≤ x && x ≤ a[mid])
        end = mid-1

    else
        start = mid+1

}
else {
    if (a[m] ≤ x && x ≤ a[end])
        s = mid+1
    else
        e = mid-1
}
```