852. Peak Index in a Mountain Array

Medium | Topics | Companies

You are given an integer **mountain** array `arr` of length `n` where the values increase to a **peak element** and then decrease.

Return the index of the peak element.

Your task is to solve it in `O(log(n))` time complexity.

Example 1:

> **Input:** `arr = [0,1,0]`
>
> **Output:** `1`

Example 2:

> **Input:** `arr = [0,2,1,0]`
>
> **Output:** `1`

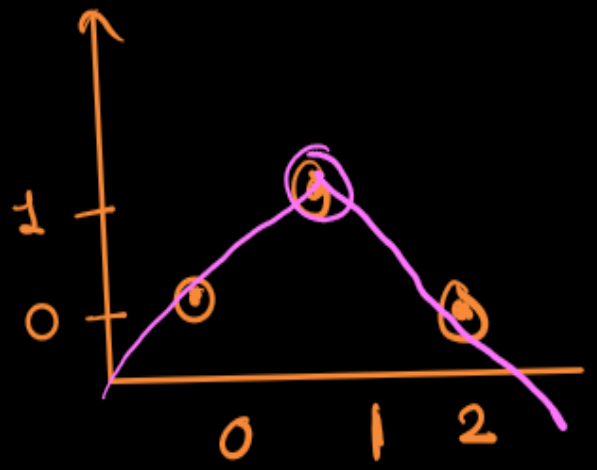
Example 3:

> **Input:** `arr = [0,10,5,2]`
>
> **Output:** `1`

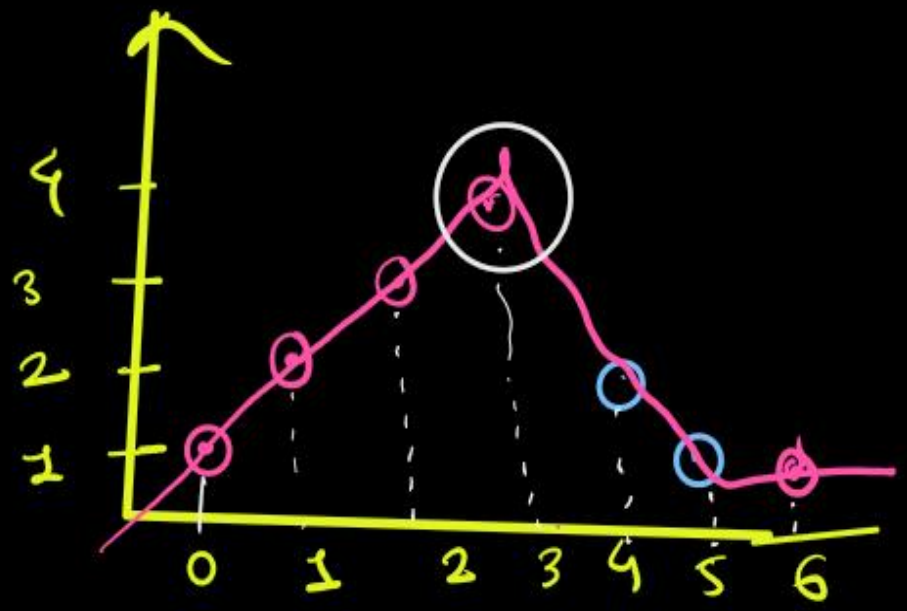
Constraints:

- `3 <= arr.length <= 10^5`
- `0 <= arr[i] <= 10^6`
- `arr` is **guaranteed** to be a mountain array.

{0, 1, 0}



{1, 2, 3, 4, 2, 1, 1}

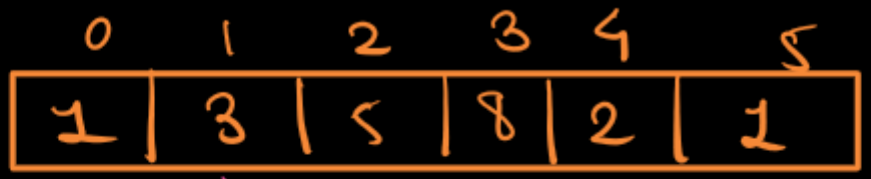


arr = {0 2, 1, 0}

0 1 2 3

o/p → 1.

Approach 2



2<8 → 8; i+1

For(i=2; i<n; i++){

if(arr[i] < arr[i-1])return i-1

}

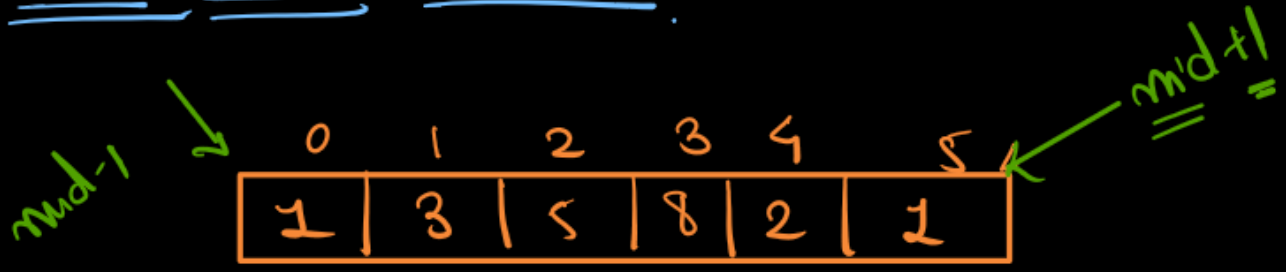
Time Complexity

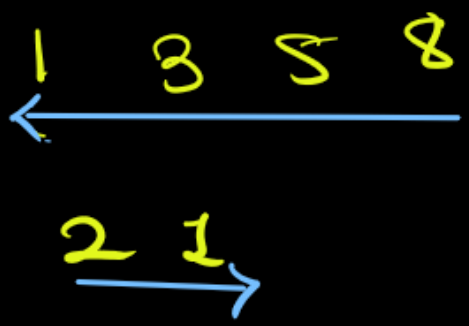
O(n)

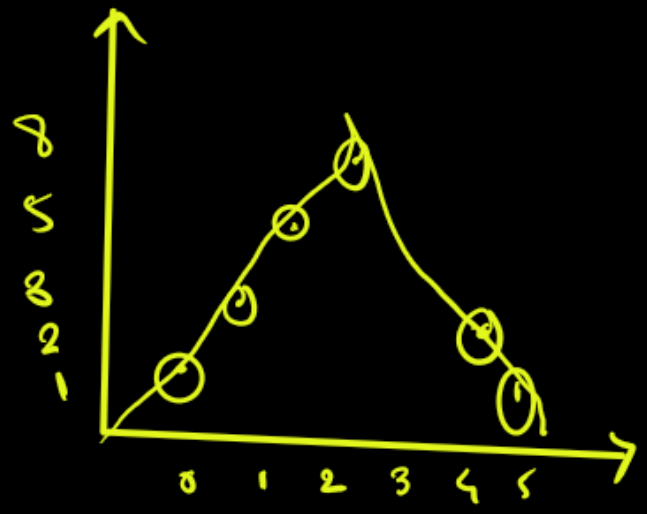
SC = O(1)

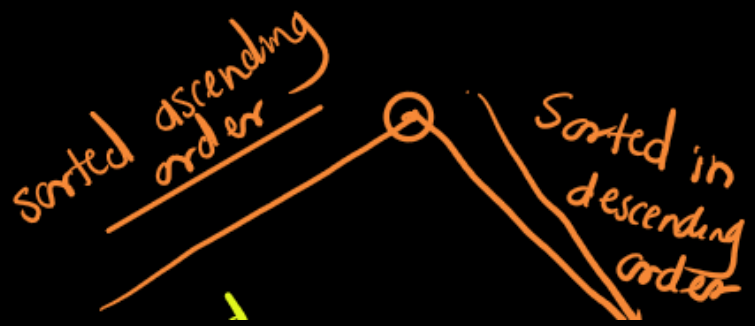
```java
class Solution {
    public int peakIndexInMountainArray(int[] arr) {
        for(int i = 1; i < arr.length; i++){
            if(arr[i] < arr[i-1])return i- 1;
        }
        return -1;
    }
}
```

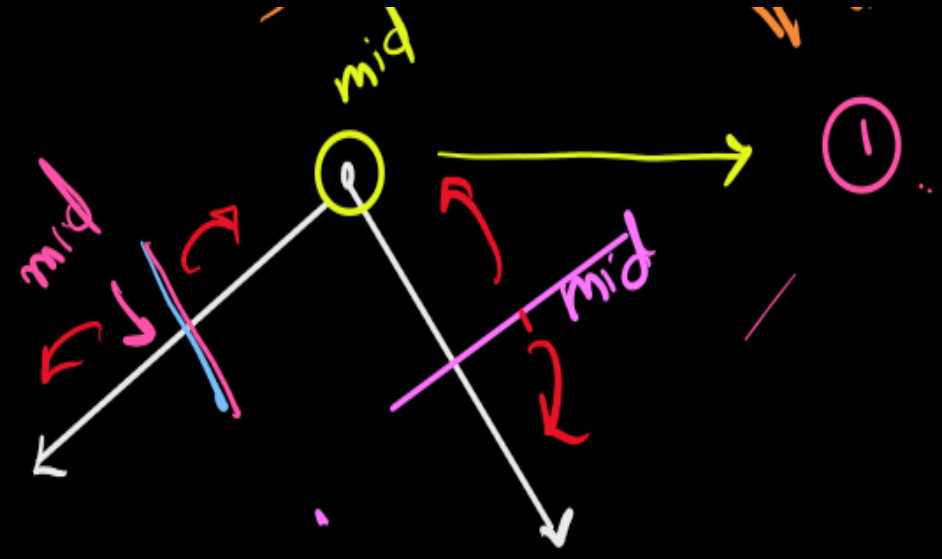
Approach 2


mid-1
0 1 2 3 4 5
1 3 5 8 2 1
mid+1

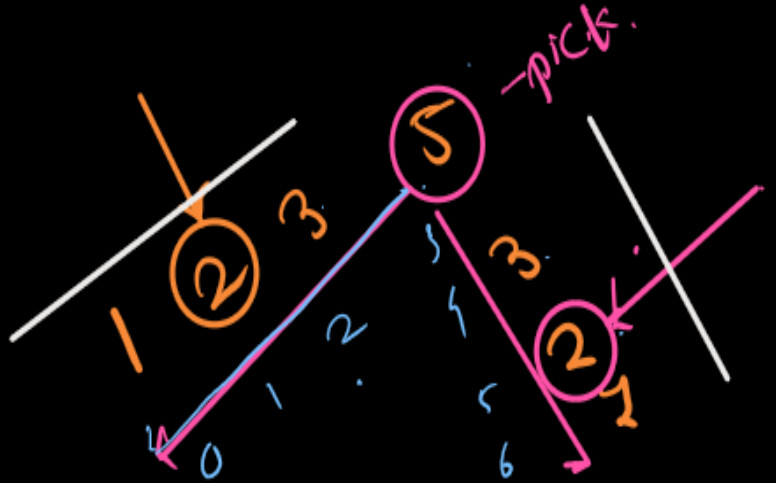

1 3 5 8
2 1


8
5
2
1
0 1 2 3 4 5


Sorted ascending order
Sorted in descending order



① if (arr[mid] > arr[mid+1] && arr[mid] > arr[mid-1]) {
return mid;
}



② left :

```
if (arr[mid] < arr[mid+1]) {
    s = mid + 1.
}
```

③ Right :

```
if (arr[mid] > arr[mid+1]) {
    e = mid - 1
}
```

Java Auto

```java
class Solution {
    public int peakIndexInMountainArray(int[] arr) {
        int s = 0;
        int e = arr.length-1;
        while (s <= e) {
            int mid = s + (e - s) / 2;
            if (mid - 1 >= 0 && mid + 1 < arr.length && arr[mid] > arr[mid + 1] && arr[mid] > arr[mid - 1]) {
                return mid;
            }
            else if(mid + 1 < arr.length && arr[mid] < arr[mid+1])s = mid + 1 ;
            else e = mid - 1;
        }
        return -1;
    }
}
```

33. Search in Rotated Sorted Array

Solved

Medium | Topics | Companies

There is an integer array `nums` sorted in ascending order (with **distinct** values).

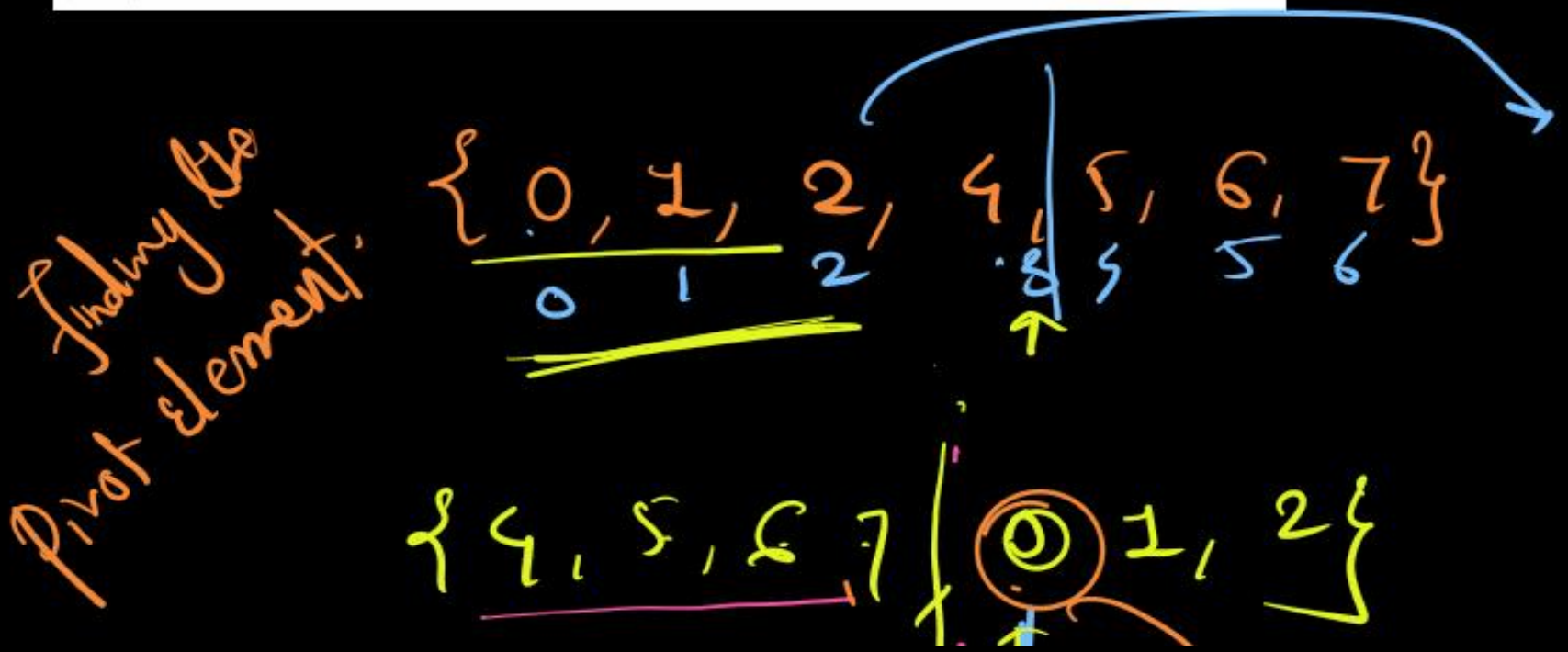
Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

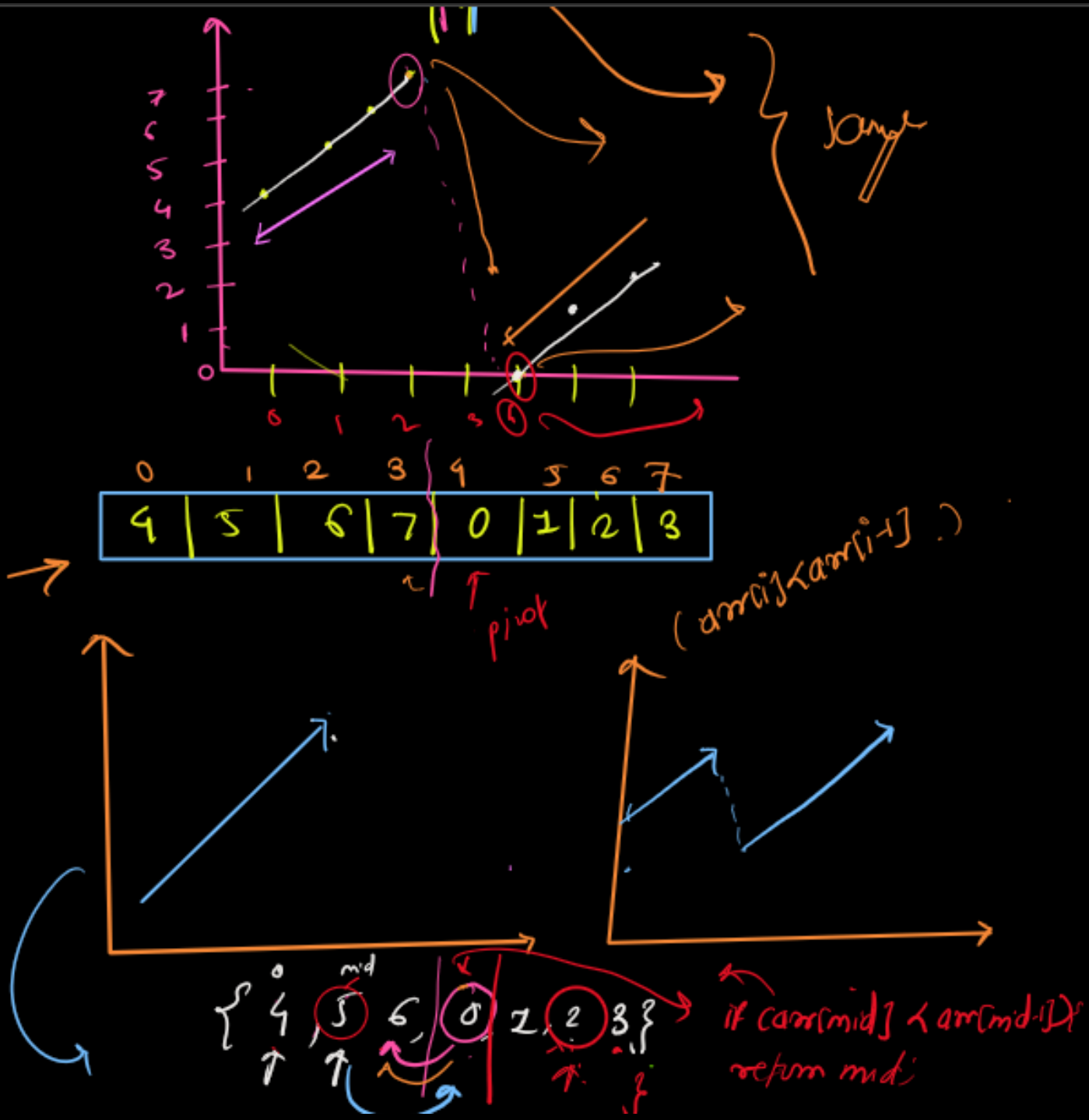
Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, *or* `-1` *if it is not in* `nums`.

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

Example 1:

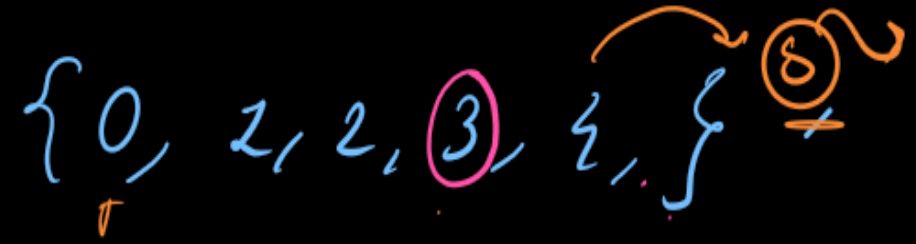
```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```




larger
0 1 2 3 4 5 6 7
4 5 6 7 0 1 2 3
pivot
(arr[i]<arr[i-1] :)
mid
if (arr[mid] < arr[mid-1])
return mid;

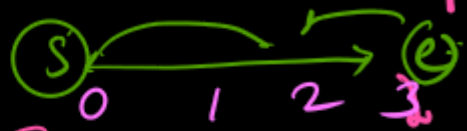
```
while(s<=e){
    mid=(s+e)/2;
    if(mid==t){
        return mid;
    }
    else if(mid<t)
        {s=mid+1;
    }
    else{
        e=mid-1;
    }
```

```
if(arr[0]<arr[mid]){
    s=mid+1;
}
else{
    e=mid-1;
}
```



target = 7

{4, 5, 6, 7 | 0, 1, 2}
0 1 2 3 4 5 6

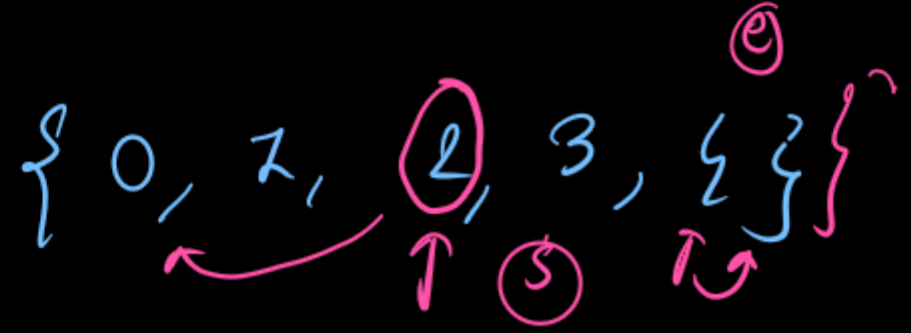


4 <= t
7 <= t

→ {4, 5, 6, 7} → arr 1.

0 <= t
2 > t

→ {0, 1, 2} = arr 2.
4 5 6



```java
class Solution {
    static int findPivot(int arr[]){
        int s = 0;
        int e = arr.length-1;
        while(s <= e){
            int mid = (s + e )/ 2;
            if(mid - 1 >= 0 && arr[mid] < arr[mid-1])return mid;
            if(arr[0] <= arr[mid])s = mid + 1;
            else e = mid - 1;
        }
        if(s >= arr.length)return 0;
        else return arr.length-1;

    }
    static int bs(int [] arr, int start, int end, int key){
        while (start <= end) {
            int mid = (start + end) / 2;  // start + (end - start)/2;
            if (arr[mid] == key) {
                return mid;
            } else if (arr[mid] < key) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        return -1;
    }
    public int search(int[] nums, int target) {
        int pivotIndex = findPivot(nums);
        if(pivotIndex - 1 >= 0 && nums[0] <= target && nums[pivotIndex-1] >= target){
            return bs(nums, 0, pivotIndex - 1 , target);
        }
        else{
            return bs(nums, pivotIndex, nums.length-1, target);
        }
    }
}
```