Binary search Basic

28 December 2022 08:05

que 1:-

Binary Search

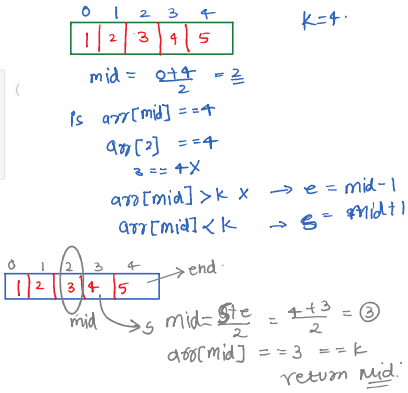
Basic Accuracy: 44.32% Submissions: 272K+ Points: 1

Given a sorted array of size **N** and an integer **K**, find the position at which **K** is present in the array using binary search.

Example 1:

Input:
N = 5
arr[] = {1 2 3 4 5}
K = 4
Output: 3
Explanation: 4 appears at index 3.

Screen clipping taken: 28-12-2022 08:07



e.

```
// User function Template for Java

class Solution {
    int binarysearch(int arr[], int n, int k) {
        // code here
        int start=0;
        int end= n-1;
        while(start<=end){
            int mid = (start + end)/2;
            if(arr[mid] == k){
                return mid;
            }
            if(arr[mid] > k){
                end = mid -1;
            }
            else{
                start = mid +1;
            }
        }
        return -1;
    }
}
```

Screen clipping taken: 28-12-2022 08:12

que 2 :- floor in a sorted Array :-

Floor in a Sorted Array

Easy Accuracy: 33.75% Submissions: 117K+ Points: 2

Given a sorted array **arr**[] of size **N** without duplicates, and given a value **x**. Floor of x is defined as the largest element **K** in arr[] such that K is smaller than or equal to x. Find the index of K(0-based indexing).

Example 1:

Input:
N = 7, x = 0
arr[] = {1,2,8,10,11,12,19}
Output: -1
Explanation: No element less than 0 is found. So output is "**-1**".

floor means greatest element smaller than x

Example 2:

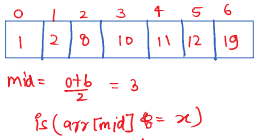
Input:
N = 7, x = 5
arr[] = {1,2,8,10,11,12,19}
Output: 1
Explanation: Largest Number less than 5 is **2 (i.e K = 2)**, whose index is **1**(0-based indexing).

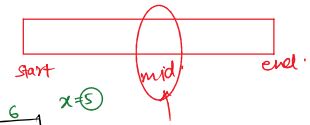
Your Task:
The task is to complete the function **findFloor**() which returns an integer denoting the index value of K or return **-1** if there isn't any such number.

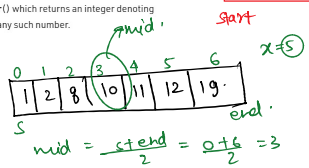
Expected Time Complexity: O(log N).
Expected Auxiliary Space: O(1).

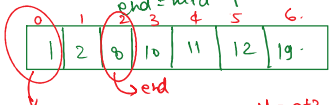
Constraints:
$1 \le N \le 10^7$

Screen clipping taken: 28-12-2022 08:14










```
class Solution{

    // Function to find floor of x
    // arr: input array
    // n is the size of array
    static int findFloor(long arr[], int n, long x)
    {
        int  ans =-1;
        int  start =0;
        int end =n-1;
        while(start <= end){
            int  mid = (start + end)/2;
            if(arr[mid] == x){
                return mid;
            }
            if(arr[mid] > x){
```

```java
class Solution{

    // Function to find floor of x
    // arr: input array
    // n is the size of array
    static int findFloor(long arr[], int n, long x)
    {
        int  ans =-1;
        int  start =0;
        int end =n-1;
        while(start <= end){
            int  mid = (start + end)/2;
            if(arr[mid] == x){
                return mid;
            }
            if(arr[mid] > x){
                end = mid -1;
            }
            else{
                ans = mid;
                start = mid+1;
            }
        }
        return ans;
    }

}
```

Screen clipping taken: 28-12-2022 08:35

que 3:- floor and ceil.

Ceil The Floor

Basic Accuracy: 43.76% Submissions: 12K+ Points: 1

Given an unsorted array **Arr[]** of **N** integers and an integer **X**, find floor and ceiling of **X** in **Arr[0..N-1]**.

Floor of **X** is the largest element which is smaller than or equal to **X**. Floor of **X** doesn't exist if **X** is smaller than smallest element of **Arr[]**.

Ceil of **X** is the smallest element which is greater than or equal to **X**. Ceil of **X** doesn't exist if **X** is greater than greatest element of **Arr[]**.

Example 1:

Input:
N = 8, X = 7
Arr[] = {5, 6, 8, 9, 6, 5, 5, 6}
Output: 6 8
Explanation:
Floor of 7 is 6 and ceil of 7 is 8.

Screen clipping taken: 28-12-2022 08:36

Example 2:

Input:
N = 8, X = 10
Arr[] = {5, 6, 8, 9, 6, 5, 5, 6}
Output: 9 -1
Explanation:
Floor of 10 is 9 but ceil of 10 is not possible.

Your Task:
You don't need to read input or print anything. Your task is to complete the function **getFloorAndCeil()** which takes the array of integers **arr, n** and **x** as parameters and returns a pair of integers denoting the answer. Return **-1** if floor or ceil is not present.

Expected Time Complexity: O(N)
Expected Auxiliary Space: O(1)

Screen clipping taken: 28-12-2022 08:36

N=8, X=7

0	1	2	3	4	5	6	7
5	6	8	9	6	5	5	6

mid = 0+7/2 ⇒ 3

0	1	2	3	4	5	6	7
5	6	8	9	6	5	5	6

↑mid

if(arr[mid] == x)
return mid.
if(arr[mid] > x)
{ ans_ceil = mid
end = mid-1
}
if(arr[mid] < x)
{ ans_floor = mid
start = mid+1;

```java
class Solve {
    Pair getFloorAndCeil(int[] arr, int n, int x) {
        // code here
        Pair pair = new Pair(-1, -1);
        Arrays.sort(arr);
        int floor = -1;
        int ceil = -1;
        int start = 0;
        int end = n-1;
        while(start <= end){
            int mid = (start + end)/2;
            if(arr[mid] == x){
                floor =  arr[mid];
                ceil = arr[mid];
                break;
            }
            if(arr[mid] < x){
                floor = arr[mid];
                start = mid +1;
            }
            else if(arr[mid] > x){
                ceil  = arr[mid];
                end = mid -1;
            }
        }
        pair.floor = floor;
        pair.ceil = ceil;

        return pair;
    }
```

Screen clipping taken: 28-12-2022 08:48

que 4:- search insert position of k in a sorted Array :-

Search insert position of K in a sorted array

Easy Accuracy: 38.99% Submissions: 32K+ Points: 2

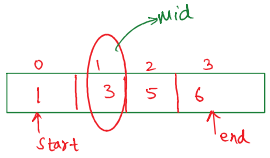
Given a sorted array **Arr[]** (0-index based) consisting of **N** distinct integers and an integer **k**, the task is to find the index of k, if its present in the array **Arr[]**. Otherwise, find the index where **k** must be inserted to keep the array sorted.

Example 1:

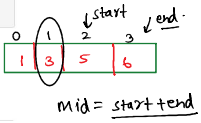
Input:
N = 4
Arr = {1, 3, 5, 6}
k = 5
Output: 2
Explanation: Since 5 is found at index 2 as Arr[2] = 5, the output is 2.

Screen clipping taken: 28-12-2022 08:54

Example 2:



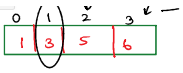
mid = 0+3/2 = 1



mid = start+end

if(arr[mid] == k)
return mid;
if(arr[mid] > k)
s = Mid+1
else e = mid-1

Screen clipping taken: 28-12-2022 08:54



if (arr[mid] < k)
s = mid+1
else e = mid-1

Example 2:

Input:
N = 4
Arr = {1, 3, 5, 6}
k = 2
Output: 1
Explaination: Since 2 is not present in the array but can be inserted at index 1 to make the array sorted.

mid = (start + end) / 2
= (0+3)/2 = 1
arr[mid] <= 5
return mid;

Screen clipping taken: 28-12-2022 08:55

TC :- O(logn)
SC :- O(1)

```java
//User function Template for Java

class Solution
{
    static int searchInsertK(int arr[], int n, int k)
    {
        // code here
        int start=0;
        int end = n-1;
        while(start <= end){
            int mid = (start + end)/2;
            if(arr[mid] == k){
                return mid;
            }
            if(arr[mid]  < k){
                start =mid +1;


            }
            else if(arr[mid] > k ){
                end = mid -1;
            }
        }
        return start;
    }
}
```

Screen clipping taken: 28-12-2022 09:03

Que :- Check if array is sorted :-

Check if array is sorted

Easy Accuracy: 39.37% Submissions: 78K+ Points: 2

Land your Dream Job with Mega Job-a-thon. Register Now!

Given an array **arr[]** of size **N**, check if it is sorted in non-decreasing order or not.

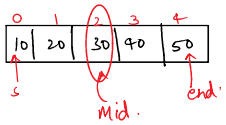
Example 1:

Input:
N = 5
arr[] = {10, 20, 30, 40, 50}
Output: 1
Explanation: The given array is sorted.

Example 2:

Input:
N = 6
arr[] = {90, 80, 100, 70, 40, 30}
Output: 0
Explanation: The given array is not sorted.

Screen clipping taken: 28-12-2022 09:04



if (arr[mid] < arr[mid-1])
{ return 1
if (arr [mid]

```java
// User function Template for Java

class Solution {
    boolean arraySortedOrNot(int[] arr, int n) {
        // code here
        for(int i=0;i<n-1;i++){
            if(arr[i] > arr[i+1]){
                return false;
            }
        }
        return true;
    }
}
```

Screen clipping taken: 28-12-2022 09:15

Que :- find the first or last occurances of a given number in a sorted array

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

Medium 15.2K 363

Companies

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

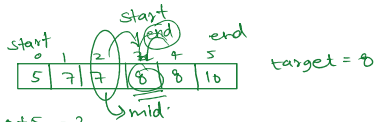
If `target` is not found in the array, return `[-1, -1]`.

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]

Screen clipping taken: 30-12-2022 00:00



0+5/2 = 2
=> arr[mid] < target
s = mid+1

3+5/2 = 4
arr[mid] == target
ans1 = arr[mid]
end = mid-1

3+3=6/2 = 3
return 3

```java
class Solution {

    public int[] searchRange(int[] nums, int target) {
        int ans[] = new int[2];
        int firstans = -1;
        int lastans = -1;

        int start =0;
        int end = nums.length -1;
        while(start <= end){
            int mid = start + (end-start) /2;
            if(nums[mid] == target){
                firstans = mid;
                end = mid -1;
            }
            if(nums[mid] < target){
                start = mid +1;
            }
            else if(nums[mid] > target){
                end = mid-1;
            }
```

```java
class Solution {

    public int[] searchRange(int[] nums, int target) {
        int ans[] = new int[2];
        int firstans = -1;
        int lastans = -1;

        int start =0;
        int end = nums.length -1;
        while(start <= end){
            int mid = start + (end-start) /2;
            if(nums[mid] == target){
                firstans = mid;
                end = mid -1;
            }
            if(nums[mid] < target){
                start = mid +1;
            }
            else if(nums[mid] > target){
                end = mid-1;
            }
        }
        ans[0] = firstans;
        start =0;
        end = nums.length -1;
        while(start <= end){
            int mid = start + (end-start) /2;
            if(nums[mid] == target){
                lastans = mid;
                start = mid +1;
            }
            if(nums[mid] < target){
                start = mid +1;
            }
            else if(nums[mid] > target){
                end = mid-1;
            }

        }
        ans[1] = lastans;
        return ans;
    }
}
```

Screen clipping taken: 30-12-2022 08:54

que:- count occurrences:-

Number of occurrence

Basic Accuracy: 59.34% Submissions: 98K+ Points: 1

Stand out from the crowd. Prepare with Complete Interview Preparation

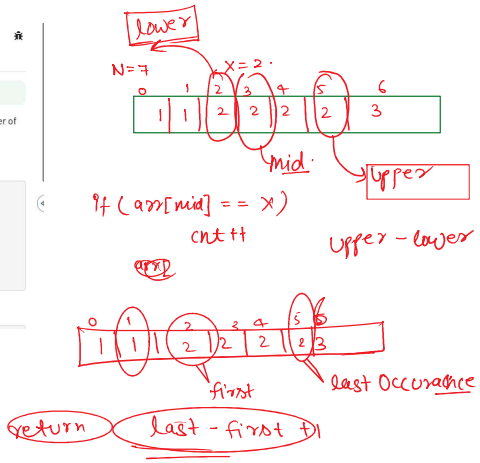
Given a sorted array **Arr** of size **N** and a number **X**, you need to find the number of occurrences of **X** in **Arr**.

Example 1:

Input:
N = 7, X = 2
Arr[] = {1, 1, 2, 2, 2, 2, 3}
Output: 4
Explanation: 2 occurs 4 times in the given array.

Example 2:

Screen clipping taken: 30-12-2022 08:54




```java
class Solution {
    int count(int[] nums, int n, int target) {
        int ans[] = new int[2];
        int firstans = -1;
        int lastans = -1;

        int start =0;
        int end = nums.length -1;
        while(start <= end){
            int mid = start + (end-start) /2;
            if(nums[mid] == target){
                firstans = mid;
                end = mid -1;
            }
            if(nums[mid] < target){
                start = mid +1;
            }
            else if(nums[mid] > target){
                end = mid-1;
            }
        }
        ans[0] = firstans;
        start =0;
        end = nums.length -1;
        while(start <= end){
            int mid = start + (end-start) /2;
            if(nums[mid] == target){
                lastans = mid;
                start = mid +1;
            }
            if(nums[mid] < target){
                start = mid +1;
            }
            else if(nums[mid] > target){
                end = mid-1;
            }

        }
        ans[1] = lastans;
        if(ans[1] == -1 || ans[0] == -1){
            return 0;
        }
        int cnt =  ans[1] - ans[0] ;
        return cnt+1;
    }
}
```

TC:-O(logn)
SC:-O(1)

Screen clipping taken: 30-12-2022 09:04

Linear search Approach:-

Method 1 (Linear Search)
Linearly search for x, count the occurrences of x and return the count.

C++ Java Python3 C# PHP Javascript

```java
// Java program to count occurrences
// of an element

class Main
{
    // Returns number of times x occurs in arr[0..n-1]
    static int countOccurrences(int arr[], int n, int x)
    {
        int res = 0;
        for (int i=0; i<n; i++)
            if (x == arr[i])
                res++;
        return res;
    }

    public static void main(String args[])
    {
        int arr[] = {1, 2, 2, 2, 2, 3, 4, 7 ,8 };
        int n = arr.length;
        int x = 2;
        System.out.println(countOccurrences(arr, n, x));
    }
}
```

TC:-O(n)
SC:-O(1)

Output :

Screen clipping taken: 30-12-2022 09:05

que:-

Peak element

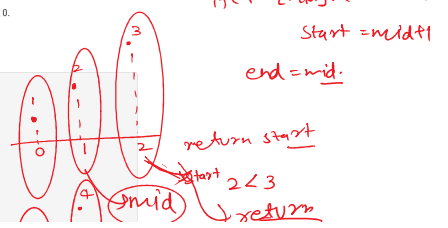
Easy Accuracy: 38.86% Submissions: 255K+ Points: 2

Stand out from the crowd. Prepare with Complete Interview Preparation

An element is called a peak element if its value is not smaller than the value of its adjacent elements(if they exists).
Given an array **arr[]** of size **N**, **Return the index of any one of its peak elements**.
Note: The generated output will always be 1 if the index that you return is correct. Otherwise output will be 0.

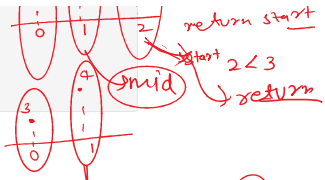
Example 1:

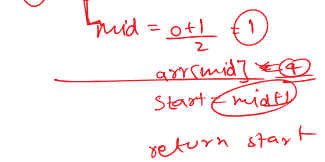
Input:
N = 3
arr[] = {1,2,3}
Possible Answer: 2
Generated Output: 1
Explanation: index 2 is 3.
It is the peak element as it is
greater than its neighbour 2.
If 2 is returned then the generated output will be 1 else 0.



Possible Answer: 2
Generated Output: 1
Explanation: index 2 is 3.
It is the peak element as it is
greater than its neighbour 2.
If 2 is returned then the generated output will be 1 else 0.

Screen clipping taken: 30-12-2022 09:06

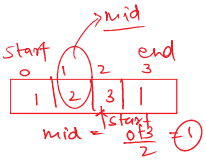




Case 2 :- [3, 4]

```
// arr: input array
// n: size of array
class Solution
{
    public:
    int peakElement(int arr[], int n)
    {
        // Your code here
        int start = 0;
        int end = n-1;
        while(start<end){
            int mid = (start + end)/2;
            if(arr[mid] < arr[mid+1]){
                start = mid+1;
            }
            else {
                end = mid;
            }
        }
        return start;
    }
};
// } Driver Code Ends
```

Screen clipping taken: 30-12-2022 09:11



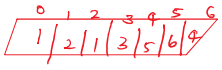
Example 1:

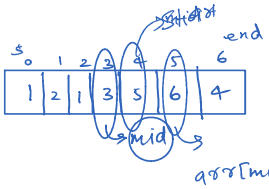
```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index
number 2.
```

Screen clipping taken: 30-12-2022 09:11

arr[mid] < mid+1
start = mid+1
3+2 = 5/2 = 2
arr[mid] =
return start
→ 2

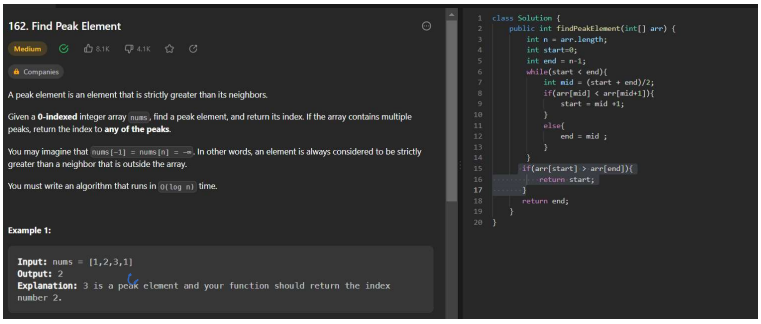
Screen clipping taken: 30-12-2022 10:09





arr[mid] < arr[mid+1]

Same question on Leetcode :-



162. Find Peak Element

Medium

Companies

A peak element is an element that is strictly greater than its neighbors.

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

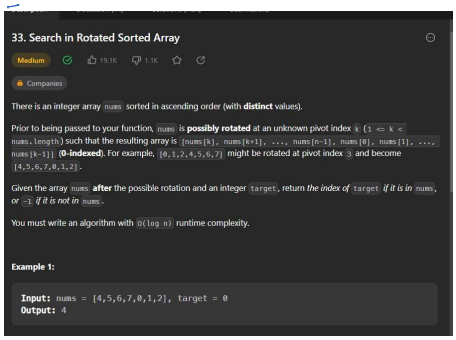
You must write an algorithm that runs in `O(log n)` time.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index
number 2.
```

```
class Solution {
    public int findPeakElement(int[] arr) {
        int n = arr.length;
        int start=0;
        int end = n-1;
        while(start < end){
            int mid = (start + end)/2;
            if(arr[mid] < arr[mid+1]){
                start = mid +1;
            }
            else{
                end = mid ;
            }
        }
        if(arr[start] > arr[end]){
            return start;
        }
        return end;
    }
}
```

que :-



33. Search in Rotated Sorted Array

Medium

Companies

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

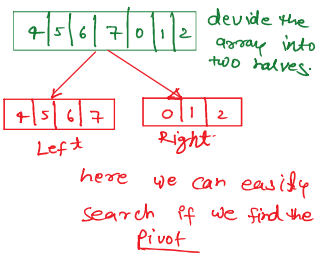
Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot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

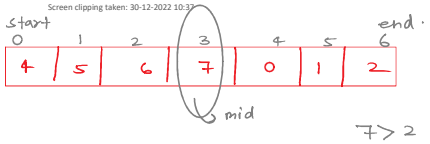
You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

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



Screen clipping taken: 30-12-2022 10:37

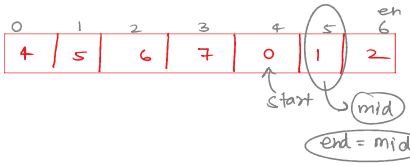


7 > 2 → go toward right.

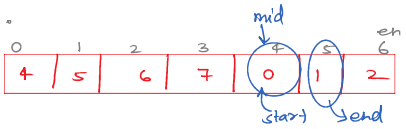
#

7 > 2 → go toward right.

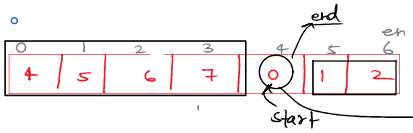
↳ left = mid + 1



1 < 2 · go towards left.



0 < 1
go towards left



→ finding that minimum element in the array

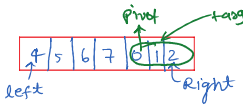
- We faces two cases :-

① if (arr[mid] > arr[right] , it means we are in right sorted array, so go towards left to find the pivot elements.

② else if means the array is rotated, so go towards left to find that right sorted array.

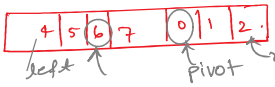
Now how ~~to~~ figure out. Where is our target located. is it in the left subarray or right subarray?

Case 1:- If our target was 1



target 1 < 2. & pivot < 1 it ovious that the pivot is lie towards right subarray.

Case 2:-



if our target was 6

Since our target 6 > 2 and at the same time 0 < 6 it's obvious that the target will lie towards left subarray

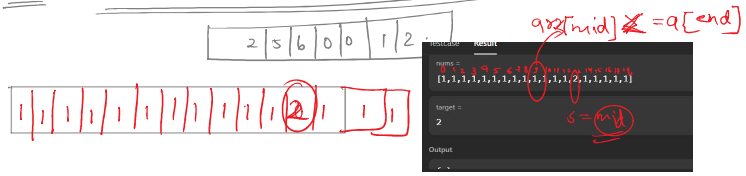
```
class Solution {
    public int search(int[] nums, int target) {
        int n = nums.length;
        int start=0;
        int end = n-1;
        while(start<end){           //iska use krke pivot find kiya
            int mid = (start + end) /2;
            if(nums[mid]  > nums[end]){
                start = mid +1;

            }
            else{
                end = mid;
            }
        }
        int pivot = start;        //abhi pivot pe start set kiya  pivot ==0
        start=0;
        end = n-1;
        if(nums[pivot] <= target && target <= nums[end] ){  // let 1 target 0 < 1 && 1 < 2 right side

            start = pivot;                                          //nums[pivot] <=target to right
        }
        else{
            end = pivot;                //nums[pivot] >=target --> left side
        }
        while(start <=end){
            int mid  = (start + end)/2;
            if(nums[mid] == target){
                return mid;
            }
            if(nums[mid] < target){
                start = mid +1;
            }
            else if(nums[mid] > target){
                end = mid -1;
            }
        }
        return -1;
    }
}
```

Screen clipping taken: 30-12-2022 11:54

que :- Search in Rotated sorted Array - II :-

arr[mid] ≤ = a[end]

que:- Search in Rotated sorted Array - II



arr[mid] ≤ = a[end]

s = mid

Screen clipping taken: 30-12-2022 15:17

```java
class Solution {
    public boolean search(int[] nums, int target) {
        int n = nums.length;
        int start= 0;
        int end = n-1;
        while(start < end){
            int mid = (start + end)/2;
            if(nums[mid] > nums[end]){
                start = mid+1;
            }
            else{
                end = mid;
            }
        }
        int pivot = start;
        start =0;
        end = n-1;
        if(nums[pivot] <= target && target <= nums[end]){
            start = pivot;
        }
        else{
            end = pivot;
        }
        while(start <=end){
            int mid = (start + end)/2;
            if(nums[mid] == target){
                return true;
            }
            if(nums[mid] < target){
                start = mid +1;
            }
            else{
                end= mid -1;
            }
        }
        return false;
    }
}
```

Screen clipping taken: 30-12-2022 15:56

que:- find Minimum in Rotated Sorted Array:-

153. Find Minimum in Rotated Sorted Array

Hint

Medium 9.3K 447

Companies

Suppose an array of length `n` sorted in ascending order is **rotated** between `1` and `n` times. For example, the array `nums = [0,1,2,4,5,6,7]` might become:

- `[4,5,6,7,0,1,2]` if it was rotated `4` times.
- `[0,1,2,4,5,6,7]` if it was rotated `7` times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

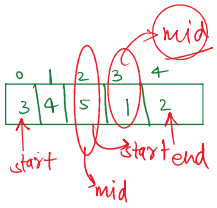
Given the sorted rotated array `nums` of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in `O(log n) time.`

Example 1:

```
Input: nums = [3,4,5,1,2]
Output: 1
Explanation: The original array was [1,2,3,4,5] rotated 3 times.
```

Screen clipping taken: 30-12-2022 15:57



arr[mid] > arr[end]
{ start = mid+1
arr[mid] < arr[end]
end = mid;
}
return arr[start]

```java
class Solution {
    public int findMin(int[] arr) {
        int n = arr.length;
        int start= 0;
        int end = n-1;
        while(start <end){
            int mid = (start + end)/2;
            if(arr[mid] > arr[end]){
                start = mid +1;
            }
            else{
                end = mid;
            }
        }
        return arr[start];
    }
}
```

Screen clipping taken: 30-12-2022 16:02

que:- Single element in a Sorted Array:-

540. Single Element in a Sorted Array

Medium 6.8K 117

Companies

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once.

Return *the single element that appears only once.*

Your solution must run in `O(log n)` time and `O(1)` space.

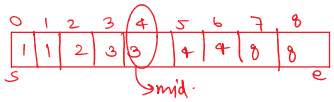
Example 1:

```
Input: nums = [1,1,2,3,3,4,4,8,8]
Output: 2
```

Example 2:

```
Input: nums = [3,3,7,7,10,11,11]
Output: 10
```

Screen clipping taken: 30-12-2022 16:03



if(arr[mid] != a[mid-1] ||
arr[mid] != a[mid+1];
return mid;

first Approach using HashMap :-

```
public int singleNonDuplicateUsingHashMap(int[] nums) {
    HashMap<Integer, Integer> map = new HashMap<>();

    //store the count of each element in the map
    for (int i = 0; i < nums.length; i++) {
        if (!map.containsKey(nums[i])) {
            //if the key doesn't exist, create a new one
            map.put(nums[i], 1);
        }else {
          //update the count of the existing by incrementing it by 1
            map.put(nums[i], map.get(nums[i]) + 1);
        }
    }

    //loop through map and get the one that has the value of 1
    for (Map.Entry<Integer, Integer> entry : map.entrySet()) {
        if (entry.getValue() == 1) {
            return entry.getKey();
        }
    }
```

O(n) + O(n)

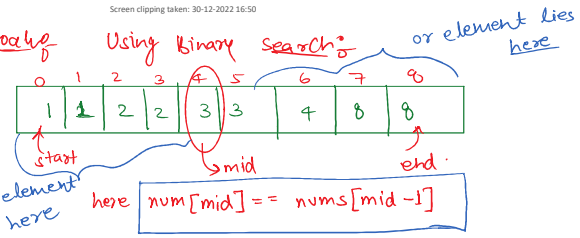
Screen clipping taken: 30-12-2022 16:49

second Approach :- below...

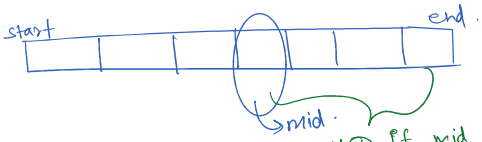
```
public int singleNonDuplicateLinearly(int[] nums) {
    if (nums.length == 1) return nums[0];
    for (int i = 0; i < nums.length; i+=2) {
        if (i == nums.length - 1) return nums[i];
        if (i+1 < nums.length && nums[i] != nums[i+1]) {
            return nums[i];
        }
    }
    return -1;
}
```

Screen clipping taken: 30-12-2022 16:50

third Approach :- Using Binary Search :-



So the condition will be-



#① If mid to end is even then element lies from start to mid

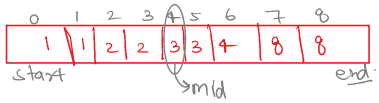
then end = mid-2

because we don't want to iterate num[mid] && nums[mid-1] again

② If mid to end is odd then the element lies from mid to end

start = mid+1

2nd scenario where nums[i] == nums[i+1]

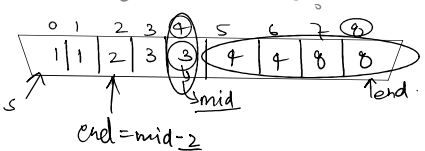


If start to mid is even then start = mid+2

else start to mid is odd then ~~start~~ end = mid-1

now the scenario when num[mid] != num[mid+1] || num[mid-1];

1st scenario:



```java
class Solution {
    public int singleNonDuplicate(int[] nums) {
        if(nums.length == 1) return nums[0];

        int len = nums.length;
        int left = 0;
        int right = len - 1;

        while(left <= right && left < len && right >= 0){
            int mid = left + (right - left)/2;

            if((mid-1 >= 0 && nums[mid-1] == nums[mid]) || (mid + 1 < len && nums[mid+1] == nums[mid])){
// nums[mid] is not single
                int currien = right - left; // actual length - 1
                if((currien/2) % 2 == 0){
                    if(nums[mid-1] == nums[mid]){
                        // The element is on the left hand side
                        right = mid - 2; // Skip mid-1 and mid as we know they are not single
                    }
                    else{
                        // The element is on the right hand side
                        left = mid + 2;
                    }
                }
                else{
                    if(nums[mid-1] == nums[mid]){
                        // The element is on the right hand side
                        left = mid + 1; // Skip mid
                    }
                    else{
                        // The element is on the left hand side
                        right = mid - 1;
                    }
                }
            }
            else return nums[mid];
        }

        return nums[left];
    }
}
```

Screen clipping taken: 30-12-2022 18:38

```
Let's start with two simple observations.

Example 1: An array with length 2*4 + 1
left = 0, right = 8, mid = 4.
If the single element X is on the left hand side, nums[mid] == nums[mid-1]:
[1, 1, X, 2, 2(mid), 3, 3, 4, 4]
If the single element X is on the right hand side, nums[mid] == nums[mid+1]:
[1, 1, 2, 2, 3(mid), 3, X, 4, 4]

Example 2: An array with length 2*3 + 1
left = 0, right = 6, mid = 3.
If the single element X is on the left hand side, nums[mid] == nums[mid+1]:
[1, 1, X, 2(mid), 2, 3, 3]
If the single element X is on the right hand side, nums[mid] == nums[mid-1]:
[1, 1, 2, 2(mid), 3, 3, X]

In general, for an array with length 2*n + 1, if n is even, the behavior of mid will be the same as that in Example 1. Otherwise, it will be as in Example 2.
```

ques- find kth element of two sorted Arrays-

Screen clipping taken: 30-12-2022 18:41

K-th element of two sorted Arrays

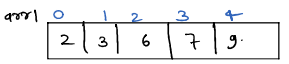
Medium Accuracy: 37.4% Submissions: 131K+ Points: 4

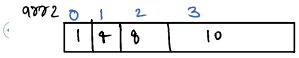
Stand out from the crowd. Prepare with Complete Interview Preparation

Given two sorted arrays **arr1** and **arr2** of size **N** and **M** respectively and an element **K**. The task is to find the element that would be at the k'th position of the final sorted array.

Example 1:

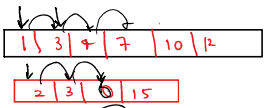
Input:
arr1[] = {2, 3, 6, 7, 9}
arr2[] = {1, 4, 8, 10}
k = 5
Output:
6
Explanation:
The final sorted array would be -
1, 2, 3, 4, 6, 7, 8, 9, 10
The 5th element of this array is 6.





Merge sort Approach-

sort



cnt = 1 2 3 4 5

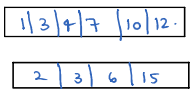
1 2 3 3 4 when cnt = 5 return this element

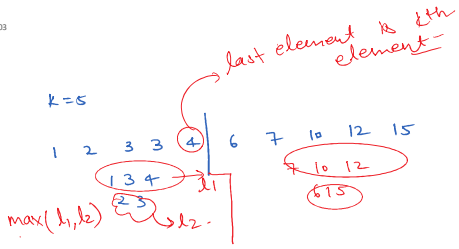
Brute force Approach-

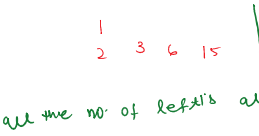
```java
//User function Template for Java

class Solution {
    public long kthElement( int arr1[], int arr2[], int n, int m, int k) {

        int merged[] = new int[n+m];
    int x=0,i=0,j=0;
    while(i<n && j<m){
        if(arr1[i]<=arr2[j]){
            merged[x++]=arr1[i++];
        }else{
            merged[x++]=arr2[j++];
        }
    }
    while(i<n){
        merged[x++]=arr1[i++];
    }
    while(j<m){
        merged[x++]=arr2[j++];
    }
    return merged[k-1];
    }
}
```

Screen clipping taken: 31-12-2022 10:03

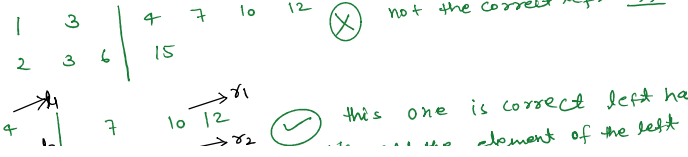
Efficient Approach-

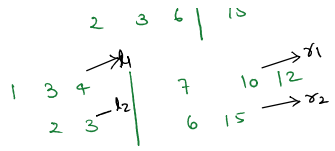






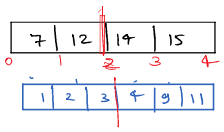
all the no. of left's always lesser than all the no. on Right-





this one is correct left half. bcz all the element of the left half is smaller than all the element of the right half.

if $l_1 <= r_2$ and if $l_2 <= r_1$ — Valid



$l_1 <= r_2$
$l_2 <= r_1$

$k = 5$

low = 0

high = 4

$\frac{0+4}{2} = 2$

$2 - 1 = 1$

Cut2 = 3

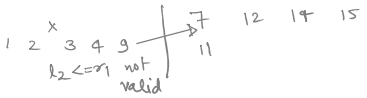
7 | 12 → not satisfy | 14 15

1 2 3 | 4 9 11

then high = mid − 1

Cut 1 = $\frac{0+1}{2} = 0$

Cut 2 = 5



$l_2 <= r_1$ not valid

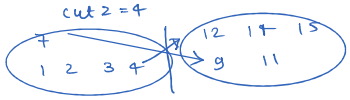
low = mid + 1

low = 1, high = 1

mid = 1

cut 1 = 1

cut 2 = 4

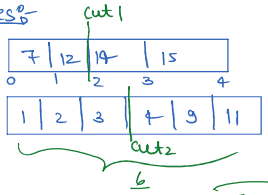


Validity condition satisfy

Kth element will be here is (7)

$\log_2(\min(m+n))$

Edge cases:



min: low = 0, high = 3 (max element) — lets $k = 3$

min element = 0, high = 4 — $X = 7$

6

$l_1 = arr1[cut1 - 1]$ $r_1 = arr1[cut1]$

$l_2 = arr2[cut2 - 1]$ $r_2 = arr2[cut2]$ → INT_MAX

If do not any thing INT_MIN

Exp. $O(\log_2(\min(m+n)))$

```
//User function Template for Java

class Solution {
    public long kthElement( int arr1[], int arr2[], int n, int m, int k) {

        if(n > m){
            return kthElement(arr2, arr1, m, n, k);
        }
        int low = Math.max(0 , k-m), high = Math.min(k,n);

        while(low <= high){
            int cut1 = (low + high)/2;
            int cut2 = k - cut1;
            int l1 = cut1 ==0 ? Integer.MIN_VALUE :arr1[cut1 -1];
            int l2 = cut2 ==0 ? Integer.MIN_VALUE : arr2[cut2 -1];
            int r1 = cut1 == n ? Integer.MAX_VALUE : arr1[cut1];
            int r2 = cut2 == m ? Integer.MIN_VALUE : arr2[cut2];

            if(l1 <= r2 && l2 <=r1){
                return Math.max(l1, l2);
            }
            else if(l1 > l2){
                high = cut1 -1;
            }
            else{
                low = cut1+1;
            }
        }
        return 1;
    }
}
```

Screen clipping taken: 31-12-2022 10:50

Que :-

Rotation

Rotation

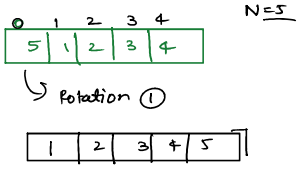
Easy Accuracy: **23.16%** Submissions: **107K+** Points: **2**

Stand out from the crowd. Prepare with Complete Interview Preparation

Given an ascending sorted rotated array **Arr** of distinct integers of size **N**. The array is right rotated **K** times. Find the value of **K**.

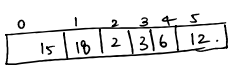
Example 1:

Input:
N = 5
Arr[] = {5, 1, 2, 3, 4}
Output: 1
Explanation: The given array is 5 1 2 3 4.
The original sorted array is 1 2 3 4 5.
We can see that the array was rotated
1 times to the right.



Screen clipping taken: 31-12-2022 10:51

Bruteforce Approach :-



mini = 15 , min index = 0

At i=1 min = 15 , min_index = 0

At i=2 min = min(2,15) = 2 , minindex = 2

At i=3 min = min(2,3) = 2 , minindex = 2

At i=4 min = min(2,6) = 2 minindex = 2.

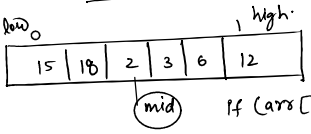
At i=5 min = min(2,18) = 2 minindex = 2

↳ return minindex

Screen clipping taken: 31-12-2022 11:01

```
//{ Driver Code Ends
//User function template for C++
class Solution{
public:
    int findKRotation(int arr[], int n) {
        // code here
        int min = arr[0] , minindex = 0;
        for(int i=0;i<n;i++){
            if(min > arr[i]){
                min = arr[i];
                minindex =i;
            }
        }
        return minindex;
    }
};
//{ Driver Code Ends
```

efficient solution :- Using Binary Search Approach :-



If (arr[mid-1] > arr[mid])

the require index is mid

```
//{ Driver Code Starts
#include <bits/stdc++.h>

using namespace std;


// } Driver Code Ends
//User function template for C++
class Solution{
public:
        int findKRotation(int arr[], int n) {
            // code here
            int low =0;
            int high = n-1;
            while(low < high){
                int mid = (low + high)/2;
                if(arr[mid] < arr[mid -1]){
                    return mid;
                }
                if(arr[mid] <  arr[high]){
                    high = mid;
                }
                else{
                    low = mid +1;
                }
            }
            return high;
        }

};

//{ Driver Code Starts.

int main() {
    int t;
    cin >> t;
    while (t--) {
        int n, i;
        cin >> n;
        int a[n];
        for (i = 0; i < n; i++) {
            cin >> a[i];
        }
        Solution ob;
        auto ans = ob.findKRotation(a, n);
        cout << ans << "\n";
    }
    return 0;
}

// } Driver Code Ends
```