Vasanthan_T_DSA_Practice-5

14/11/2024

DSA Practice Problems_5

1. **Find Transition Point**

Given a **sorted array, arr[]** containing only **0s** and **1s**, find the **transition point**, i.e., the **first index** where **1** was observed, and **before that**, only 0 was observed. If **arr** does not have any **1**, return **-1**. If array does not have any **0**, return **0**.

Examples:

Input: arr[] = [0, 0, 0, 1, 1]

Output: 3

Explanation: index 3 is the transition point where 1 begins.

Input: arr[] = [0, 0, 0, 0]

Output: -1

Explanation: Since, there is no "1", the answer is -1.

Code:

```
import java.util.*;
class TransitionPt{
   public static int Point(int arr[]) {
         int l=0;
      int r=arr.length-1;
      while(l<=r){
        int mid=l+(r-l)/2;
        if(arr[mid]==0){
           l=mid+1;
        }
```

```
        else{
            if(mid==0 || mid>0&& arr[mid-1]==0) return mid;
            r=mid-1;
        }
      }
      return -1;
   }
        public static void main(String[] ar){
                int[] arr= {0, 0, 0, 1, 1};
                System.out.println(Point(arr));
        }
}
```

Output:

```
D:\code\JavaCodes>javac TransitionPt.java

D:\code\JavaCodes>java TransitionPt.java
3
```

Time Complexity:O(log n)

Space Complexity: O(1)

2. Wave Array

Given a sorted array arr[] of distinct integers. Sort the array into a wave-like array(In Place). In other words, arrange the elements into a sequence such that arr[1] >= arr[2] <= arr[3] >= arr[4] <= arr[5].....
If there are multiple solutions, find the lexicographically smallest one.

Note: The given array is sorted in ascending order, and you don't need to return anything to change the original array.

Examples:

Input: arr[] = [1, 2, 3, 4, 5]

Output: [2, 1, 4, 3, 5]

Explanation: Array elements after sorting it in the waveform are 2, 1, 4, 3, 5

Code:

```
import java.util.*;
class WaveArray{

   public static int[] convertToWave(int[] arr) {
      if(arr.length<2) return arr ;
      for(int i=0;i<arr.length-1;i+=2){
         arr[i]=arr[i]^arr[i+1];
         arr[i+1]=arr[i]^arr[i+1];
         arr[i]=arr[i]^arr[i+1];
      }
         return arr;


   }
         public static void main(String[] ar){
                  int[] arr = {1, 2, 3, 4, 5};
                  arr = convertToWave(arr);
                  for(int i=0;i<arr.length;i++){
                           System.out.print(arr[i] +" ");
                  }

         }
```

}

Output:

```
D:\code\JavaCodes>javac  WaveArray.java

D:\code\JavaCodes>java  WaveArray.java
2 1 4 3 5
```

Time Complexity: O(n)

Space Complexity: O(1)

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

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]

Code:

```
import java.util.*;
public class FirstLastElement{
   public static int[] searchRange(int[] nums, int target) {
      int[] ans=new int[2];
      ans[0]=findFirst(nums,target);
      ans[1]=findLast(nums,target);
      return ans;
   }
```

```
public static int findFirst(int[] nums,int k){
    int l=0;
    int r=nums.length-1;
    int st=-1;
    while(l<=r){
        int m=l+(r-l)/2;
        if(nums[m]<k){
            l=m+1;
        }
        else if(nums[m]>=k){
            if(nums[m]==k) st=m;
            r=m-1;
        }

    }
    return st;
}
public static int findLast(int[] nums,int k){
    int l=0;
    int r=nums.length-1;
    int end=-1;
    while(l<=r){
        int m=l+(r-l)/2;
        if(nums[m]<=k){
            if(nums[m]==k){
                end=m;
            }
            l=m+1;
```

```
        }
        else if(nums[m]>k){
           r=m-1;
        }

      }
      return end;
   }
        public static void main(String[] args){
                int[] nums = {5,7,7,8,8,10};
                int k=8;
      System.out.println(Arrays.toString(searchRange(nums, k)));
        }
}
```

Output:

```
D:\code\JavaCodes>javac FirstLastElement.java

D:\code\JavaCodes>java FirstLastElement.java
[3, 4]
```

Time Complexity:O(log n)

Space Complexity: O(1)

4. First Repeating Element

Difficulty: EasyAccuracy: 32.57%Submissions: 268K+Points: 2

Given an array arr[], find the first repeating element. The element should occur more than once and the index of its first occurrence should be the smallest.

Note:- The position you return should be according to 1-based indexing.

Examples:

Input: arr[] = [1, 5, 3, 4, 3, 5, 6]

Output: 2

Explanation: 5 appears twice and its first appearance is at index 2 which is less than 3 whose first the occurring index is 3.

Code:

```
import java.util.*;

class FirstRepeated{


   public static int Repeated(int[] arr) {
      HashSet<Integer> hs=new HashSet<>();
      int ind=-1;
      int n=arr.length;
      for(int i=n-1;i>=0;i--){
         if(hs.contains(arr[i])){
            ind=i;
         }
         else{
            hs.add(arr[i]);
         }
      }
      return(ind==-1)?-1:ind+1;
   }
         public static void main(String[] ar){
```

```
        int[] arr={1, 5, 3, 4, 3, 5, 6};
        System.out.println(Repeated(arr));
    }
}
```

Output:

```
D:\code\JavaCodes>javac FirstRepeated.java

D:\code\JavaCodes>java FirstRepeated.java
2
```

Time Complexity:O(n)

Space Complexity: O(n)

5. Remove Duplicates Sorted Array

Given a sorted array arr. Return the size of the modified array which contains only distinct elements.
Note:
1. Don't use set or HashMap to solve the problem.
2. You must return the modified array size only where distinct elements are present and modify the original array such that all the distinct elements come at the beginning of the original array.

Examples :

Input: arr = [2, 2, 2, 2, 2]

Output: [2]

Explanation: After removing all the duplicates only one instance of 2 will remain i.e. [2] so modified array will contains 2 at first position and you should return 1 after modifying the array, the driver code will print the modified array elements.

Code:

```
import java.util.*;
```

```
class RemoveDup{


    public static int remove_duplicate(List<Integer> arr) {
        // Code Here
        int n=arr.size();
        int ind=1;
        for(int i=1;i<n;i++){
            if(!arr.get(i).equals(arr.get(i-1))){
                arr.set(ind, arr.get(i));
                ind++;
            }
        }
        return ind;


    }
            public static void main(String[] ar){
                    List<Integer> ans=new ArrayList<>(Arrays.asList(2, 2, 2, 2, 2));
                    System.out.println(remove_duplicate(ans));
            }
}
```

Output:

```
D:\code\JavaCodes>javac RemoveDup.java

D:\code\JavaCodes>java RemoveDup.java
1
```