Prasath S(22IT075)

DSA-PRACTICE Day-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:

class Solution {

public static int transitionPoint(int arr[]) {

int n = arr.length;

if (arr[0] == 1) {

return 0;

}

int lb = 0, ub = n - 1;

while (lb <= ub) {

int mid = (lb + ub) / 2;

if (arr[mid] == 0) lb = mid + 1;

else if (arr[mid] == 1) {

if (mid == 0 || arr[mid - 1] == 0) return mid;

ub = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

int[] arr1 = {0, 0, 0, 1, 1};

int[] arr2 = {0, 0, 0, 0};

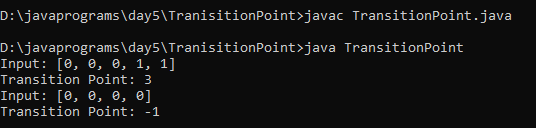
System.out.println("Input: " + java.util.Arrays.toString(arr1));

System.out.println("Transition Point: " + transitionPoint(arr1));

System.out.println("Input: " + java.util.Arrays.toString(arr2));

System.out.println("Transition Point: " + transitionPoint(arr2));

}

}  
output: 

TIME COMPLEXITY:O(log n)

### 2. First Repeating Element

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.

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

**Output:** -1

**Explanation:** All elements appear only once so answer is -1.

CODE:

import java.util.HashMap;

class FirstRepeatingElementFinder {

public static int firstRepeated(int[] arr) {

HashMap<Integer, Integer> ans = new HashMap<>();

int minIndex = Integer.MAX\_VALUE;

for (int i = 0; i < arr.length; i++) {

if (ans.containsKey(arr[i])) {

minIndex = Math.min(minIndex, ans.get(arr[i]));

} else {

ans.put(arr[i], i + 1);

}

}

return (minIndex == Integer.MAX\_VALUE) ? -1 : minIndex;

}

public static void main(String[] args) {

int[] arr1 = {1, 5, 3, 4, 3, 5, 6};

int[] arr2 = {1, 2, 3, 4};

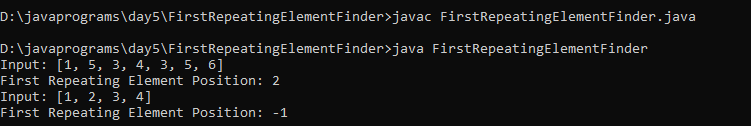
System.out.println("Input: " + java.util.Arrays.toString(arr1));

System.out.println("First Repeating Element Position: " + firstRepeated(arr1));

System.out.println("Input: " + java.util.Arrays.toString(arr2));

System.out.println("First Repeating Element Position: " + firstRepeated(arr2));

}

}  
OUTPUT:  
  
TIME COMPLEXITY:O(n)

### 3. 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.

**Input:** arr = [1, 2, 4]

**Output:** [1, 2, 4]  
**Explation:** As the array does not contain any duplicates so you should return 3.

CODE: import java.util.ArrayList;

import java.util.Arrays;

class DuplicateRemover {

public int removeDuplicates(ArrayList<Integer> nums) {

if (nums.size() == 0) return 0;

int j = 1;

for (int i = 1; i < nums.size(); i++) {

if (!nums.get(i).equals(nums.get(i - 1))) {

nums.set(j, nums.get(i));

j++;

}

}

while (nums.size() > j) {

nums.remove(nums.size() - 1);

}

return j;

}

public static void main(String[] args) {

DuplicateRemover remover = new DuplicateRemover();

ArrayList<Integer> arr1 = new ArrayList<>(Arrays.asList(2, 2, 2, 2, 2));

int newSize1 = remover.removeDuplicates(arr1);

System.out.println("Output Array: " + arr1.subList(0, newSize1));

System.out.println("Returned Size: " + newSize1);

ArrayList<Integer> arr2 = new ArrayList<>(Arrays.asList(1, 2, 4));

int newSize2 = remover.removeDuplicates(arr2);

System.out.println("Output Array: " + arr2.subList(0, newSize2));

System.out.println("Returned Size: " + newSize2);

}

}

TIME COMPLEXITY:O(N)

4.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.

**Input:** arr[] = [2, 4, 7, 8, 9, 10]

**Output: [**4, 2, 8, 7, 10, 9]

**Explanation:** Array elements after sorting it in the waveform are 4, 2, 8, 7, 10, 9.  
  
Input: arr[] = [1]  
Output: [1]

CODE:

import java.util.Arrays;

class WaveArraySorter {

public static void waveSort(int[] arr) {

for (int i = 1; i < arr.length; i += 2) {

int temp = arr[i];

arr[i] = arr[i - 1];

arr[i - 1] = temp;

}

}

public static void main(String[] args) {

int[] arr1 = {1, 2, 3, 4, 5};

waveSort(arr1);

System.out.println(Arrays.toString(arr1)); // Output: [2, 1, 4, 3, 5]

int[] arr2 = {2, 4, 7, 8, 9, 10};

waveSort(arr2);

System.out.println(Arrays.toString(arr2)); // Output: [4, 2, 8, 7, 10, 9]

int[] arr3 = {1};

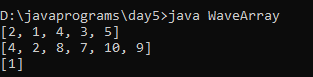
waveSort(arr3);

System.out.println(Arrays.toString(arr3)); // Output: [1]

}

}

OUTPUT:



TIME COMPLEXITY:O(N)