1.Geeks For Geeks Section

Topic: Arrays (Level 1)

1. Given an array of size **n-1** such that it only contains distinct integers in the range of **1 to n**. Return the missing element.

static int missingNumber(int n, int arr[]) {

// Your Code Here

Arrays.sort(arr);

for(int i = 0; i < n - 1; i++)

{

if(arr[i] == i + 1)

{

continue;

}

else

{

return i+1;

}

}

return n;

}

1. **Two Repeated Elements**

Difficulty: **Medium** Accuracy: **28.95%**Submissions: **122K+**Points: **4**

You are given an integer **n**and an integer array **arr** of size **n+2**. All elements of the array are in the range from 1 to **n**. Also, all elements occur once except two numbers which occur twice. Find the two repeating numbers.  
**Note:**Return the numbers in their order of appearing twice. So, if **x** and **y** are repeating numbers, and **x**'s second appearance comes before the second appearance of **y**, then the order should be (**x**, **y**).

**Examples:**

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

**Output:** 1 3

**Explanation:** In the given array, 1 and 3 are repeated two times, and as 1's second appearance occurs before 2's second appearance, so the output should be 1 3.

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

**Output:** 2 1

**Explanation:** In the given array, 1 and 2 are repeated two times and second occurrence of 2 comes before 1. So, the output is 2 1.

Solution :

class Solution {

// Function to find two repeated elements.

public int[] twoRepeated(int n, int arr[]) {

// Your code here

ArrayList<Integer> a = new ArrayList<>();

HashSet<Integer> Set = new HashSet<>();

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

{

boolean boo = Set.add(arr[i]);

if(boo == false)

{

a.add(arr[i]);

}

}

int[] result = new int[a.size()];//we are creating the result variable because we cannot return arraylist

for(int i = 0; i < a.size(); i++)

{

result[i] = a.get(i);//In ArrayList we use get method to assign values

}

return result;

}

}

3.

**Array Duplicates**

Difficulty: **Easy** Accuracy: **18.95%**Submissions: **726K+**Points: **2**

Given an array **arr** of size **n** which contains elements in range from **0** to **n-1**, you need to find all the elements occurring more than once in the given array. Return the answer in ascending order. If no such element is found, return list containing **[-1]**.

**Note:** Try and perform all operations within the provided array. The extra (non-constant) ) space needs to be used only for the array to be returned.

**Examples:**

**Input:** arr[] = {0,3,1,2}, n = 4

**Output:** -1

**Explanation:** There is no repeating element in the array. Therefore output is -1.

**Input:** arr[] = {2,3,1,2,3}, n = 5

**Output:** 2 3

**Explanation:** 2 and 3 occur more than once in the given array.

Solutions:

class Solution {

public static ArrayList<Integer> duplicates(int[] arr) {

// code here

Arrays.sort(arr);

HashSet<Integer> set = new HashSet<>();

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

{

if(arr[i] == arr[i - 1])

{

set.add(arr[i]);

}

}

if(set.isEmpty())

{

set.add(-1);

}

ArrayList<Integer> numbers = new ArrayList<>(set);

Collections.sort(numbers);

return numbers;

}

}

4.

**Array Leaders**

Difficulty: **Easy** Accuracy: **29.94%**Submissions: **688K+**Points: **2**

Given an array **arr**of **n** positive integers, your task is to find all the leaders in the array. An element of the array is considered a leader if it is greater than all the elements on its right side or if it is equal to the maximum element on its right side. The rightmost element is always a leader.

**Examples**

**Input:** n = 6, arr[] = {16,17,4,3,5,2}

**Output:** 17 5 2

**Explanation:** Note that there is nothing greater on the right side of 17, 5 and, 2.

**Input:** n = 5, arr[] = {10,4,2,4,1}

**Output:** 10 4 4 1  
**Explanation:** Note that both of the 4s are in output, as to be a leader an equal element is also allowed on the right. side

**Input:** n = 4, arr[] = {5, 10, 20, 40}   
**Output:** 40  
**Explanation:** When an array is sorted in increasing order, only the rightmost element is leader.

**Input:** n = 4, arr[] = {30, 10, 10, 5}   
**Output:** 30 10 10 5  
**Explanation:** When an array is sorted in non-increasing order, all elements are leaders.

Solution :

class Solution {

// Function to find the leaders in the array.

static ArrayList<Integer> leaders(int n, int arr[]) {

// Your code here

ArrayList<Integer> numbers = new ArrayList<>();

int max = arr[n-1];

numbers.add(max);

for(int i = n - 2; i >= 0; i--)

{

if(arr[i] >= max)

{

numbers.add(arr[i]);

max = arr[i];

}

}

Collections.reverse(numbers);//to reverse arraylist

return numbers;

}

}

5.

**Second Largest**

Difficulty: **Easy**Accuracy: **26.72%**Submissions: **626K+**Points: **2**

Given an array **arr**, return the **second largest** distinct element from an array. If the second largest element doesn't exist then return **-1**.

**Examples:**

**Input:** arr = [12, 35, 1, 10, 34, 1]

**Output:** 34

**Explanation:** The largest element of the array is 35 and the second largest element is 34.

**Input:** arr = [10, 10]

**Output:** -1

**Explanation:** The largest element of the array is 10 and the second largest element does not exist so answer is -1.

class Solution {

public int print2largest(List<Integer> arr) {

// Code Here

HashSet<Integer> set = new HashSet<>();

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

{

set.add(arr.get(i));

}

ArrayList<Integer> al = new ArrayList<>(set);

Collections.sort(al);

int n = al.size();

int m = al.get(n-2);

if(al.size() == 1)

{

return -1;

}

else

{

return m;

}

}

}

6.

**Equilibrium Point**

Difficulty: **Easy** Accuracy: **28.13%**Submissions: **571K+**Points: **2**

Given an array**arr**of non-negative numbers. The task is to find the first **equilibrium point** in an array. The equilibrium point in an array is an index (or position) such that the **sum** of all elements **before**that index is the **same** as the **sum**of elements **after**it.

**Note:** Return equilibrium point in 1-based indexing. Return -1 if no such point exists.

**Examples:**

**Input:** arr[] = [1, 3, 5, 2, 2]  
**Output:** 3

**Explanation:** The equilibrium point is at position 3 as the sum of elements before it (1+3) = sum of elements after it (2+2).

**Input:** arr[] = [1]  
**Output:** 1

**Explanation:** Since there's only one element hence it's only the equilibrium point.

**Input:** n = 3, arr[] = [1, 2, 3]  
**Output:** -1

**Explanation:** There is no equilibrium point in the given array.

Solution:

This question was not solved by me:

class Solution {

// arr: input array

// Function to find equilibrium point in the array.

public static int equilibriumPoint(long arr[]) {

// Your code here

long v1 = 0, v2 = 0;

if(arr.length == 1)

{

return 1;

}

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

{

v1 = v1 + arr[i];

}

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

{

v1 = v1 - arr[i];

if(v1 == v2)

{

return i+1;

}

v2 = v2 + arr[i];

}

return -1;

}

}

**7. Parenthesis Checker**

Difficulty: **Easy** Accuracy: **28.56%**Submissions: **567K+**Points: **2**

Given an expression string **x**. Examine whether the pairs and the orders of {,},(,),[,] are correct in exp.  
For example, the function should return 'true' for exp = [()]{}{[()()]()} and 'false' for exp = [(]).

**Note:**The drive code prints "balanced" if function return true, otherwise it prints "not balanced".

**Example 1:**

**Input**:

{([])}

**Output**:

true

**Explanation**:

{ ( [ ] ) }. Same colored brackets can form

balanced pairs, with 0 number of

unbalanced bracket.

**Example 2:**

**Input**:

()

**Output**:

true

**Explanation**:

(). Same bracket can form balanced pairs,

and here only 1 type of bracket is

present and in balanced way.

**Example 3:**

**Input**:

([]

**Output**:

false

**Explanation**:

([]. Here square bracket is balanced but

the small bracket is not balanced and

Hence, the output will be unbalanced.

Solution:

This solution is not done because this topic comes under stack

8. **Edit Distance**

Difficulty: **Hard** Accuracy: **35.14%**Submissions: **213K+**Points: **8**

Given two strings **str1** and **str2.**Return the minimum number of operations required to convert **str1**to **str2**.  
The possible operations are permitted:

1. Insert a character at any position of the string.
2. Remove any character from the string.
3. Replace any character from the string with any other character.

**Examples:**

**Input:** str1 = "geek", srt2 = "gesek"

**Output:** 1

**Explanation:** One operation is required, inserting 's' between two 'e'.

**Input :** str1 = "gfg", str2 = "gfg"

**Output:** 0

**Explanation:** Both strings are same.

This solution is not done because this topic comes under brute force

9.

**Count pairs with given sum**

Difficulty: **Easy**Accuracy: **31.49%**Submissions: **435K+**Points: **2**

Given an array **arr**, and an integer **k**, find the number of pairs of elements in the array whose sum is **k**.

**Examples:**

**Input:** k = 6, arr[] = [1, 5, 7, 1]

**Output:** 2

**Explanation:**

arr[0] + arr[1] = 1 + 5 = 6

and arr[1] + arr[3] = 5 + 1 = 6.

**Input:** k = 2, arr[] = [1, 1, 1, 1]  
**Output:** 6  
**Explanation:** Each 1 will produce sum 2 with any 1.

**Input:** k = 2, arr[] = [4, 12, 5, 4]  
**Output:** 0  
**Explanation:** There are 0 pairs with sum 2.

Solution:

class Solution {

int getPairsCount(int[] arr, int sum) {

// code here

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

int count = 0;

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

{

if(arr[i] > sum)

{

continue;

}

int Secondval = sum - arr[i];

if(hm.containsKey(Secondval))

{

count = count + hm.get(Secondval);

}

hm.put(arr[i],hm.getOrDefault(arr[i], 0)+1);

}

return count;

}

}

10.

**Indexes of Subarray Sum**

Difficulty: **Medium** Accuracy: **16.5%**Submissions: **1.6M**Points: **4**

Given an unsorted array **arr**of size **n** that contains only non negative integers, find a sub-array (continuous elements) that has sum equal to **s**. You mainly need to return the left and right indexes(**1-based indexing**) of that subarray.

In case of multiple subarrays, return the subarray indexes which come first on moving from left to right. If no such subarray exists return an array consisting of element **-1**.

**Examples:**

**Input:** arr[] = [1,2,3,7,5], n = 5, s = 12

**Output:** 2 4

**Explanation:** The sum of elements from 2nd to 4th position is 12.

**Input:** arr[] = [1,2,3,4,5,6,7,8,9,10], n = 10, s = 15,

**Output:** 1 5

**Explanation:** The sum of elements from 1st to 5th position is 15.

**Input:** arr[] = [7,2,1], n = 3, s = 2

**Output:** 2 2

**Explanation:** The sum of elements from 2nd to 2nd position is 2.

**Input:** arr[] = [5,3,4], n = 3, s = 2

**Output:** -1

**Explanation:** There is no subarray with sum 2

Solution:

class Solution {

public static ArrayList<Integer> subarraySum(int[] arr, int n, int s) {

// code here

ArrayList<Integer> al = new ArrayList<>();

int Current\_Sum = 0;

int start = 0;

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

{

Current\_Sum += arr[end];

while( Current\_Sum > s && end > start)

{

Current\_Sum -= arr[start];

start++;

}

if(Current\_Sum == s)

{

al.add(start + 1);

al.add(end + 1);

return al;

}

}

al.add(-1);

return al;

}

}

Understand the concept of Kadane’s Algorithm

11.

**Kadane's Algorithm**

Difficulty: **Medium** Accuracy: **36.28%**Submissions: **941K+**Points: **4**

Given an integer array **arr[].** Find the contiguous sub-array(containing at least one number) that has the maximum sum and return its sum.

**Examples:**

**Input:** arr[] = [1, 2, 3, -2, 5]

**Output:** 9

**Explanation:** Max subarray sum is 9 of elements (1, 2, 3, -2, 5) which is a contiguous subarray.

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

**Output:** -1

**Explanation:** Max subarray sum is -1 of element (-1)

**Input:** arr[] = [5, 4, 7]

**Output:** 16

**Explanation:** Max subarray sum is 16 of element (5, 4, 7)

Solution:

class Solution {

// arr: input array

// Function to find the sum of contiguous subarray with maximum sum.

long maxSubarraySum(int[] arr) {

// Your code here

long sum = 0;

long maxi = arr[0];

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

{

sum += arr[i];

maxi = Math.max(sum, maxi);

if(sum < 0)

{

sum = 0;

}

}

return maxi;

}

}

12.

**Kth Smallest**

Difficulty: **Medium** Accuracy: **35.17%**Submissions: **624K+**Points: **4**

Given an array **arr[]** and an integer **k** where k is smaller than the size of the array, the task is to find the **kth smallest** element in the given array. It is given that all array elements are distinct.

**Follow up:** Don't solve it using the inbuilt sort function.

**Examples:**

**Input:** arr[] = [7, 10, 4, 3, 20, 15], k = 3

**Output:** 7

**Explanation:** 3rd smallest element in the given array is 7.

**Input:** arr[] = [7, 10, 4, 20, 15], k = 4

**Output:** 15

**Explanation:** 4th smallest element in the given array is 15.

Solution:

class Solution {

public static int kthSmallest(int[] arr, int k) {

// Your code here

int res = 0;

Arrays.sort(arr);

res = arr[k - 1];

return res;

}

}

13.

**Sort 0s, 1s and 2s**

Difficulty: **Medium** Accuracy: **50.58%**Submissions: **675K+**Points: **4**

Given an array of size**n** containing only 0s, 1s, and 2s; sort the array in ascending order.

**Examples:**

**Input:** n = 5, arr[]= [0 2 1 2 0]

**Output:** 0 0 1 2 2

**Explanation:** 0s 1s and 2s are segregated

into ascending order.

**Input:** n = 3, arr[] = [0 1 0]

**Output:** 0 0 1

**Explanation:** 0s 1s and 2s are segregated

into ascending order.

Solution:

class Solution

{

public static void sort012(int a[], int n)

{

// code here

Arrays.sort(a);

/\*for(int i = 0; i < a.length; i++)

{

System.out.print(" "+a[i]);

}

\*/

}

}

14.

**Majority Element**

Difficulty: **Medium**Accuracy: **27.82%**Submissions: **636K+**Points: **4**

Given an array **arr**. Find the majority element in the array. If no majority exists, return -1.

A majority element in an array of size n is an element that appears **strictly**more than**n/2 times** in the array.

**Examples:**

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

**Output:** 3

**Explanation:** Since, 3 is present more than n/2 times, so it is the majority element.

**Input:** arr[] = [7]

**Output:** 7

**Explanation:** Since, 7 is single element and present more than n/2 times, so it is the majority element.

**Input:** arr[] = [2, 13]

**Output:** -1

**Explanation:** Since, no element is present more than n/2 times, so there is no majority element.

Solution:

Approach 1

class Solution

{

static int majorityElement(int a[], int size)

{

// your code here

if(size == 1)

{

return a[0];

}

Arrays.sort(a);

int count = 1;

for(int i = 0; i < size - 1; i++)

{

if(a[i] == a[i + 1])

{

count ++;

}

else

{

count = 1;

}

if(count > size / 2)

{

return a[i];

}

}

return -1;

}

}

Note: This solution takes n class complexity

Approach 02:

Using HashMap

class Solution

{

static int majorityElement(int a[], int size)

{

// your code here

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

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

{

map.put(a[i], map.getOrDefault(a[i], 0) + 1);

}

for(Integer key : map.keySet())// for loop is directly used for keys

{

if(map.get(key) > size / 2)

{

return key;

}

}

return -1;

}

}

15.

**Minimize the Heights II**

Difficulty: **Medium** Accuracy: **15.06%**Submissions: **577K+**Points: **4**

Given an array **arr[]** denoting heights of **N** towers and a positive integer **K.**

For **each**tower, you must perform **exactly one** of the following operations **exactly once**.

* **Increase**the height of the tower by **K**
* **Decrease**the height of the tower by **K**

Find out the **minimum**possible difference between the height of the shortest and tallest towers after you have modified each tower.

You can find a slight modification of the problem [here](https://practice.geeksforgeeks.org/problems/minimize-the-heights-i/1/).  
**Note:** It is **compulsory**to increase or decrease the height by K for each tower. **After** the operation, the resultant array should **not** contain any **negative integers**.

**Example 1:**

**Input:**

K = 2, N = 4

Arr[] = {1, 5, 8, 10}

**Output:**

5

**Explanation:**

The array can be modified as   
{1+k, 5-k, 8-k, 10-k} = {3, 3, 6, 8}.   
The difference between

the largest and the smallest is 8-3 = 5.

**Example 2:**

**Input:**

K = 3, N = 5

Arr[] = {3, 9, 12, 16, 20}

**Output:**

11

**Explanation:**

The array can be modified as  
{3+k, 9+k, 12-k, 16-k, 20-k} -> {6, 12, 9, 13, 17}.   
The difference between

the largest and the smallest is 17-6 = 11.

Solution:

class Solution {

int getMinDiff(int[] arr, int n, int k) {

// code here

Arrays.sort(arr);

int diff = arr[n - 1] - arr[0];

int smallest = arr[0] + k;

int largest = arr[n - 1] - k;

for(int i = 0; i < arr.length - 1; i++)

{

int mini = Math.min(smallest, arr[i + 1] - k);//to get the minimum from the next element

int maxi = Math.max(largest, arr[i] + k);//to get the maximum from the current element

if(mini < 0)//skip if the minimum is less than 0

{

continue;

}

diff = Math.min(diff, maxi - mini);

}

return diff;

}

}

Level 2 Pending (Medium + Hard)

2.Cracking Coding Textbook Section

Question 1

Is Unique: Implement an algorithm to determine if a string has all unique characters. What if you cannot use additional data structures?

Answer :

import java.util.\*;

public class IsUnique {

    static boolean testunique(String str)

    {

        char[] chars = str.toUpperCase().toCharArray();

        HashSet<Character> set = new HashSet<>();

        for(int c = 0; c <= chars.length - 1; c++)

        {

            if(!set.add(str.charAt(c)))

            {

                return false;

            }

        }

        return true;

    }

    public static void main(String[] args) {

        Scanner read = new Scanner(System.in);

        System.out.println("Enter a string : ");

        String str = read.nextLine();

        System.out.println(""+testunique(str));

    }

}

Question 2

My try :

import java.util.\*;

public class CheckPermutation {

    public static boolean testPermutation(String str1, String str2)

    {

        char[] chars1 = str1.toUpperCase().toCharArray();

        Arrays.sort(chars1);

        char[] chars2 = str2.toUpperCase().toCharArray();

        Arrays.sort(chars2);

        if(chars1.length == chars2.length)

        {

            for(int i = 0; i < chars1.length - 1; i++)

            {

                if(chars1[i] != chars2[i])

                {

                    return false;

                }

            }

            return true;

        }

        else

        {

            return false;

        }

    }

    public static void main(String[] args) {

        Scanner read = new Scanner(System.in);

        System.out.println("Enter string 1 : ");

        String str1 = read.nextLine();

        System.out.println("Enter string 2 : ");

        String str2  = read.nextLine();

        System.out.println("Permutation is : "+testPermutation(str1,str2));

    }

}

Time Complexity: (nlogn)

Feasible Solution:

import java.util.\*;

public class CheckPermutation {

    public static boolean testPermutation(String str1, String str2)

    {

        char[] x = str1.toCharArray();

        char[] y = str2.toCharArray();

        if(x.length != y.length)

        {

            return false;

        }

        Arrays.sort(x);

        Arrays.sort(y);

        return Arrays.equals(x, y);

    }

    public static void main(String[] args) {

        Scanner read = new Scanner(System.in);

        System.out.println("Enter string 1 : ");

        String str1 = read.nextLine();

        System.out.println("Enter string 2 : ");

        String str2  = read.nextLine();

        System.out.println("Permutation is : "+testPermutation(str1,str2));

    }

}

Level 2 Pending questions from textbook

Topic: Recursion (Level 1)