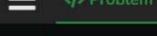


```
Java (1.8) 🕶
                                                Start Timer ()
                         Average Time: 20m
   class Solution {
        public static int largest(int[] arr) {
             int max=Integer.MIN_VALUE;
             for(int i=0;i<arr.length;i++){</pre>
56
                 if(arr[i]>max){
                     max=arr[i];
            return max;
                                                       Custom Input
                                                                     Compile & Run
                                                                                       Submit
```



(1) Submissions

Comments

雅

Array Search □

Difficulty: Basic Accuracy: 40.95% Submissions: 400K+

Points: 1

Given an array, arr of n integers, and an integer element x, find whether element x is present in the array. Return the index of the first occurrence of x in the array, or -1 if it doesn't exist.

Examples:

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

Output: 2

Explanation: There is one test case with array as [1, 2, 3 4] and element to be

searched as 3. Since 3 is present at index 2, the output is 2.

Input: arr[] = [10, 8, 30, 4, 5], x = 5

Output: 4

Explanation: For array [1, 2, 3, 4, 5], the element to be searched is 5 and it is at

index 4. So, the output is 4.

Input: arr[] = [10, 8, 30], x = 6

Output: -1

Explanation: The element to be searched is 6 and its not present, so we return -1.

Expected Time Complexity: O(n).

Expected Auxiliary Space: O(1).

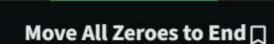
日 B ® U X Average Time: 15m Java (1.8) class Solution { static int search(int arr[], int x) { for(int i=0;i<arr.length;i++){</pre> if (arr[i]==x){ return i; return -1; 44 45 40

○ Comments

ŵ







</>
Problem

Difficulty: Easy Accuracy: 45.51%

Submissions: 264K+

Points: 2

(1) Submissions

You are given an array arr[] of non-negative integers. Your task is to move all the zeros in the array to the right end while maintaining the relative order of the non-zero elements. The operation must be performed in place, meaning you should not use extra space for another array.

Examples:

Input: arr[] = [1, 2, 0, 4, 3, 0, 5, 0]

Output: [1, 2, 4, 3, 5, 0, 0, 0]

Explanation: There are three 0s that are moved to the end.

Input: arr[] = [10, 20, 30]

Output: [10, 20, 30]

Explanation: No change in array as there are no Os.

Input: arr[] = [0, 0]

Output: [0, 0]

Explanation: No change in array as there are all Os.

Constraints:

 $1 \le arr.size() \le 10^5$

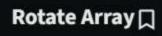
 $0 \le arr[i] \le 10^5$

```
Java (1.8)
                        Average Time: 15m
                                               Start Timer ()
     // } Driver Code Ends
29
30
32
   class Solution {
33
        void pushZerosToEnd(int[] arr) {
34
            int lastNonZeroIndex = 0; // Points to the position of the last non-zero e
35
36
37
            for (int i = 0; i < arr.length; i++) {
38
                if (arr[i] != 0) {
39
40
                    int temp = arr[lastNonZeroIndex];
41
                    arr[lastNonZeroIndex] = arr[i];
42
                    arr[i] = temp;
43
44
45
                    lastNonZeroIndex++;
46
47
48
49
50
51
```

Custom Input

O Comments

ŵ



</>
Problem

Difficulty: Medium Accuracy: 37.06%

Submissions: 429K+

Points: 4

(1) Submissions

Given an array arr[]. Rotate the array to the left (counter-clockwise direction) by d steps, where d is a positive integer. Do the mentioned change in the array in place.

Note: Consider the array as circular.

Examples:

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

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

Explanation: when rotated by 2 elements, it becomes 3 4 5 1 2.

Input: arr[] = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20], d = 3

Output: [8, 10, 12, 14, 16, 18, 20, 2, 4, 6]

Explanation: when rotated by 3 elements, it becomes 8 10 12 14 16 18 20 2 4 6.

Input: arr[] = [7, 3, 9, 1], d = 9

Output: [3, 9, 1, 7]

Explanation: when we rotate 9 times, we'll get 3 9 1 7 as resultant array.

Constraints:

1 <= arr.size(), d <= 10⁵ $0 \le arr[i] \le 10^5$



Average Time: 20m







```
O M
```

```
( ) Driver Code Ends
41
42
43
44
   class Solution {
45
46
        static void rotateArr(int arr[], int d) {
47
            int n = arr.length;
48
49
50
            d = d % n; // To handle cases where d is larger than the array size
51
52
53
            int[] temp = new int[n];
54
55
            // Copy the elements from index d to the end into the temporary array
56
57
            for (int i = 0; i < n - d; i++) {
                temp[i] = arr[i + d];
58
59
60
61
            for (int i = 0; i < d; i++) {
62
                temp[n - d + i] = arr[i];
63
64
65
66
            System.arraycopy(temp, 0, arr, 0, n);
67
68
69
70
```

Custom Input

Submit







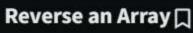
Compile & Run

Custom Input

Submit







</>
Problem

Editorial

(Submissions

O Comments

ŵ

Difficulty: Easy Accuracy: 55.32% Submissions: 103K+

Points: 2

You are given an array of integers arr[]. Your task is to reverse the given array.

Examples:

Input: arr = [1, 4, 3, 2, 6, 5]

Output: [5, 6, 2, 3, 4, 1]

Explanation: The elements of the array are 1 4 3 2 6 5. After reversing the array, the first element goes to the last position, the second element goes to the second last position and so on. Hence, the answer is 5 6 2 3 4 1.

Input: arr = [4, 5, 2]

Output: [2, 5, 4]

Explanation: The elements of the array are 4 5 2. The reversed array will be 2 5 4.

Input: arr = [1]

Output: [1]

Explanation: The array has only single element, hence the reversed array is same as

the original.

Constraints:

1<=arr.size()<=10⁵ 0<=arr[i]<=10⁵



```
Average Time: 5m
                                                                                       O 28
     Java (1.8)
     1: 1 // ) Driver Code Ends
    27
    28
    29 class Solution {
            public void reverseArray(int arr[]) {
    30
                int start = 0;
    31
                int end = arr.length - 1;
    32
    33
    34
                while (start < end) {
    35
    36
                    int temp = arr[start];
    37
                    arr[start] = arr[end];
    38
                    arr[end] = temp;
    39
    40
    41
                    start++;
    42
                    end--;
    43
    44
40
    45
    46
    47
```

雅

Accuracy: 64.46%

Submissions: 346+

Points: 2

Given an array, arr[], generate all possible subarrays using recursion and return them as a vector of vectors.

The subarrays must be returned in the following order:

- Subarrays starting from the first element, followed by subarrays starting from the second element, and so on.
 - 2. For each starting index, subarrays should be in increasing length.

Examples:

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

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

Explanation: Starting with the first element, we generate subarrays [1], [1, 2], and [1, 2, 3]. Then, starting from the second element, we get [2] and [2, 3]. Finally, starting from the third element, we only get [3].

Input: arr[] = [1, 2]

Output: [[1], [1, 2], [2]]

Explanation: Starting with the first element, we generate subarrays [1] and [1, 2]. Then, starting from the second element, we get [2].

Input: arr[] = [1, 1]

Output: [[1], [1, 1], [1]]

Explanation: Starting with the first element, we generate subarrays [1] and [1, 1] (including both elements). Starting from the second element, we only get the subarray [1].

```
Java (1.8)
                           O Start Timer ()
       // ] Driver Code Ends
   44
   45
   46
   47
   48
   49 class Solution {
           public List<List<Integer>> getSubArrays(int[] arr) {
   50
               List<List<Integer>> result = new ArrayList<>();
   51
               for (int i = 0; i < arr.length; i++) {
   52
                    for (int j = i; j < arr.length; j++) {
   53
                       List<Integer> subarray = new ArrayList<>();
   54
                       for (int k = i; k \le j; k++) {
   55
                           subarray.add(arr[k]);
   56
   57
                       result.add(subarray);
   58
   59
   60
   61
D
   62
               return result;
   63
   64
   65
```

Custom Input

Compile & Run

Remove Duplicates Sorted Array □

Difficulty: Easy Accuracy: 38.18% Submissions: 264K+ Points: 2

Given a **sorted** array **arr.** Return the size of the modified array which contains only distinct elements. *Note:*

(1) Submissions

- 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:

Problem

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.

Constraints:

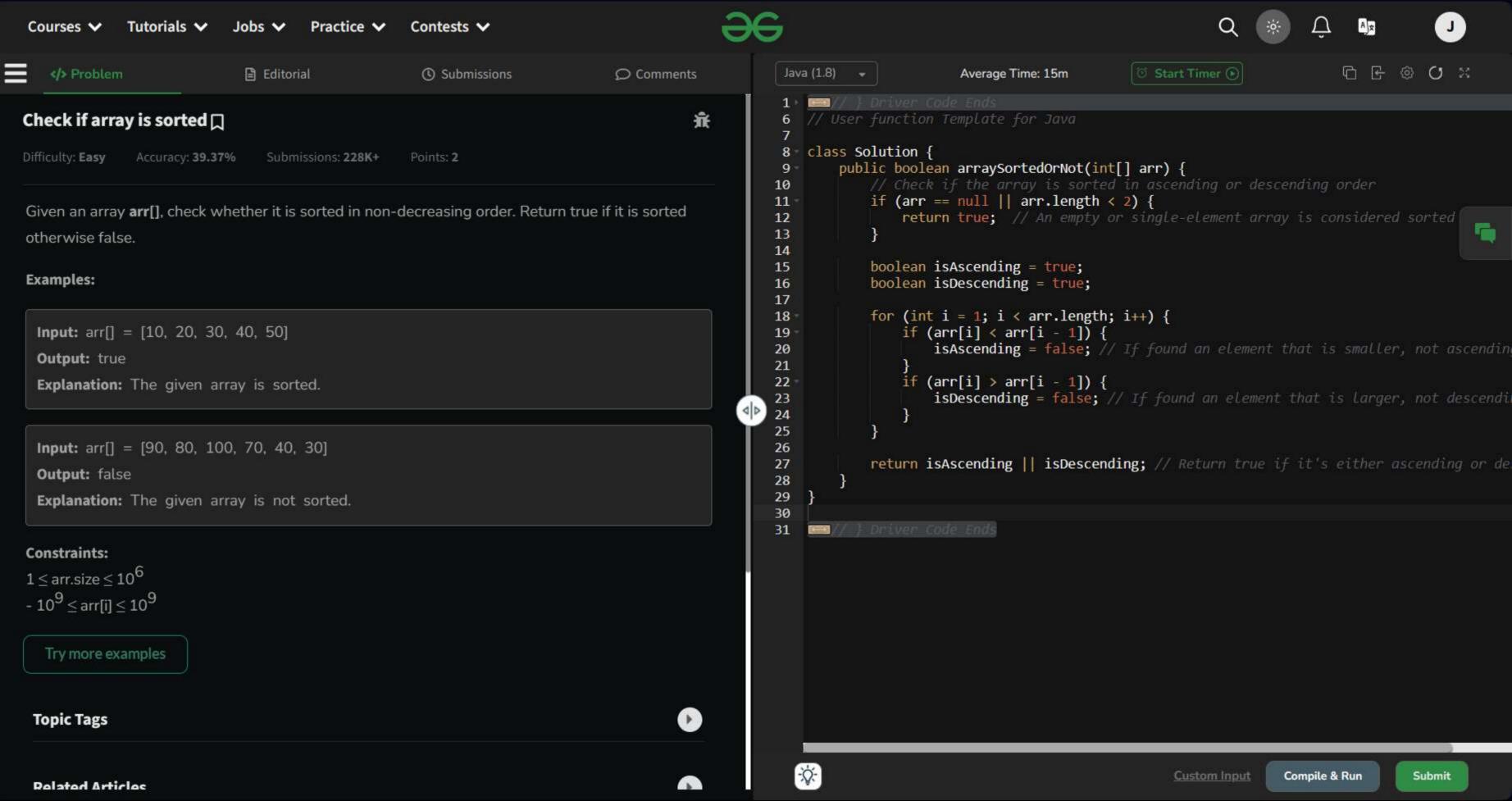
 $1 \le \text{arr.size()} \le 10^5$ $1 \le a_i \le 10^6$

Try more examples

ŵ

O Comments

```
Java (1.8)
                        Average Time: 20m
24
25
    // User function Template for Java
27 class Solution {
        // Function to remove duplicates from the given array
28
        public int removeDuplicates(int[] arr) {
29
            if (arr == null || arr.length == 0) {
30
                return 0; // Return 0 for empty array
31
32
33
            int index = 1; // Start from the second element
34
35
            for (int i = 1; i < arr.length; i++) {
36
                if (arr[i] != arr[i - 1]) { // Check if current element is different from pr
37
                    arr[index] = arr[i];
38
                    index++:
39
40
41
42
            return index; // Return the length of the array with unique elements
43
44
45
46
47
```



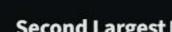












</>
Problem

Editorial

Submissions

Comments

ŵ

Second Largest □

Difficulty: Easy Accuracy: 26.72%

Submissions: 911K+

Points: 2

Given an array of **positive** integers **arr**[], return the **second largest** element from the array. If the second largest element doesn't exist then return **-1**.

Note: The second largest element should not be equal to the largest element.

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, 5, 10]

Output: 5

Explanation: The largest element of the array is 10 and the second largest element is 5.

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

Output: -1

Explanation: The largest element of the array is 10 and the second largest element

does not exist.

Constraints:

2 - 200 012011 - 105

```
Java (1.8)
                           Average Time: 15m
       ( ) Driver Code Ends
    25
    26
       class Solution {
            public int getSecondLargest(int[] arr) {
    29
                int largest = Integer.MIN_VALUE;
    30
                int secondLargest = Integer.MIN VALUE;
    31
                for (int i = 0; i < arr.length; i++) {
    32
                    if (arr[i] > largest) {
   33
                        secondLargest = largest;
    34
                        largest = arr[i];
    35
                    } else if (arr[i] > secondLargest && arr[i] != largest) {
    36
                        secondLargest = arr[i];
    37
    38
    39
    40
   41
                if (secondLargest == Integer.MIN VALUE) {
    42
1 D
                    return -1; // Or any other value to indicate no second largest
   43
   44
   45
                return secondLargest;
   46
    47
    48
    49
```