



</> Problem

Editorial

Submissions

Comments

Largest Element in Array



Difficulty: Basic

Accuracy: 67.48%

Submissions: 381K+

Points: 1

Given an array `arr[]`. The task is to find the **largest** element and return it.

Examples:

Input: `arr[] = [1, 8, 7, 56, 90]`

Output: 90

Explanation: The largest element of the given array is 90.

Input: `arr[] = [5, 5, 5, 5]`

Output: 5

Explanation: The largest element of the given array is 5.

Input: `arr[] = [10]`

Output: 10

Explanation: There is only one element which is the largest.

Constraints:

$1 \leq \text{arr.size()} \leq 10^6$

$0 \leq \text{arr}[i] \leq 10^6$

[Try more examples](#)

Java (1.8)

Average Time: 20m

Start Timer



```
1  
50  
51  
52  
53 class Solution {  
54     public static int largest(int[] arr) {  
55         int max=Integer.MIN_VALUE;  
56         for(int i=0;i<arr.length;i++){  
57             if(arr[i]>max){  
58                 max=arr[i];  
59             }  
60         }  
61     }  
62     return max;  
63     // code here  
64 }  
65 }  
66
```

[Custom Input](#)

Compile & Run

Submit



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:** $O(1)$.**Constraints:**

Java (1.8)

Average Time: 15m

[Start Timer](#)

```
1 // } Driver Code Ends
30
31
32 class Solution {
33
34     static int search(int arr[], int x) {
35         for(int i=0;i<arr.length;i++){
36             if (arr[i]==x){
37                 return i;
38             }
39         }
40         return -1;
41
42         // Your code here
43     }
44 }
45
```

[Custom Input](#)[Compile & Run](#)[Submit](#)



Move All Zeroes to End



Difficulty: Easy

Accuracy: 45.51%

Submissions: 264K+

Points: 2

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 0s.**Input:** arr[] = [0, 0]**Output:** [0, 0]**Explanation:** No change in array as there are all 0s.

Constraints:

$$1 \leq \text{arr.size()} \leq 10^5$$

$$0 \leq \text{arr}[i] \leq 10^5$$

Java (1.8)

Average Time: 15m

Start Timer



```
1 // } Driver Code Ends
29
30
31 // User function Template for Java
32
33 class Solution {
34     void pushZerosToEnd(int[] arr) {
35         int lastNonZeroIndex = 0; // Points to the position of the last non-zero element
36
37         // Traverse the array
38         for (int i = 0; i < arr.length; i++) {
39             if (arr[i] != 0) {
40                 // Swap the current element with the element at lastNonZeroIndex
41                 int temp = arr[lastNonZeroIndex];
42                 arr[lastNonZeroIndex] = arr[i];
43                 arr[i] = temp;
44
45                 // Increment the lastNonZeroIndex
46                 lastNonZeroIndex++;
47             }
48         }
49     }
50 }
51
```



Custom Input

Compile & Run

Submit



Rotate Array



Difficulty: Medium Accuracy: 37.06% Submissions: 429K+ Points: 4

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:

$1 \leq arr.size(), d \leq 10^5$

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

Java (1.8)

Average Time: 20m

🕒 Start Timer



```
1 // } Driver Code Ends
41
42
43 // User function Template for Java
44
45 class Solution {
46     // Function to rotate an array by d elements in counter-clockwise direction
47     static void rotateArr(int arr[], int d) {
48         int n = arr.length;
49
50         // Handle the case when d is greater than the array length
51         d = d % n; // To handle cases where d is larger than the array size
52
53         // Create a temporary array to store rotated elements
54         int[] temp = new int[n];
55
56         // Copy the elements from index d to the end into the temporary array
57         for (int i = 0; i < n - d; i++) {
58             temp[i] = arr[i + d];
59         }
60
61         // Copy the first d elements into the temporary array
62         for (int i = 0; i < d; i++) {
63             temp[n - d + i] = arr[i];
64         }
65
66         // Copy the elements back from the temporary array to the original array
67         System.arraycopy(temp, 0, arr, 0, n);
68     }
69 }
70
```

[Custom Input](#)

Compile & Run

Submit



</> Problem

Editorial

Submissions

Comments

Reverse an Array



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:

 $1 \leq \text{arr.size()} \leq 10^5$ $0 \leq \text{arr}[i] \leq 10^5$

Java (1.8)

Average Time: 5m

Start Timer



```
1 // } Driver Code Ends
27
28
29 class Solution {
30     public void reverseArray(int arr[]) {
31         int start = 0;
32         int end = arr.length - 1;
33
34         // Swap elements until start and end pointers meet
35         while (start < end) {
36             // Swap the elements at start and end
37             int temp = arr[start];
38             arr[start] = arr[end];
39             arr[end] = temp;
40
41             // Move the pointers towards each other
42             start++;
43             end--;
44         }
45     }
46 }
47
```



Custom Input

Compile & Run

Submit



Generating All Subarrays



Difficulty: Easy

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:

1. 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]`.

1 // } Driver Code Ends

44

45

46 // User function Template for Java

47

48

49 class Solution {

50 public List<List<Integer>> getSubArrays(int[] arr) {

51 List<List<Integer>> result = new ArrayList<>();

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

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

54 List<Integer> subarray = new ArrayList<>();

55 for (int k = i; k <= j; k++) {

56 subarray.add(arr[k]);

57 }

58 result.add(subarray);

59 }

60 }

61

62 return result;

63 }

64 }

65

[Custom Input](#)

Compile & Run

Submit



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. 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]

Explantation: As the array does not contain any duplicates so you should return 3.

Constraints:

$$1 \leq \text{arr.size()} \leq 10^5$$

$$1 \leq a_i \leq 10^6$$

[Try more examples](#)

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

[Custom Input](#)

Compile & Run

Submit



Check if array is sorted



Difficulty: Easy

Accuracy: 39.37%

Submissions: 228K+

Points: 2

Given an array `arr[]`, check whether it is sorted in non-decreasing order. Return true if it is sorted otherwise false.

Examples:

Input: `arr[] = [10, 20, 30, 40, 50]`**Output:** true**Explanation:** The given array is sorted.**Input:** `arr[] = [90, 80, 100, 70, 40, 30]`**Output:** false**Explanation:** The given array is not sorted.

Constraints:

 $1 \leq \text{arr.size} \leq 10^6$ $-10^9 \leq \text{arr}[i] \leq 10^9$ [Try more examples](#)

Topic Tags



Related Articles



Java (1.8)

Average Time: 15m

Start Timer



```
1 // } Driver Code Ends
2 // User function Template for Java
3
4 class Solution {
5     public boolean arraySortedOrNot(int[] arr) {
6         // Check if the array is sorted in ascending or descending order
7         if (arr == null || arr.length < 2) {
8             return true; // An empty or single-element array is considered sorted
9         }
10
11         boolean isAscending = true;
12         boolean isDescending = true;
13
14         for (int i = 1; i < arr.length; i++) {
15             if (arr[i] < arr[i - 1]) {
16                 isAscending = false; // If found an element that is smaller, not ascending
17             }
18             if (arr[i] > arr[i - 1]) {
19                 isDescending = false; // If found an element that is larger, not descending
20             }
21         }
22
23         return isAscending || isDescending; // Return true if it's either ascending or descending
24     }
25 }
26
27 // } Driver Code Ends
```

[Custom Input](#)

Compile & Run

Submit



</> Problem

Editorial

Submissions

Comments

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:

$2 \leq \text{arr.size()} \leq 10^5$

Java (1.8)

Average Time: 15m

Start Timer



```
1 // } Driver Code Ends
25
26
27 // User function Template for Java
28 class Solution {
29     public int getSecondLargest(int[] arr) {
30         int largest = Integer.MIN_VALUE;
31         int secondLargest = Integer.MIN_VALUE;
32         for (int i = 0; i < arr.length; i++) {
33             if (arr[i] > largest) {
34                 secondLargest = largest;
35                 largest = arr[i];
36             } else if (arr[i] > secondLargest && arr[i] != largest) {
37                 secondLargest = arr[i];
38             }
39         }
40
41         // Handle the case when all elements are the same
42         if (secondLargest == Integer.MIN_VALUE) {
43             return -1; // Or any other value to indicate no second largest
44         }
45
46         return secondLargest;
47     }
48 }
49
```

[Custom Input](#)

Compile & Run

Submit