

Name: Anisha Raj (2CSE8)

Roll no: 2410031455

You are given an array of integers `arr[]`. You have to **reverse** the given array.

Note: Modify the array in place.

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]`.

The screenshot displays a coding platform interface for the 'Reverse Array' problem. The left sidebar contains the problem description, which includes the input, output, and explanation for the example array `[1, 4, 3, 2, 6, 5]`. The main editor area shows a Java solution for reversing the array in place using a two-pointer technique. The code is as follows:

```
1 class Solution {
2     public void reverseArray(int[] arr) {
3         int n = arr.length;
4
5         for (int i = 0; i < n / 2; i++) {
6             int temp = arr[i];
7             arr[i] = arr[n - 1 - i];
8             arr[n - 1 - i] = temp;
9         }
10    }
11 }
12
```

The bottom of the interface features buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

Name: Anisha Raj (2CSE8)

Roll no: 2410031455

Given an array `arr[]`. Your task is to find the **minimum** and **maximum** elements in the array.

Examples:

Input: `arr[] = [1, 4, 3, 5, 8, 6]`

Output: `[1, 8]`

Explanation: minimum and maximum elements of array are 1 and 8.

Link: <https://www.geeksforgeeks.org/problems/find-minimum-and-maximum-1428/1>

The screenshot shows a coding IDE interface with a dark theme. On the left, there's a sidebar with a search bar and navigation tabs: 'Problem', 'Editorial', 'Submissions', and 'Comments'. Below these, there's an 'Output Window' and 'Compilation Results' section. The 'Compilation Results' section shows 'Compilation Completed' for 'Case 1'. The input is 'arr[] = 1 4 3 5 4 8 6' and the output is '[1, 8]'. The main editor area on the right shows a Java code snippet for finding the minimum and maximum elements in an array. The code is as follows:

```
1 import java.util.ArrayList;
2
3 class Solution {
4     public ArrayList<Integer> getMinMax(int[] arr) {
5
6         int min = arr[0];
7         int max = arr[0];
8         for (int i = 1; i < arr.length; i++) {
9             if (arr[i] < min) {
10                 min = arr[i];
11             }
12             if (arr[i] > max) {
13                 max = arr[i];
14             }
15         }
16         ArrayList<Integer> result = new ArrayList<>();
17         result.add(min);
18         result.add(max);
19         return result;
20     }
21 }
22
23
```

At the bottom of the IDE, there are buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

Name: Anisha Raj (2CSE8)

Roll no: 2410031455

Given an integer array `arr[]` and an integer `k`, your task is to find and return the **`kth smallest`** element in the given array.

Note: The `kth` smallest element is determined based on the sorted order of the array.

Examples :

Input: `arr[] = [10, 5, 4, 3, 48, 6, 2, 33, 53, 10]`, `k = 4`

Output: 5

Explanation: 4th smallest element in the given array is 5.

The screenshot displays a coding environment with a dark theme. On the left, the 'Output Window' is active, showing 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. It indicates 'Compilation Completed' for 'Case 1'. The input fields show 'arr[] = 3 5 4 2 9' and 'k = 3'. The 'Your Output' field contains '4', which matches the 'Expected Output'.

On the right, the code editor shows a Java solution:

```
1 import java.util.Arrays;
2
3 class Solution {
4     public static int kthSmallest(int arr[], int k) {
5         Arrays.sort(arr);
6         return arr[k - 1];
7     }
8 }
9
```

At the bottom right, there are buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

Name: Anisha Raj (2CSE8)

Roll no: 2410031455

You are given two arrays **a[]** and **b[]**, return the **Union** of both the arrays in any order.

The **Union** of two arrays is a collection of all **distinct elements** present in either of the arrays. If an element appears more than once in one or both arrays, it should be included **only once** in the result.

Note: Elements of **a[]** and **b[]** are not necessarily distinct.

Note that, You can return the Union in any order but the driver code will print the result in **sorted order** only.

Examples:

Input: a[] = [1, 2, 3, 2, 1], b[] = [3, 2, 2, 3, 3, 2]

Output: [1, 2, 3]

Explanation: Union set of both the arrays will be 1, 2 and 3.

The screenshot displays a coding environment with a dark theme. On the left, the 'Output Window' is open, showing 'Compilation Results' for 'Case 1'. It indicates 'Compilation Completed' and provides the input arrays: a[] = [1 2 3 2 1] and b[] = [3 2 2 3 3 2]. The 'Expected Output' is [1, 2, 3]. The main editor on the right shows a Java solution for the problem. The code imports java.util.* and defines a class Solution with a static method findUnion. This method uses a HashSet to store unique elements from both arrays and returns them as a sorted ArrayList.

```
1 import java.util.*;
2
3 class Solution {
4     public static ArrayList<Integer> findUnion(int[] a, int[] b) {
5
6         HashSet<Integer> set = new HashSet<>();
7
8         for (int x : a) set.add(x);
9         for (int x : b) set.add(x);
10
11         return new ArrayList<>(set);
12     }
13 }
14
```

At the bottom right, there are buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

Name: Anisha Raj (2CSE8)

Roll no: 2410031455

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

Examples:

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

The screenshot displays a coding platform interface with a dark theme. The top navigation bar includes a search bar, a menu icon, and links for Courses, Tutorials, Practice, and Jobs. The main workspace is divided into three sections: an Output Window on the left, a code editor in the center, and a bottom status bar. The Output Window shows 'Compilation Results' for 'Case 1' with 'Custom Input'. The input is 'arr[] = 1 8 7 56 90', the 'Your Output' is '90', and the 'Expected Output' is '90'. The code editor shows a Java solution for finding the largest element in an array. The code is as follows:

```
1 class Solution {
2     public static int largest(int[] arr) {
3
4         int max = arr[0];
5
6         for (int i = 1; i < arr.length; i++) {
7             if (arr[i] > max) {
8                 max = arr[i];
9             }
10        }
11
12        return max;
13    }
14 }
15
```

The bottom status bar includes a 'Custom Input' button, a 'Compile & Run' button, and a 'Submit' button.

Name: Anisha Raj (2CSE8)

Roll no: 2410031455

Given an array **arr**, rotate the array by one position in clockwise direction.

Examples:

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

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

Explanation: If we rotate arr by one position in clockwise 5 come to the front and remaining those are shifted to the end.

The screenshot shows a coding platform interface with a dark theme. On the left, the 'Output Window' is open, displaying 'Compilation Results' for 'Custom Input'. It shows 'Compilation Completed' for 'Case 1'. The input is '1 2 3 4 5', the user's output is '5 1 2 3 4', and the expected output is also '5 1 2 3 4'. On the right, the code editor shows a Java solution for rotating an array. The code is as follows:

```
1 class Solution {
2     public void rotate(int[] arr) {
3
4         int n = arr.length;
5         int last = arr[n - 1];
6
7         for (int i = n - 1; i > 0; i--) {
8             arr[i] = arr[i - 1];
9         }
10
11         arr[0] = last;
12     }
13 }
14
```

At the bottom right, there are buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

Name: Anisha Raj (2CSE8)

Roll no: 2410031455

You are given an integer array **arr[]**. You need to find the **maximum** sum of a subarray (containing at least one element) in the array **arr[]**.

Note : A **subarray** is a continuous part of an array.

Examples:

Input: arr[] = [2, 3, -8, 7, -1, 2, 3]

Output: 11

Explanation: The subarray [7, -1, 2, 3] has the largest sum 11.

The screenshot shows a coding IDE interface. On the left, the 'Output Window' is open, displaying 'Compilation Results' for 'Custom Input'. It shows 'Compilation Completed' for 'Case 1'. The input is 'arr[] = 1 2 3 -2 5', the 'Your Output' is '9', and the 'Expected Output' is '9'. On the right, the code editor shows a Java solution for the 'Maximum Subarray Sum' problem. The code is as follows:

```
1 class Solution {
2     int maxSubarraySum(int[] arr) {
3
4         int currentSum = arr[0];
5         int maxSum = arr[0];
6
7         for (int i = 1; i < arr.length; i++) {
8
9             currentSum = Math.max(arr[i], currentSum + arr[i]);
10            maxSum = Math.max(maxSum, currentSum);
11        }
12
13        return maxSum;
14    }
15 }
16
```

At the bottom right, there are buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

Name: Anisha Raj (2CSE8)

Roll no: 2410031455

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: nums = [1,3,5,6], target = 5

Output: 2

The screenshot shows the LeetCode interface for problem 35, "Search Insert Position". The problem description states: "Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order. You must write an algorithm with $O(\log n)$ runtime complexity." Examples provided are: Example 1 (Input: [1,3,5,6], target: 5, Output: 2), Example 2 (Input: [1,3,5,6], target: 2, Output: 1), and Example 3 (Input: [1,3,5,6], target: 7, Output: 4). Constraints include an array size of 1 to 10^4 and values between 0 and 10^4. A Java solution is shown in the code editor, implementing a binary search algorithm. The test results section shows three cases (Case 1, Case 2, Case 3) all marked as "Accepted" with a runtime of 0 ms. The input for the test cases is nums = [1,3,5,6] and target = 5.

35. Search Insert Position

Easy Topics Companies

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: nums = [1,3,5,6], target = 5
Output: 2

Example 2:

Input: nums = [1,3,5,6], target = 2
Output: 1

Example 3:

Input: nums = [1,3,5,6], target = 7
Output: 4

Constraints:

- 1 <= nums.length <= 10⁴
- 0 <= nums[i] <= 10⁴
- 0 <= target <= 10⁴

18.5K 413 288 Online

```
class Solution {
    public int searchInsert(int[] nums, int target) {
        int low = 0, high = nums.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (nums[mid] == target)
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

nums = [1,3,5,6]

target = 5

Name: Anisha Raj (2CSE8)

Roll no: 2410031455

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have **exactly one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

Input: `nums = [2,7,11,15]`, `target = 9`

Output: `[0,1]`

Explanation: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`.

The screenshot shows the LeetCode interface for the 'Two Sum' problem. On the left, the problem description is visible, including the input, output, and explanation for Example 1. The main area displays a Java solution using a HashMap. The code is as follows:

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int need = target - nums[i];
            if (map.containsKey(need)) {
                return new int[] { map.get(need), i };
            }
            map.put(nums[i], i);
        }
        return new int[] {};
    }
}
```

Below the code, the 'Testcase' section shows the input `9` and the output `[0,1]`, which matches the expected result.

Name: Anisha Raj (2CSE8)

You are given an array `arr[]` of non-negative numbers. Each number tells you the **maximum number of steps** you can jump forward from that position.

For example:

- If `arr[i] = 3`, you can jump to index `i + 1`, `i + 2`, or `i + 3` from position `i`.
- If `arr[i] = 0`, you **cannot jump forward** from that position.

Your task is to find the **minimum number of jumps** needed to move from the **first** position in the array to the **last** position.

Note: Return **-1** if you can't reach the end of the array.

Examples :

Input: `arr[] = [1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9]`

Output: 3

Explanation: First jump from 1st element to 2nd element with value 3. From here we jump to 5th element with value 9, and from here we will jump to the last.

The screenshot displays a coding environment with the following components:

- Top Bar:** Includes a search bar, navigation links for Courses, Tutorials, Practice, and Jobs, and user profile icons.
- Left Panel:**
 - Output Window:** Shows the problem statement and input/output details.
 - Compilation Results:** Indicates "Compilation Completed" for Case 1.
 - Input:** `arr[] = 1 3 5 8 9 2 6 7 6 8 9`
 - Your Output:** 3
 - Expected Output:** 3
- Right Panel:** Contains the Java code for the solution.

```
1 class Solution {
2     public int minJumps(int[] arr) {
3
4         int n = arr.length;
5
6         if (n <= 1) return 0;
7         if (arr[0] == 0) return -1;
8
9         int maxReach = arr[0];
10        int steps = arr[0];
11        int jumps = 1;
12
13        for (int i = 1; i < n; i++) {
14
15            if (i == n - 1)
16                return jumps;
17
18            maxReach = Math.max(maxReach, i + arr[i]);
19            steps--;
20
21            if (steps == 0) {
22                jumps++;
23
24                if (i >= maxReach)
25                    return -1;
26
27                steps = maxReach - i;
28            }
29        }
30
31        return -1;
32    }
33 }
```
- Bottom Bar:** Includes buttons for "Custom Input", "Compile & Run", and "Submit".