

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 shows a Java code editor interface. At the top, there are tabs for 'Problem', 'Editorial', 'Submissions', and 'Comments'. Below the tabs, it says 'Difficulty: Easy', 'Accuracy: 55.32%', 'Submissions: 267K+', 'Points: 2', and 'Average Time: 5m'. The main area contains the following Java code:

```
Java (21) Start Timer
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 }
```

On the left side of the editor, there is a sidebar with three examples. Each example includes input, output, and an explanation.

- Example 1:** 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].
- Example 2:** 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].
- Example 3:** Input: arr = [1]
Output: [1]
Explanation: The array has only single element, hence the reversed array is same as the original.

At the bottom right of the editor, there are 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-14081>

The screenshot shows a Java code editor on the GeeksforGeeks platform. The code is a solution for finding the minimum and maximum values in an array. It uses a for loop to iterate through the array, comparing each element with the current minimum and maximum values. If an element is less than the current minimum, it becomes the new minimum. If it is greater than the current maximum, it becomes the new maximum. Finally, it returns an ArrayList containing the minimum and maximum values.

```
import java.util.ArrayList;
class Solution {
    public ArrayList<Integer> getMinMax(int[] arr) {
        int min = arr[0];
        int max = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < min) {
                min = arr[i];
            }
            if (arr[i] > max) {
                max = arr[i];
            }
        }
        ArrayList<Integer> result = new ArrayList<>();
        result.add(min);
        result.add(max);
        return result;
    }
}
```

The input field contains the array [1, 4, 3, 5, 8, 6]. The output field shows the expected result [1, 8]. The code has been successfully compiled and run.

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 shows a Java code editor interface. The code in the editor is:

```
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 }
```

The interface includes a search bar, navigation tabs like 'Courses', 'Tutorials', 'Practice', and 'Jobs', and various tool icons. On the left, there's an 'Output Window' tab and a 'Compilation Results' section showing 'Compilation Completed'. The input fields show 'arr[] = 3 5 4 2 9' and 'k = 3'. The output field shows '4', which is the expected result.

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 shows a Java code editor interface. The code is as follows:

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

The code uses a HashSet to store unique elements from both arrays and then converts it back into an ArrayList. The interface includes tabs for "Output Window", "Compilation Results", and "Custom Input". The "Compilation Results" tab shows "Compilation Completed". The "Custom Input" section contains input fields for arrays **a[]** and **b[]**, and an expected output field. The "Output Window" is currently empty. At the bottom, 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 shows a LeetCode problem interface. The top navigation bar includes 'Three 99 Ending', 'Courses', 'Tutorials', 'Practice', 'Jobs', and user icons. The 'Practice' dropdown is open, showing 'Java (21)'. A 'Start Timer' button is visible. The main area has tabs for 'Problem', 'Editorial', 'Submissions', and 'Comments'. The 'Compilation Results' tab is selected, showing 'Custom Input' and 'Compilation Completed' status. Under 'Case 1', the input is 'arr[] = 1 8 7 56 90' and the output is '90'. The expected output is also '90'. On the right, the Java code is displayed:

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

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

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 LeetCode problem page for "Rotate Array". The Java code provided is:

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

The "Compilation Results" section shows "Case 1" passed. The "Input" field contains "1 2 3 4 5" and the "Your Output:" field also shows "1 2 3 4 5". The "Expected Output:" field shows "5 1 2 3 4".

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 the LeetCode problem editor interface. The top navigation bar includes 'Courses', 'Tutorials', 'Practice', and 'Jobs'. The main area displays a Java code snippet for solving the 'Maximum Subarray' problem. The code uses a dynamic programming approach with two variables, `currentSum` and `maxSum`, initialized to the first element of the array. It iterates through the array, updating `currentSum` to be the maximum of the current element or the sum of the current element and `currentSum`. It also updates `maxSum` to be the maximum of `maxSum` and `currentSum`. Finally, it returns `maxSum`. The code editor has tabs for 'Java (21)', 'Start Timer', and other language options. On the left, there's an 'Output Window' tab, 'Compilation Results' (which shows 'Compilation Completed'), and a 'Custom Input' section. The input field contains the array `[1, 2, 3, -2, 5]`. The output field shows the result `9`, and the expected output is also `9`. At the bottom right, there are buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

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

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 a LeetCode problem page for "35. Search Insert Position".

Problem Description: 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: 18.5K likes, 413 dislikes, 288 Online users.

Code:

```
1 class Solution {  
2     public int searchInsert(int[] nums, int target) {  
3         int low = 0, high = nums.length - 1;  
4  
5         while (low <= high) {  
6             int mid = low + (high - low) / 2;  
7  
8             if (nums[mid] == target)  
9                 return mid;  
10            else if (nums[mid] < target)  
11                low = mid + 1;  
12            else  
13                high = mid - 1;  
14        }  
15        return low;  
16    }  
17}
```

Test Result: Accepted. Runtime: 0 ms. Cases: 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 $\text{nums}[0] + \text{nums}[1] == 9$, we return `[0, 1]`.

The screenshot shows a programming environment with the following details:

- Problem List:** Shows a single problem entry for "1. Two Sum".
- Description:** Describes the problem: Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to target*. It states that each input has **exactly one solution** and no *same* element twice.
- Topics:** Includes "Easy", "Topics", "Companies", and "Hint".
- Input/Output:** Example 1: Input: `nums = [2,7,11,15]`, target = 9; Output: `[0,1]`. Explanation: Because $\text{nums}[0] + \text{nums}[1] == 9$, we return `[0, 1]`. Example 2: Input: `nums = [3,2,4]`, target = 6; Output: `[1,2]`. Example 3: Input: `nums = [3,3]`, target = 6; Output: `[0,1]`.
- Code:** Java code for the `Solution` class:

```
3  class Solution {
4      public int[] twoSum(int[] nums, int target) {
5          HashMap<Integer, Integer> map = new HashMap<>();
6
7          for (int i = 0; i < nums.length; i++) {
8
9              int need = target - nums[i];
10
11             if (map.containsKey(need)) {
12                 return new int[] { map.get(need), i };
13             }
14
15             map.put(nums[i], i);
16         }
17
18         return new int[] {};
19     }
20 }
```
- Test Result:** Shows a test case with input 9 and output [0,1].
- Statistics:** 67.1K likes, 1.8K comments, 2894 Online.

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.

```
1 class Solution {
2     public int minJumps(int[] arr) {
3         int n = arr.length;
4         if (n <= 1) return 0;
5         if (arr[0] == 0) return -1;
6         int maxReach = arr[0];
7         int steps = arr[0];
8         int jumps = 1;
9
10        for (int i = 1; i < n; i++) {
11            if (i == n - 1)
12                return jumps;
13            maxReach = Math.max(maxReach, i + arr[i]);
14            steps--;
15            if (steps == 0) {
16                jumps++;
17                if (i >= maxReach)
18                    return -1;
19                steps = maxReach - i;
20            }
21        }
22        return -1;
23    }
24 }
```