

EXPERIMENT LIST FOR PROGRAMMING ABILITY AND LOGIC BUILDING 1

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]

Approach (Two-Pointer Method)

Use two pointers:

Left → start of the array

right → end of the array

Swap arr[left] and arr[right]

Move left++ and right--

Stop when left >= right

The screenshot shows a programming environment with a dark theme. On the left, there's a sidebar with navigation links like 'Courses', 'Tutorials', 'Practice', and 'Jobs'. Below that is an 'Output Window' section with tabs for 'Compilation Results', 'Custom Input', and 'Y.O.G.I. (AI Bot)'. The main area displays a solved problem titled 'Problem Solved Successfully'. It shows 'Test Cases Passed' as 1115 / 1115, 'Attempts : Correct / Total' as 1 / 1, 'Accuracy' as 100%, and 'Time Taken' as 0.91. On the right, the code editor contains Java code for reversing an array:

```
1 class Solution {
2     static void reverseArray(int arr[]) {
3         // code here
4         int left = 0;
5         int right = arr.length - 1;
6
7         while (left < right) {
8             int temp = arr[left];
9             arr[left] = arr[right];
10            arr[right] = temp;
11            left++;
12            right--;
13        }
14    }
15
16    public static void main(String[] args) {
17        int[] arr = {1, 4, 3, 2, 6, 5};
18        reverseArray(arr);
19
20        for (int num : arr) {
21            System.out.print(num + " ");
22        }
23    }
24
25
26
27 }
```

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

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.

Approach

Initialize

min = arr[0]

max = arr[0]

Traverse the array from index 1

Update:

if arr[i] < min → update min

if arr[i] > max → update max

The screenshot shows a LeetCode problem submission page for the "Two Sum" problem. The code submitted is:

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

The submission was successful, with the message "Problem Solved Successfully". The test cases passed are 1111 / 1111, and the accuracy is 100%. The time taken was 0.34 seconds. The user's total score is 3.

Given an integer array arr[] and an integer k, your task is to find and return the k

th smallest element in the given array.

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

Examples :

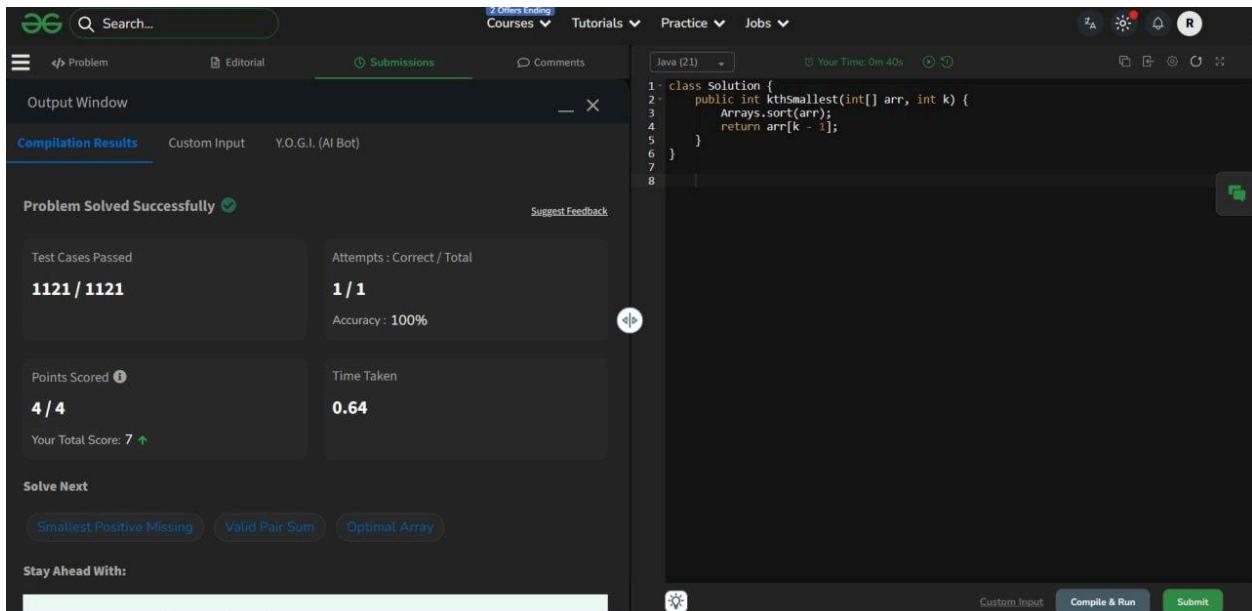
Input: arr[] = [10, 5, 4, 3, 48, 6, 2, 33, 53, 10], k = 4

Output: 5

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

Approach

Sort the array → the element at index k-1 is the answer.



The screenshot shows a programming environment with a dark theme. At the top, there are navigation links: 'Courses', 'Tutorials', 'Practice', and 'Jobs'. Below the navigation, there's a search bar and a 'Submissions' tab which is currently selected. On the left side, there's an 'Output Window' and tabs for 'Compilation Results', 'Custom Input', and 'Y.O.G.I. (AI Bot)'. A message 'Problem Solved Successfully' with a green checkmark is displayed. In the center, the code editor contains Java code to find the kth smallest element:

```
Java (21) Your Time: 0m 40s
1. class Solution {
2.     public int kthSmallest(int[] arr, int k) {
3.         Arrays.sort(arr);
4.         return arr[k - 1];
5.     }
6. }
```

Below the code editor, the results section shows: 'Test Cases Passed' (1121 / 1121), 'Attempts : Correct / Total' (1 / 1), 'Accuracy : 100%', 'Points Scored' (4 / 4), 'Time Taken' (0.64), and 'Your Total Score: 7'. There are also buttons for 'Solve Next' (with options like 'Smallest Positive Missing', 'Valid Pair Sum', 'Optimal Array') and 'Stay Ahead With:' (with a placeholder box). At the bottom right, there are buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

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.

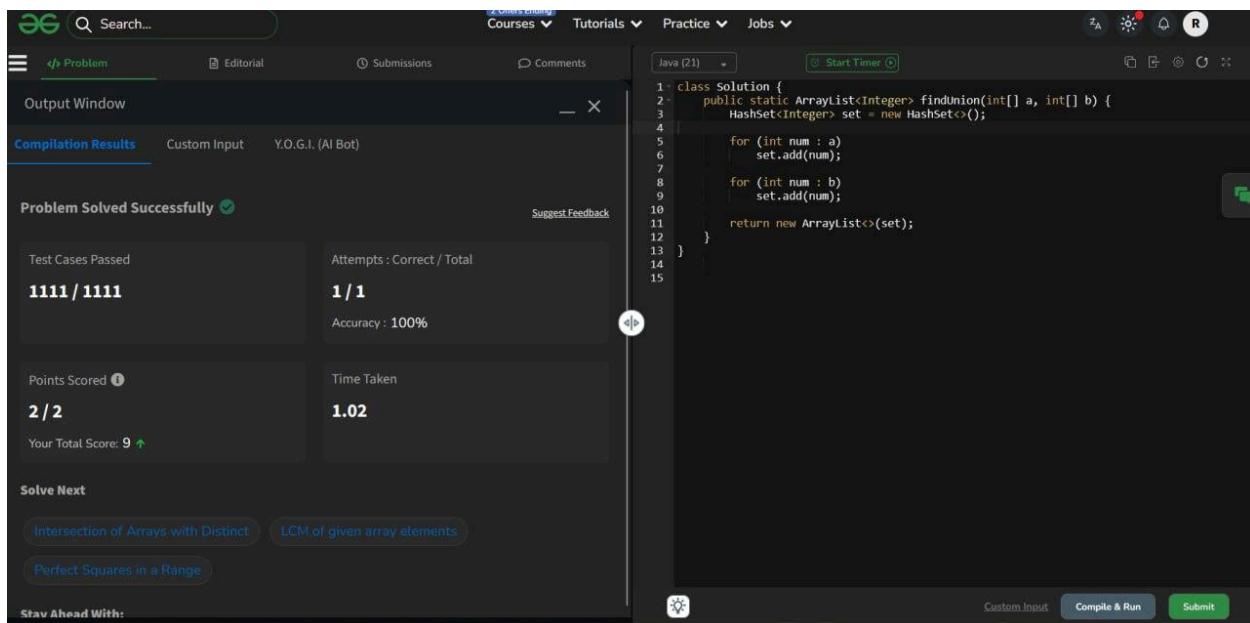
Note that, You can return the Union in any order but the driver code will print the result in sorted order only.

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.



```
class Solution {
    public static ArrayList<Integer> findUnion(int[] a, int[] b) {
        HashSet<Integer> set = new HashSet<>();
        for (int num : a)
            set.add(num);
        for (int num : b)
            set.add(num);
        return new ArrayList<>(set);
    }
}
```

Output Window

Compilation Results

Problem Solved Successfully

Test Cases Passed: 1111 / 1111

Attempts : Correct / Total: 1 / 1

Accuracy : 100%

Points Scored: 2 / 2

Your Total Score: 9

Solve Next

Intersection of Arrays with Distinct

LCM of given array elements

Perfect Squares in a Range

Custom Input

Compile & Run

Submit

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.

Approach

Assume the first element is the largest

Traverse the array

Update the maximum whenever a bigger value is found

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

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

The status bar at the bottom indicates "Custom Input" and "Compile & Run" buttons.

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.

Approach

Save the last element

Shift all elements one step to the right

Put the saved element at index 0

The screenshot shows a LeetCode problem page for "Rotate Array". The code editor contains the following Java solution:

```
1 // User function Template for Java
2
3= class Solution {
4=     public void rotate(int[] arr) {
5=
6         int n = arr.length;
7         int last = arr[n - 1];
8
9         for (int i = n - 1; i > 0; i--) {
10             arr[i] = arr[i - 1];
11         }
12
13         arr[0] = last;
14     }
15 }
```

The "Compilation Results" section shows "Problem Solved Successfully" with 1115 / 1115 test cases passed. The "Attempts: Correct / Total" is 1 / 1, and the "Accuracy: 100%". The "Points Scored" is 1 / 1, and the "Time Taken" is 1.42 seconds. The "Custom Input" button is visible at the bottom right of the editor area.

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.

Approach (Kadane's Algorithm)

Keep two variables:

currentSum → max sum ending at current index

maxSum → overall maximum found so far

At each step:

Either start new subarray from current element

Or extend previous subarray

The screenshot shows a LeetCode problem page for "Maximum Subarray" (Problem 53). The code editor contains the following Java solution:

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

The "Compilation Results" section shows "Problem Solved Successfully" with a green checkmark. It displays "Test Cases Passed: 1120 / 1120", "Attempts: Correct / Total: 1 / 1", and "Accuracy: 100%". Below this, "Points Scored" is 4 / 4 and "Time Taken" is 0.58 seconds. At the bottom, there are links to "Solve Next" problems: "Count of Subarrays", "Longest Arithmetic Subsequence", and "Smallest sum contiguous subarray".

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: $\text{nums} = [1, 3, 5, 6]$, target = 5

Output: 2

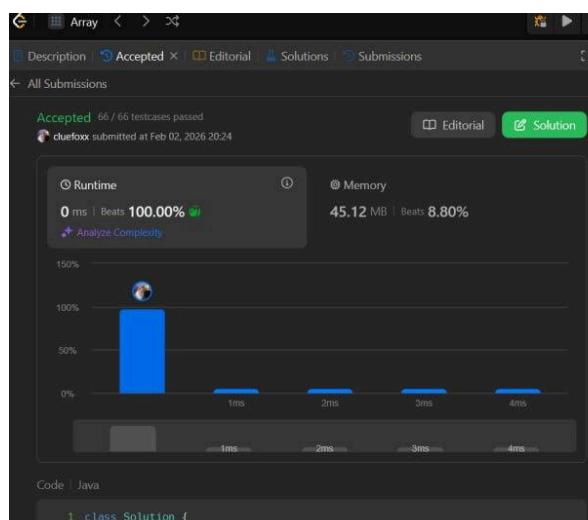
Approach (Binary Search)

Since the array is sorted and distinct

Use binary search:

If target is found → return its index

If not found → return the position where it should be inserted



The screenshot shows a LeetCode submission page for a Java solution. The submission was accepted with 66/66 testcases passed by duefoxx at Feb 02, 2026 20:24. The runtime is 0 ms (Beats 100.00%) and the memory usage is 45.12 MB (Beats 8.80%). A chart shows memory usage over time, with a large spike at the start and smaller spikes during the search loop. The code editor contains the following Java code:

```
1 class Solution {
2     public int searchInsert(int[] nums, int target) {
3         int left = 0;
4         int right = nums.length - 1;
5
6         while (left <= right) {
7             int mid = left + (right - left) / 2;
8
9             if (nums[mid] == target)
10                 return mid;
11             else if (nums[mid] < target)
12                 left = mid + 1;
13             else
14                 right = mid - 1;
15         }
16
17         // left is the correct insertion position
18         return left;
19     }
20 }
```

The code is saved and the cursor is at line 19, column 9.

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 shows a programming environment with a Java code editor and a results summary. The code implements a solution for the Jump Game II problem. The results indicate the user solved the problem successfully with 100% accuracy in 0.58 seconds.

```
1 class Solution {
2     public int minJumps(int[] arr) {
3         int n = arr.length;
4
5         // If array has 1 element, no jump needed
6         if (n == 1)
7             return 0;
8
9         // If first element is 0, we cannot move
10        if (arr[0] == 0)
11            return -1;
12
13        int maxReach = arr[0];
14        int steps = arr[0];
15        int jumps = 1;
16
17        for (int i = 1; i < n; i++) {
18
19            // If we reached the end
20            if (i == n - 1)
21                return jumps;
22
23            // Update the maximum reachable index
24            maxReach = Math.max(maxReach, i + arr[i]);
25
26            // Use a step
27            steps--;
28
29            // If no steps remain
30            if (steps == 0) {
31                jumps++;
32
33                // If current index is beyond maxReach, cannot proceed
34                if (i >= maxReach)
35                    return -1;
36
37            }
38        }
39
40        // Re-initialize steps
41    }
42}
```

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]`.

Approach (HashMap)

Traverse the array once

For each number:

Compute complement = `target - nums[i]`

If complement already exists in the map → solution found

Otherwise, store the current number with its index

The screenshot shows a LeetCode problem page for "Two Sum". The code editor contains a Java solution using a HashMap to store complements. The code is accepted with a runtime of 0 ms. The test cases passed are Case 1, Case 2, and Case 3. The input provided was `nums = [3,3]` and `target = 6`. The output returned was `[0,1]`.

```
1 class Solution {
2     public int[] twoSum(int[] nums, int target) {
3         HashMap<Integer, Integer> map = new HashMap<>();
4
5         for (int i = 0; i < nums.length; i++) {
6             int complement = target - nums[i];
7
8             if (map.containsKey(complement)) {
9                 return new int[]{map.get(complement), i};
10            }
11        }
12    }
13 }
```