

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 displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Compilation Results' for a problem named 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. Performance metrics include 'Test Cases Passed: 1115 / 1115', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 2 / 2', and 'Time Taken: 0.91'. Below this, 'Solve Next' buttons are visible for 'Mountain Subarray Problem', 'Java ArrayList Operation', and 'Even and odd elements at even and odd positions'. The main editor on the right shows a Java solution for reversing an array. The code uses a two-pointer approach, initializing 'left' at 0 and 'right' at 'arr.length - 1', and swapping elements until they meet. The 'main' method initializes the array [1, 4, 3, 2, 6, 5] and prints its elements. The bottom of the interface includes a 'Submit' button and a 'Custom Input' field.

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

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 displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Problem Solved Successfully' with a green checkmark. Below this, statistics are listed: 'Test Cases Passed: 1111 / 1111', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 1 / 1', and 'Time Taken: 0.34'. At the bottom left, there are buttons for 'Solve Next' and 'Stay Ahead With:'. The main area on the right shows a Java code editor with the following code:

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

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

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 `k`th 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 displays a coding platform interface 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)'. A green checkmark indicates 'Problem Solved Successfully'. Below this, statistics are shown: 'Test Cases Passed: 1121 / 1121', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Time Taken: 0.64'. A 'Your Total Score: 7' is also visible. Under 'Solve Next', there are buttons for 'Smallest Positive Missing', 'Valid Pair Sum', and 'Optimal Array'. At the bottom, a 'Stay Ahead With:' section is partially visible. On the right, the code editor shows a Java solution for the 'kthSmallest' problem. The code is as follows:

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

At the bottom right of the code editor, 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.

The screenshot displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. It confirms 'Problem Solved Successfully' with 11/11 test cases passed, 1/1 attempts, 100% accuracy, 2/2 points scored, and a time taken of 1.02. Below this, 'Solve Next' buttons are visible for 'Intersection of Arrays with Distinct', 'LCM of given array elements', and 'Perfect Squares in a Range'. The right side shows a Java code editor with the following code:

```
1 class Solution {
2     public static ArrayList<Integer> findUnion(int[] a, int[] b) {
3         HashSet<Integer> set = new HashSet<>();
4
5         for (int num : a)
6             set.add(num);
7
8         for (int num : b)
9             set.add(num);
10
11         return new ArrayList<>(set);
12     }
13 }
14
15
```

At the bottom right, there are buttons for 'Custom Input', 'Compile & Run', and '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 displays a coding platform interface with a dark theme. On the left, the 'Problem' tab is active, showing 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. A green checkmark indicates 'Problem Solved Successfully'. Below this, statistics are shown: 'Test Cases Passed: 1115 / 1115', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 1 / 1', and 'Time Taken: 0.85'. A 'Your Total Score: 10' is also visible. Under 'Solve Next', there are buttons for 'Last index of One', 'Pairs with Positive Negative values', and 'Repeated IDs'. At the bottom, a 'Stay Ahead With:' section has a search bar. On the right, the 'Editor' tab is active, showing a Java solution for the 'largest' problem. The code is as follows:

```
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        return max;
11    }
12 }
```

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

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]

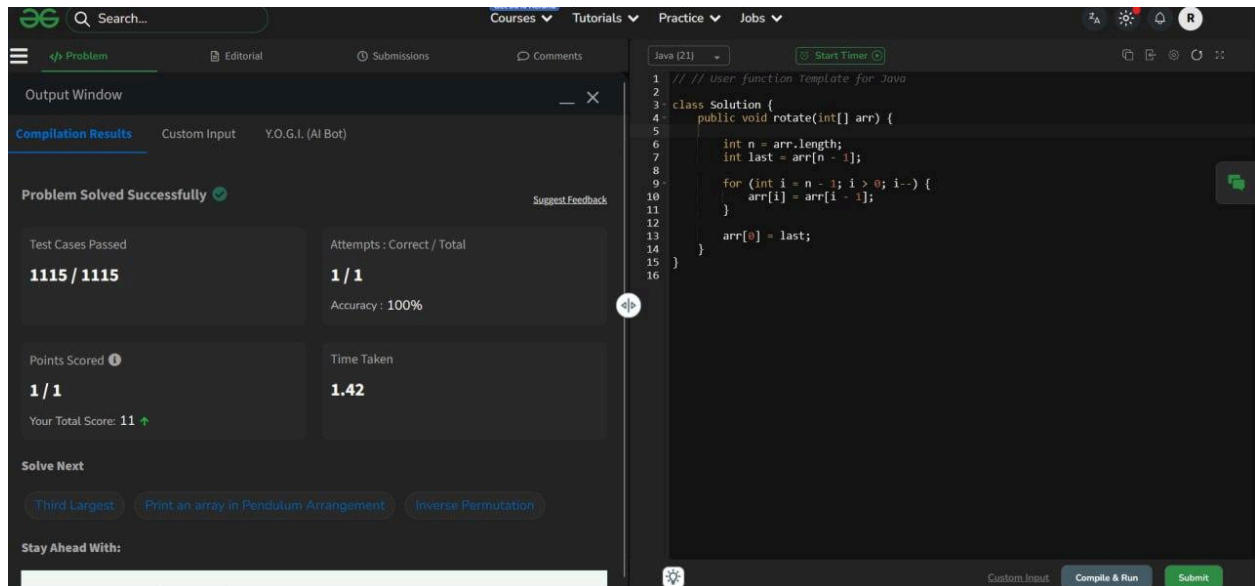
Explanation: If we rotate arr by one position in clockwise 5 come to the front and remaining those are shifted to the end.

Approach

Save the last element

Shift all elements one step to the right

Put the saved element at index 0



```
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 }
16
```

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:

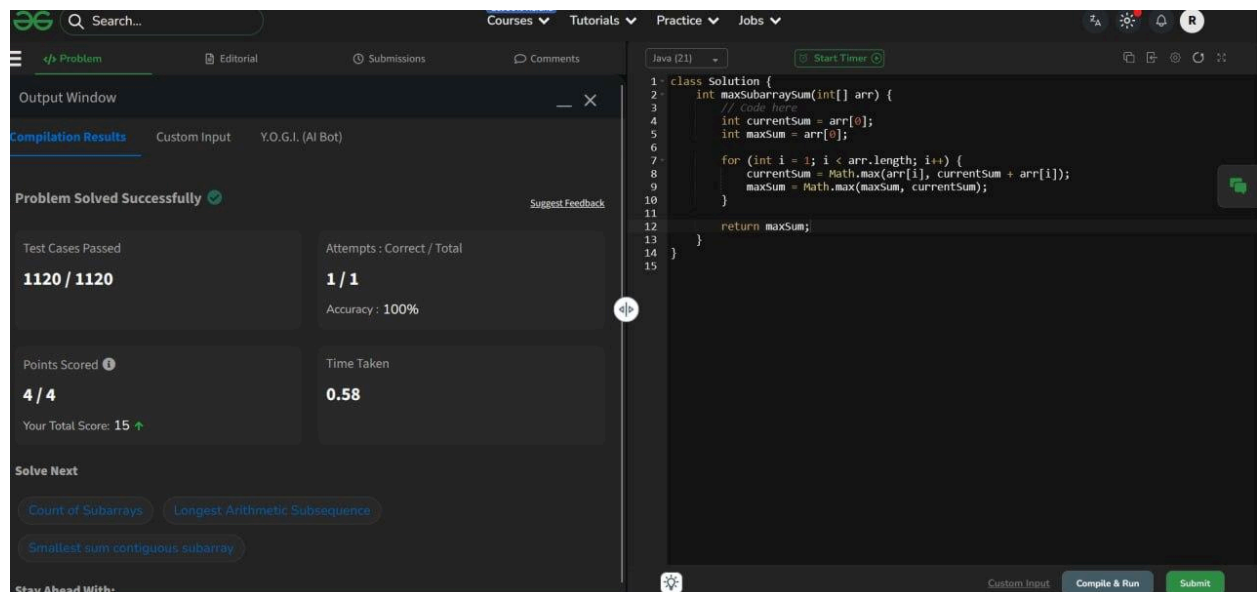
`currentSum` → max sum ending at current index

`maxSum` → overall maximum found so far

At each step:

Either start new subarray from current element

Or extend previous subarray



The screenshot displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. Below this, statistics are shown: 'Test Cases Passed: 1120 / 1120', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Time Taken: 0.58'. A 'Your Total Score: 15' is also visible. Under 'Solve Next', there are buttons for 'Count of Subarrays', 'Longest Arithmetic Subsequence', and 'Smallest sum contiguous subarray'. The main editor on the right shows a Java solution for 'maxSubarraySum' in a class named 'Solution'. The code implements Kadane's Algorithm, initializing 'currentSum' and 'maxSum' to the first element of the array, then iterating through the rest of the array to update these values based on whether adding the current element increases the current sum or if starting a new subarray is more beneficial. The 'Submit' button is at the bottom right.

```
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
12        return maxSum;
13    }
14 }
15 }
```

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

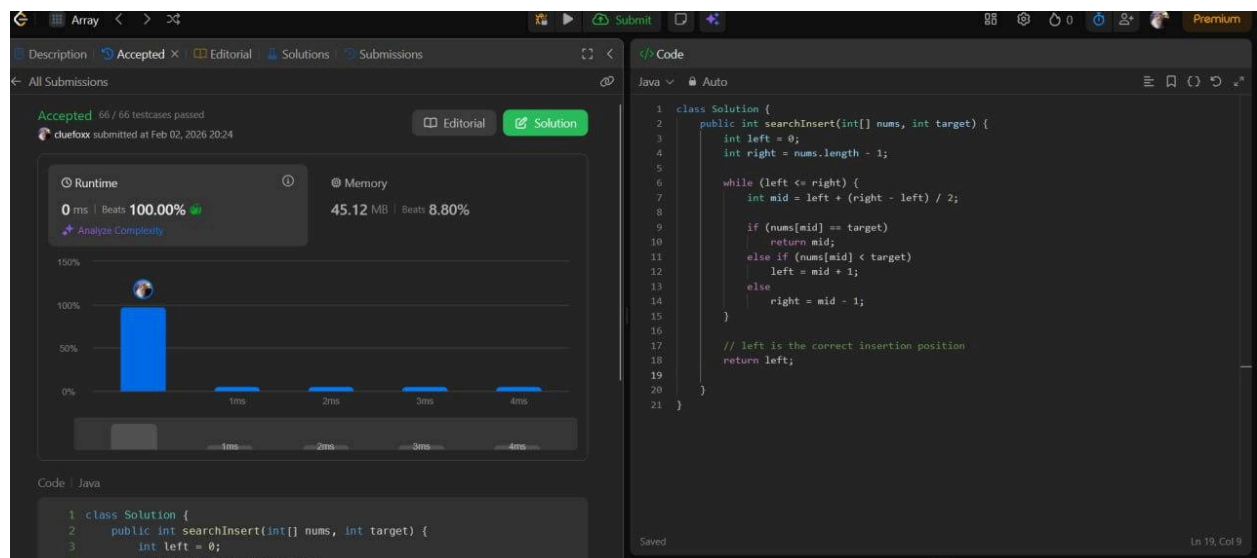
Approach (Binary Search)

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



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
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                // Re-initialize steps
```

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]`.

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 displays a LeetCode submission page for the "Two Sum" problem. The left sidebar shows the problem description and a list of related challenges. The main area is divided into two panels: "Code" and "Test Result".

Code Panel: Shows the Java solution for the "Two Sum" problem. The code uses a HashMap to store the complement of each number in the array. The solution is as follows:

```
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            map.put(nums[i], i);
12        }
13    }
14 }
```

Test Result Panel: Shows the test results for the submission. The status is "Accepted" with a runtime of 0 ms. The input is `nums = [3,3]` and `target = 6`. The output is `[0,1]`.