JAVA PRACTICE SHEET(26/01/26)

1.

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.

Input: arr[] = [7, 10, 4, 3, 20, 15], k = 3

Output: 7

Explanation: 3rd smallest element in the given array is 7.

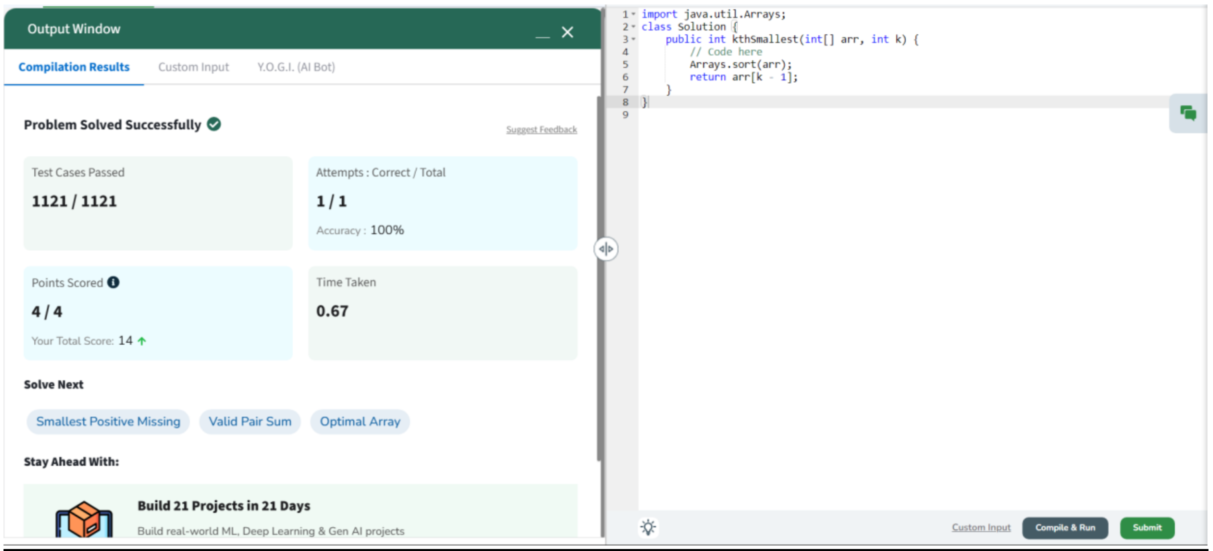
Constraints:

1 ≤ arr.size() ≤ 105

1 ≤ arr[i] ≤ 105

1 ≤ k ≤ arr.size()

Sol



Output Window

Compilation Results | Custom Input | Y.O.G.I. (AI Bot)

Problem Solved Successfully

Test Cases Passed
1121 / 1121

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
4 / 4
Your Total Score: 14

Time Taken
0.67

Solve Next: Smallest Positive Missing | Valid Pair Sum | Optimal Array

Stay Ahead With: Build 21 Projects in 21 Days — Build real-world ML, Deep Learning & Gen AI projects

```java
import java.util.Arrays;
class Solution {
    public int kthSmallest(int[] arr, int k) {
        // Code here
        Arrays.sort(arr);
        return arr[k - 1];
    }
}
```

2.

Given an array arr[] denoting heights of n towers and a positive integer k.

For each tower, you must perform exactly one of the following operations exactly once.

Increase the height of the tower by k

Decrease the height of the tower by k

Find out the minimum possible difference between the height of the shortest and tallest towers after you have modified each tower.

You can find a slight modification of the problem here.

Note: It is compulsory to increase or decrease the height by k for each tower. After the operation, the resultant array should not contain any negative integers.

Examples :

Input: k = 2, arr[] = [1, 5, 8, 10]

Output: 5

Explanation: The array can be modified as [1+k, 5-k, 8-k, 10-k] = [3, 3, 6, 8]. The difference between the largest and the smallest is 8-3 = 5.

Input: k = 3, arr[] = [3, 9, 12, 16, 20]

Output: 11

Explanation: The array can be modified as [3+k, 9+k, 12-k, 16-k, 20-k] = [6, 12, 9, 13, 17]. The difference between the largest and the smallest is 17-6 = 11.

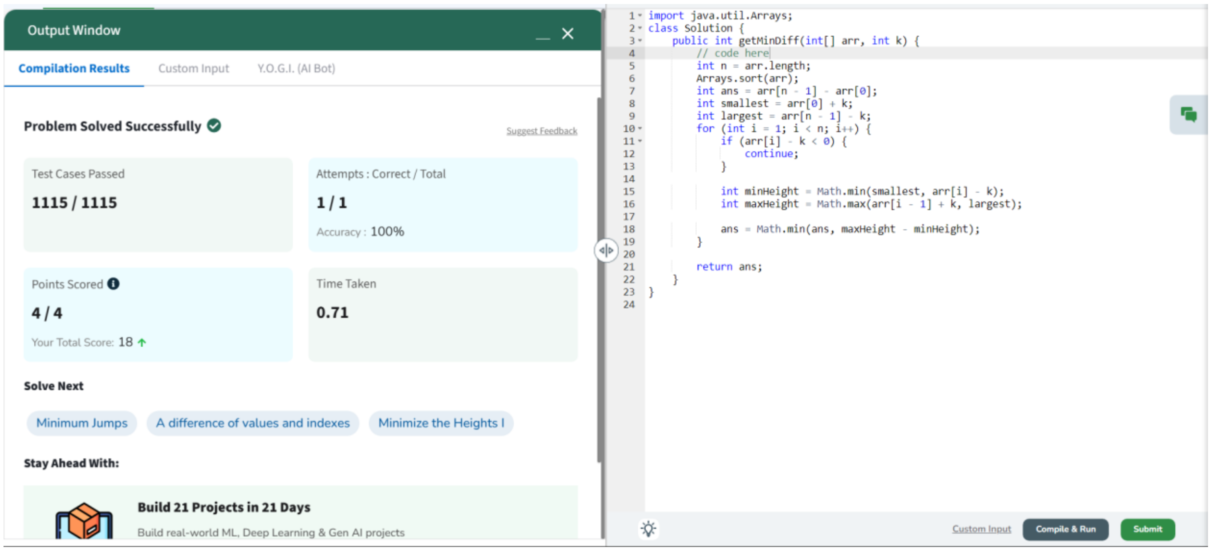
Constraints

1 ≤ k ≤ 107

1 ≤ n ≤ 105

1 ≤ arr[i] ≤ 107

Sol


Output Window

Compilation Results | Custom Input | Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed
1115 / 1115

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
4 / 4
Your Total Score: 18

Time Taken
0.71

Solve Next

Minimum Jumps | A difference of values and indexes | Minimize the Heights I

Stay Ahead With:

Build 21 Projects in 21 Days
Build real-world ML, Deep Learning & Gen AI projects

```java
import java.util.Arrays;
class Solution {
    public int getMinDiff(int[] arr, int k) {
        // code here
        int n = arr.length;
        Arrays.sort(arr);
        int ans = arr[n - 1] - arr[0];
        int smallest = arr[0] + k;
        int largest = arr[n - 1] - k;
        for (int i = 1; i < n; i++) {
            if (arr[i] - k < 0) {
                continue;
            }

            int minHeight = Math.min(smallest, arr[i] - k);
            int maxHeight = Math.max(arr[i - 1] + k, largest);

            ans = Math.min(ans, maxHeight - minHeight);
        }

        return ans;
    }
}
```

3.

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.

School of Computer Science and Engineering

Experiment List for Programming Ability and Logic Building - 1

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.

Input: arr = [1, 4, 3, 2, 6, 7]

Output: 2

Explanation: First we jump from the 1st to 2nd element and then jump to the last element.

Input: arr = [0, 10, 20]

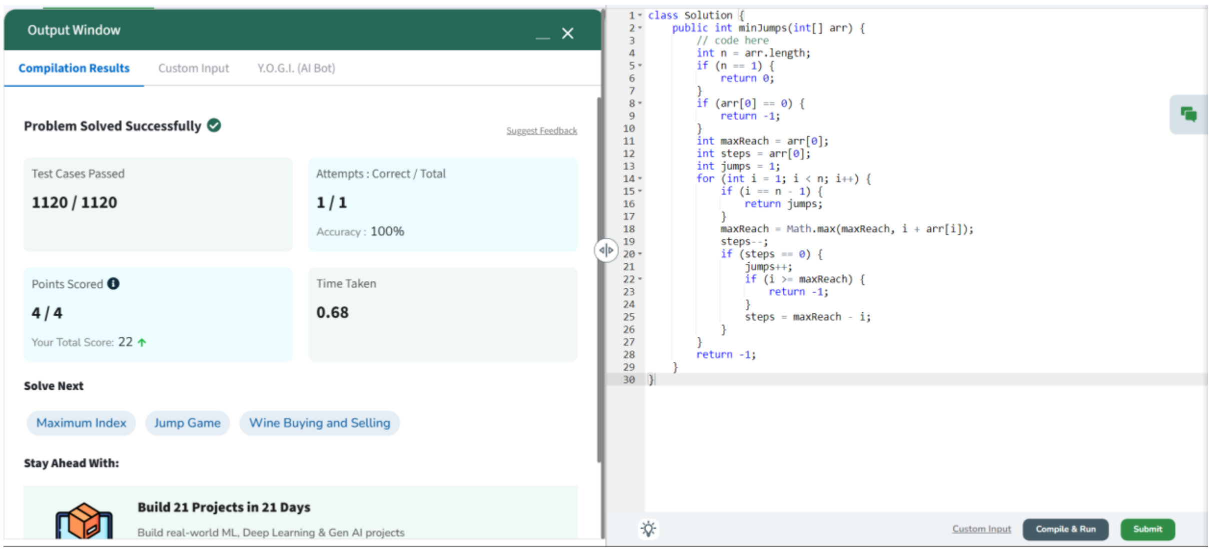
Output: -1

Explanation: We cannot go anywhere from the 1st element.

Constraints:

2 ≤ arr.size() ≤ 105

0 ≤ arr[i] ≤ 105



Problem Solved Successfully

Test Cases Passed
1120 / 1120

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
4 / 4
Your Total Score: 22

Time Taken
0.68

```java
class Solution {
    public int minJumps(int[] arr) {
        // code here
        int n = arr.length;
        if (n == 1) {
            return 0;
        }
        if (arr[0] == 0) {
            return -1;
        }
        int maxReach = arr[0];
        int steps = arr[0];
        int jumps = 1;
        for (int i = 1; i < n; i++) {
            if (i == n - 1) {
                return jumps;
            }
            maxReach = Math.max(maxReach, i + arr[i]);
            steps--;
            if (steps == 0) {
                jumps++;
                if (i >= maxReach) {
                    return -1;
                }
                steps = maxReach - i;
            }
        }
        return -1;
    }
}
```

4.

Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n]

inclusive.

There is only one repeated number in nums, return this repeated number.

You must solve the problem without modifying the array nums and using only constant extra

space.

Example 1:

Input: nums = [1,3,4,2,2]

Output: 2

Example 2:

Input: nums = [3,1,3,4,2]

Output: 3

Example 3:

Input: nums = [3,3,3,3,3]

Output: 3

Constraints:

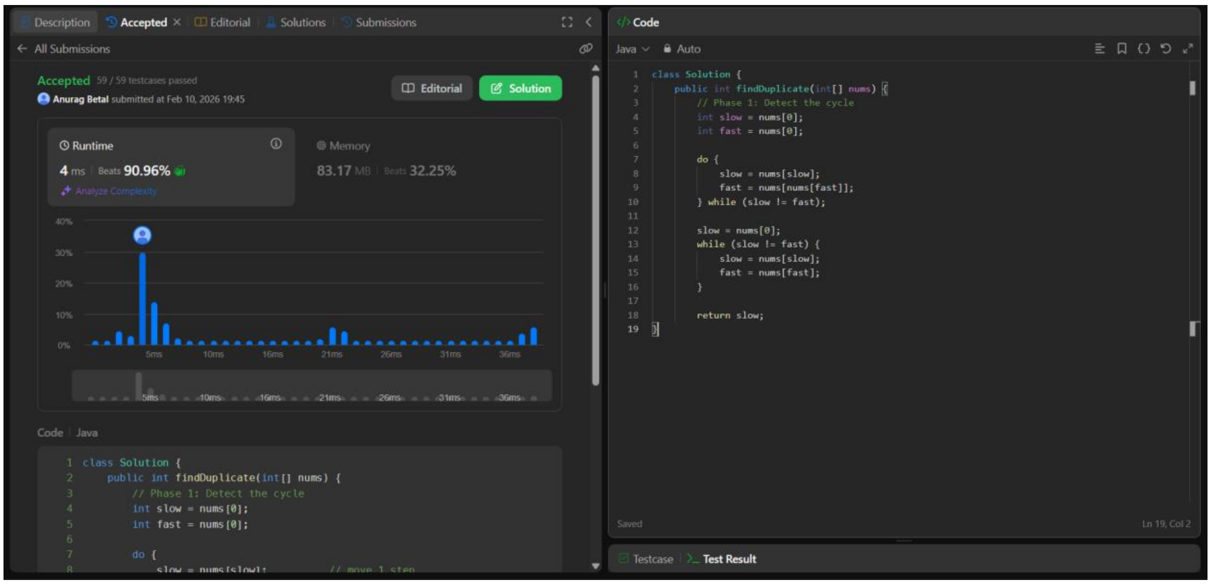
1 <= n <= 105

nums.length == n + 1

1 <= nums[i] <= n

All the integers in nums appear only once except for precisely one integer which appears two or more times.

<u>Sol</u>



Accepted 59 / 59 testcases passed

Anurag Betal submitted at Feb 10, 2026 19:45

Runtime 4 ms | Beats 90.96%

Memory 83.17 MB | Beats 32.25%

```java
class Solution {
    public int findDuplicate(int[] nums) {
        // Phase 1: Detect the cycle
        int slow = nums[0];
        int fast = nums[0];

        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);

        slow = nums[0];
        while (slow != fast) {
            slow = nums[slow];
            fast = nums[fast];
        }

        return slow;
}
```

5.

Given two sorted arrays a[] and b[] of size n and m respectively, the task is to merge them in sorted order without using any extra space. Modify a[] so that it contains the first n elements and modify b[] so that it contains the last m elements.

Examples:

Input: a[] = [2, 4, 7, 10], b[] = [2, 3]

Output: a[] = [2, 2, 3, 4], b[] = [7, 10]

Explanation: After merging the two non-decreasing arrays, we get, [2, 2, 3, 4, 7, 10]

Input: a[] = [1, 5, 9, 10, 15, 20], b[] = [2, 3, 8, 13]

Output: a[] = [1, 2, 3, 5, 8, 9], b[] = [10, 13, 15, 20]

Explanation: After merging two sorted arrays we get [1, 2, 3, 5, 8, 9, 10, 13, 15, 20].

Input: a[] = [0, 1], b[] = [2, 3]

Output: a[] = [0, 1], b[] = [2, 3]

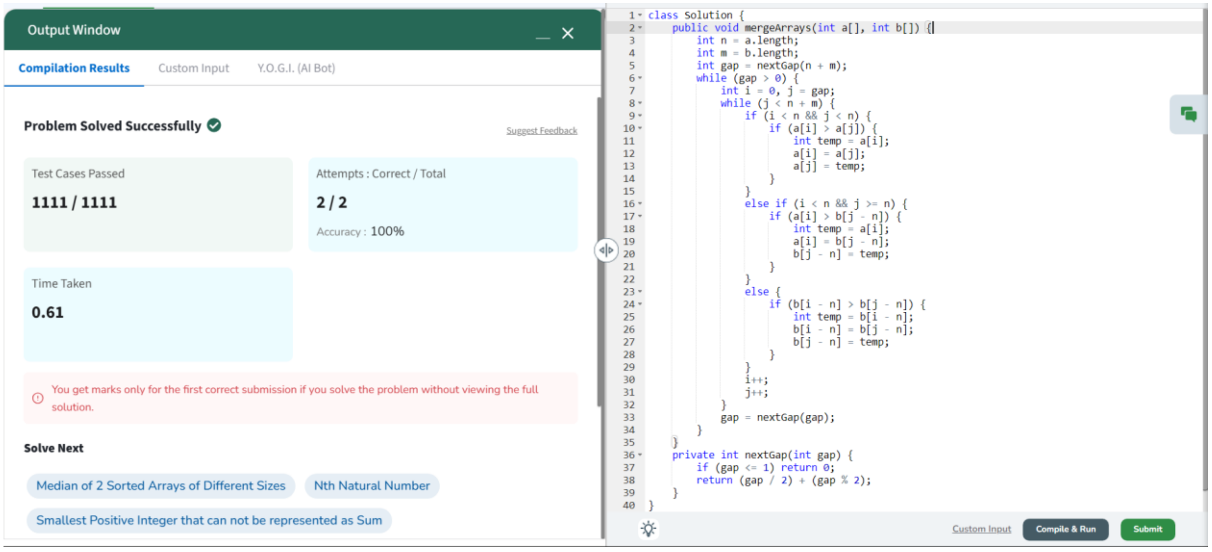
Explanation: After merging two sorted arrays we get [0, 1, 2, 3].

Constraints:

1 ≤ n, m ≤ 105

0 ≤ a[i], b[i] ≤ 10^7

<u>Sol</u>



Output Window

Compilation Results | Custom Input | Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed
1111 / 1111

Attempts : Correct / Total
2 / 2
Accuracy : 100%

Time Taken
0.61

You get marks only for the first correct submission if you solve the problem without viewing the full solution.

Solve Next

Median of 2 Sorted Arrays of Different Sizes | Nth Natural Number | Smallest Positive Integer that can not be represented as Sum

```java
class Solution {
    public void mergeArrays(int a[], int b[]) {
        int n = a.length;
        int m = b.length;
        int gap = nextGap(n + m);
        while (gap > 0) {
            int i = 0, j = gap;
            while (j < n + m) {
                if (i < n && j < n) {
                    if (a[i] > a[j]) {
                        int temp = a[i];
                        a[i] = a[j];
                        a[j] = temp;
                    }
                }
                else if (i < n && j >= n) {
                    if (a[i] > b[j - n]) {
                        int temp = a[i];
                        a[i] = b[j - n];
                        b[j - n] = temp;
                    }
                }
                else {
                    if (b[i - n] > b[j - n]) {
                        int temp = b[i - n];
                        b[i - n] = b[j - n];
                        b[j - n] = temp;
                    }
                }
                i++;
                j++;
            }
            gap = nextGap(gap);
        }
    }
    private int nextGap(int gap) {
        if (gap <= 1) return 0;
        return (gap / 2) + (gap % 2);
    }
}
```

6.

Given an array of intervals where intervals[i] = [starti, endi], merge all

overlapping intervals, and return an array of the non-overlapping intervals that

cover all the intervals in the input.

Example 1:

Input: intervals = [[1,3],[2,6],[8,10],[15,18]]

Output: [[1,6],[8,10],[15,18]]

Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].

Example 2:

Input: intervals = [[1,4],[4,5]]

Output: [[1,5]]

Explanation: Intervals [1,4] and [4,5] are considered overlapping.

Example 3:

Input: intervals = [[4,7],[1,4]]

Output: [[1,7]]

Explanation: Intervals [1,4] and [4,7] are considered overlapping.

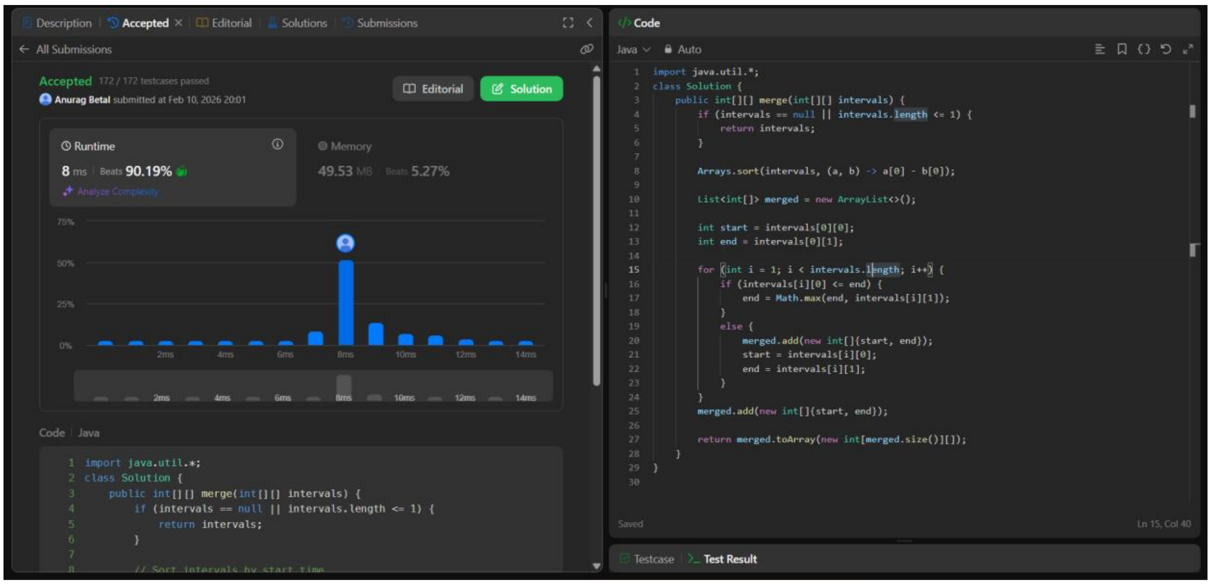
Constraints:

1 <= intervals.length <= 10^4

intervals[i].length == 2

0 <= starti <= endi <= 10^4

<u>Sol</u>



Accepted 172 / 172 testcases passed
Anurag Betal submitted at Feb 10, 2026 20:01

8 ms | Beats 90.19%

49.53 MB | Beats 5.27%

```java
import java.util.*;
class Solution {
    public int[][] merge(int[][] intervals) {
        if (intervals == null || intervals.length <= 1) {
            return intervals;
        }

        Arrays.sort(intervals, (a, b) -> a[0] - b[0]);

        List<int[]> merged = new ArrayList<>();

        int start = intervals[0][0];
        int end = intervals[0][1];

        for (int i = 1; i < intervals.length; i++) {
            if (intervals[i][0] <= end) {
                end = Math.max(end, intervals[i][1]);
            }
            else {
                merged.add(new int[]{start, end});
                start = intervals[i][0];
                end = intervals[i][1];
            }
        }
        merged.add(new int[]{start, end});

        return merged.toArray(new int[merged.size()][]);
    }
}
```

7.

Given three sorted arrays in non-decreasing order, print all common elements

in non-decreasing order across these arrays. If there are no such elements return

an empty array. In this case, the output will be -1.

Note: can you handle the duplicates without using any additional Data Structure?

Examples :

Input: arr1 = [1, 5, 10, 20, 40, 80] , arr2 = [6, 7, 20, 80, 100] , arr3 = [3, 4, 15, 20, 30, 70, 80, 120]

Output: [20, 80]

Explanation: 20 and 80 are the only common elements in arr1, arr2 and arr3.

Input: arr1 = [1, 2, 3, 4, 5] , arr2 = [6, 7] , arr3 = [8,9,10]

Output: [-1]

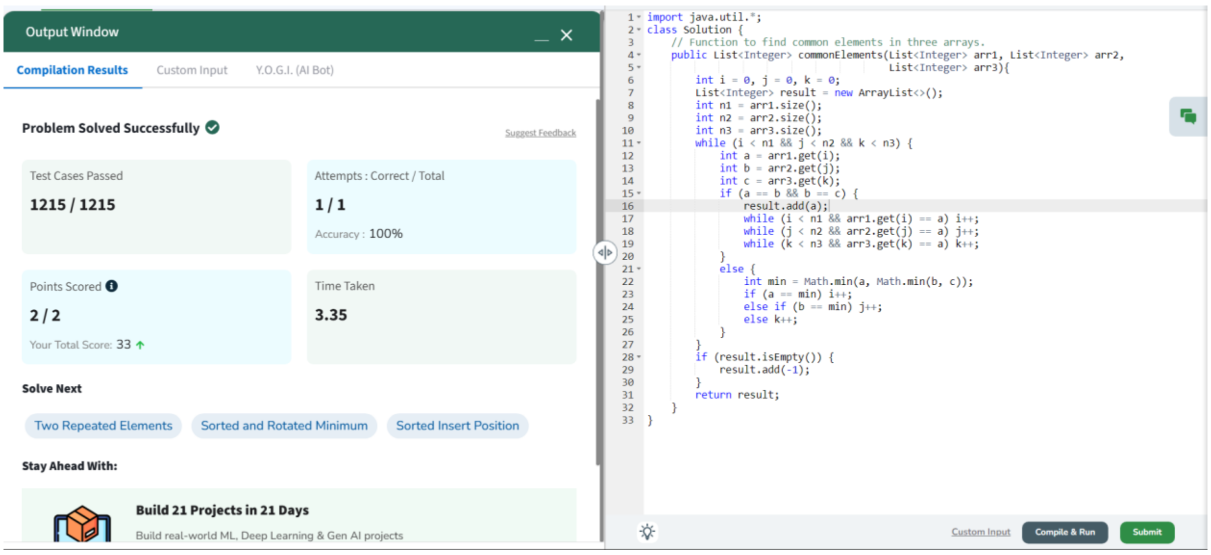
Explanation: There are no common elements in arr1, arr2 and arr3.

Input: arr1 = [1, 1, 1, 2, 2, 2], arr2 = [1, 1, 2, 2, 2], arr3 = [1, 1, 1, 1, 2, 2, 2, 2]

Output: [1, 2]

Explanation: We do not need to consider duplicates

Sol



Problem Solved Successfully

Test Cases Passed
1215 / 1215

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
2 / 2
Your Total Score: 33

Time Taken
3.35

```java
import java.util.*;
class Solution {
    // Function to find common elements in three arrays.
    public List<Integer> commonElements(List<Integer> arr1, List<Integer> arr2,
                                        List<Integer> arr3){
        int i = 0, j = 0, k = 0;
        List<Integer> result = new ArrayList<>();
        int n1 = arr1.size();
        int n2 = arr2.size();
        int n3 = arr3.size();
        while (i < n1 && j < n2 && k < n3) {
            int a = arr1.get(i);
            int b = arr2.get(j);
            int c = arr3.get(k);
            if (a == b && b == c) {
                result.add(a);
                while (i < n1 && arr1.get(i) == a) i++;
                while (j < n2 && arr2.get(j) == a) j++;
                while (k < n3 && arr3.get(k) == a) k++;
            }
            else {
                int min = Math.min(a, Math.min(b, c));
                if (a == min) i++;
                else if (b == min) j++;
                else k++;
            }
        }
        if (result.isEmpty()) {
            result.add(-1);
        }
        return result;
    }
}
```

8.

Given an integer n, find its factorial. Return a list of integers denoting the digits

that make up the factorial of n.

Examples:

Input: n = 5

Output: [1, 2, 0]

Explanation: 5! = 1*2*3*4*5 = 120

Input: n = 10

Output: [3, 6, 2, 8, 8, 0, 0]

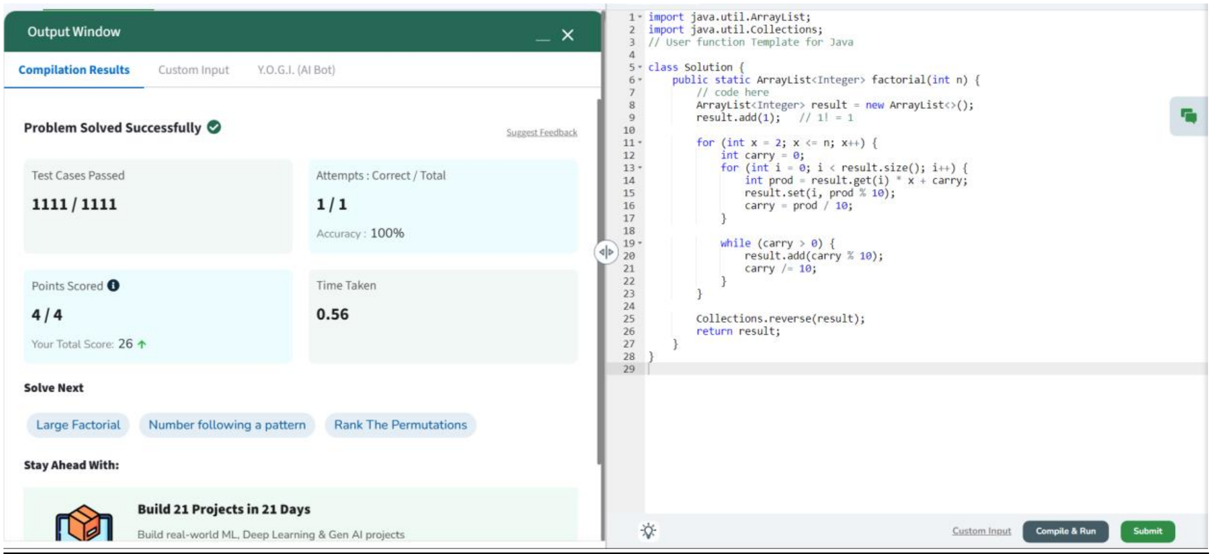
Explanation: 10! = 1*2*3*4*5*6*7*8*9*10 = 3628800

Input: n = 1

Output: [1]

Explanation: 1! = 1

Sol



Output Window

Compilation Results | Custom Input | Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed
1111 / 1111

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
4 / 4
Your Total Score: 26

Time Taken
0.56

Solve Next

Large Factorial | Number following a pattern | Rank The Permutations

Stay Ahead With:

Build 21 Projects in 21 Days
Build real-world ML, Deep Learning & Gen AI projects

```java
import java.util.ArrayList;
import java.util.Collections;
// User function Template for Java

class Solution {
    public static ArrayList<Integer> factorial(int n) {
        // code here
        ArrayList<Integer> result = new ArrayList<>();
        result.add(1);   // 1! = 1

        for (int x = 2; x <= n; x++) {
            int carry = 0;
            for (int i = 0; i < result.size(); i++) {
                int prod = result.get(i) * x + carry;
                result.set(i, prod % 10);
                carry = prod / 10;
            }

            while (carry > 0) {
                result.add(carry % 10);
                carry /= 10;
            }
        }

        Collections.reverse(result);
        return result;
    }
}
```

9.

Given two arrays a[] and b[], your task is to determine whether b[] is a subset of a[].

Examples:

Input: a[] = [11, 7, 1, 13, 21, 3, 7, 3], b[] = [11, 3, 7, 1, 7]

Output: true

Explanation: b[] is a subset of a[]

Input: a[] = [1, 2, 3, 4, 4, 5, 6], b[] = [1, 2, 4]

Output: true

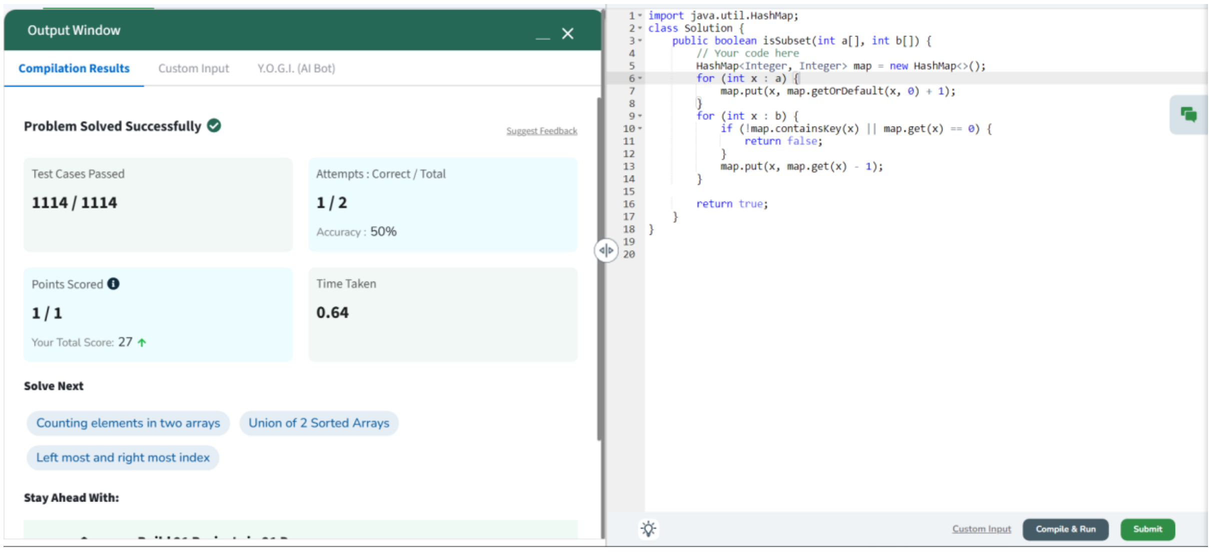
Explanation: b[] is a subset of a[]

Input: a[] = [10, 5, 2, 23, 19], b[] = [19, 5, 3]

Output: false

Explanation: b[] is not a subset of a[]

Sol



Problem Solved Successfully

Test Cases Passed
1114 / 1114

Attempts : Correct / Total
1 / 2
Accuracy : 50%

Points Scored
1 / 1
Your Total Score: 27

Time Taken
0.64

Solve Next

Counting elements in two arrays
Union of 2 Sorted Arrays
Left most and right most index

Stay Ahead With:

```java
import java.util.HashMap;
class Solution {
    public boolean isSubset(int a[], int b[]) {
        // Your code here
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int x : a) {
            map.put(x, map.getOrDefault(x, 0) + 1);
        }
        for (int x : b) {
            if (!map.containsKey(x) || map.get(x) == 0) {
                return false;
            }
            map.put(x, map.get(x) - 1);
        }

        return true;
    }
}
```

10.

Given an array arr[] and an integer target, determine if there exists a triplet in the array whose sum equals the given target.

Return true if such a triplet exists, otherwise, return false.

Examples:

Input: arr[] = [1, 4, 45, 6, 10, 8], target = 13

Output: true

Explanation: The triplet {1, 4, 8} sums up to 13.

Input: arr[] = [1, 2, 4, 3, 6, 7], target = 10

Output: true

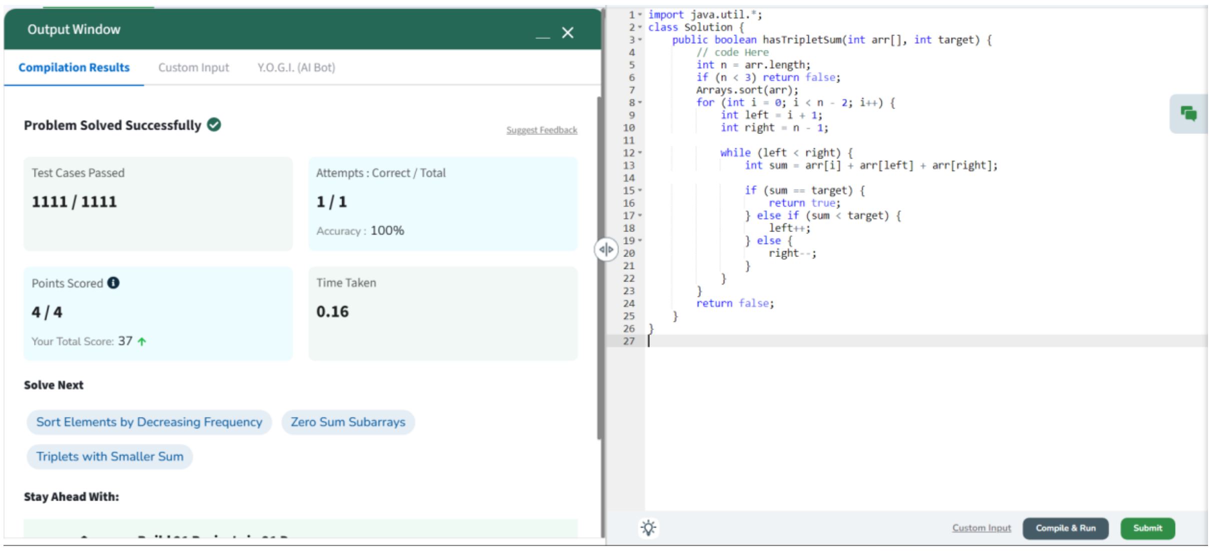
Explanation: The triplets {1, 3, 6} and {1, 2, 7} both sum to 10.

Input: arr[] = [40, 20, 10, 3, 6, 7], target = 24

Output: false

Explanation: No triplet in the array sums to 24.

Sol


Output Window

Compilation Results | Custom Input | Y.O.G.I. (AI Bot)

Problem Solved Successfully

Test Cases Passed
1111 / 1111

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
4 / 4
Your Total Score: 37

Time Taken
0.16

Solve Next

Sort Elements by Decreasing Frequency | Zero Sum Subarrays | Triplets with Smaller Sum

Stay Ahead With:

```java
import java.util.*;
class Solution {
    public boolean hasTripletSum(int arr[], int target) {
        // code Here
        int n = arr.length;
        if (n < 3) return false;
        Arrays.sort(arr);
        for (int i = 0; i < n - 2; i++) {
            int left = i + 1;
            int right = n - 1;

            while (left < right) {
                int sum = arr[i] + arr[left] + arr[right];

                if (sum == target) {
                    return true;
                } else if (sum < target) {
                    left++;
                } else {
                    right--;
                }
            }
        }
        return false;
    }
}
```

11.

Given an array arr[] with non-negative integers representing the height of blocks.

If the width of each block is 1, compute how much water can be trapped between

the blocks during the rainy season.

Examples:

Input: arr[] = [3, 0, 1, 0, 4, 0 2]

Output: 10

Explanation: Total water trapped = 0 + 3 + 2 + 3 + 0 + 2 + 0 = 10 units.

Input: arr[] = [3, 0, 2, 0, 4]

Output: 7

Explanation: Total water trapped = 0 + 3 + 1 + 3 + 0 = 7 units.

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

Output: 0

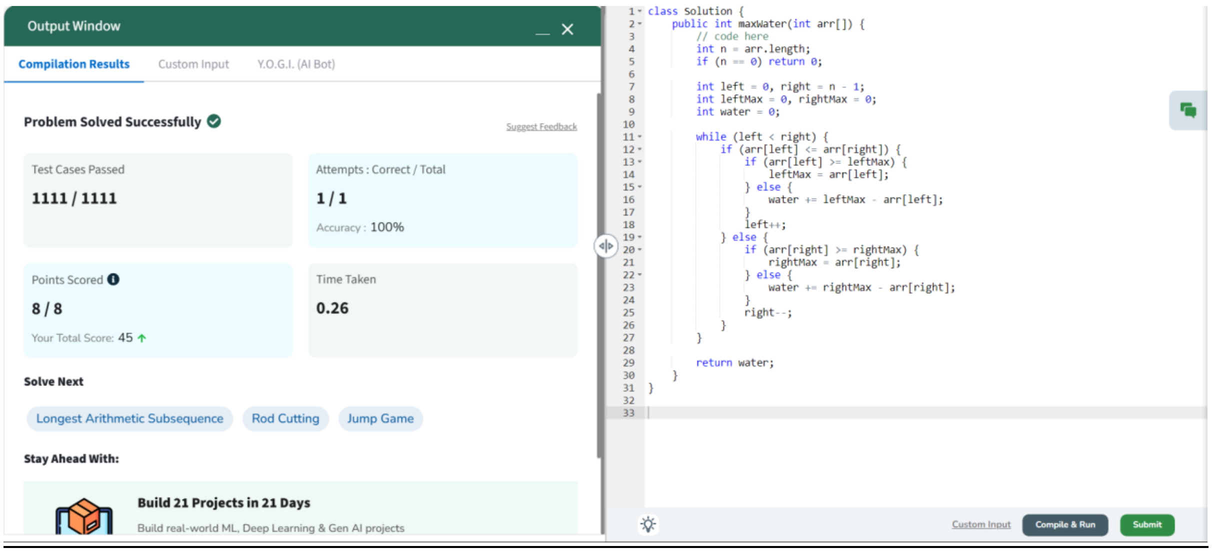
Explanation: We cannot trap water as there is no height bound on both sides.

Input: arr[] = [2, 1, 5, 3, 1, 0, 4]

Output: 9

Explanation: Total water trapped = 0 + 1 + 0 + 1 + 3 + 4 + 0 = 9 units.

Sol



Problem Solved Successfully

Test Cases Passed
1111 / 1111

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
8 / 8
Your Total Score: 45

Time Taken
0.26

Solve Next

Stay Ahead With:

Build 21 Projects in 21 Days
Build real-world ML, Deep Learning & Gen AI projects

```java
class Solution {
    public int maxWater(int arr[]) {
        // code here
        int n = arr.length;
        if (n == 0) return 0;

        int left = 0, right = n - 1;
        int leftMax = 0, rightMax = 0;
        int water = 0;

        while (left < right) {
            if (arr[left] <= arr[right]) {
                if (arr[left] >= leftMax) {
                    leftMax = arr[left];
                } else {
                    water += leftMax - arr[left];
                }
                left++;
            } else {
                if (arr[right] >= rightMax) {
                    rightMax = arr[right];
                } else {
                    water += rightMax - arr[right];
                }
                right--;
            }
        }

        return water;
    }
}

```