EXPERIMENT LIST FOR PROGRAMMING ABILITY AND LOGIC BUILDING – 1

(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

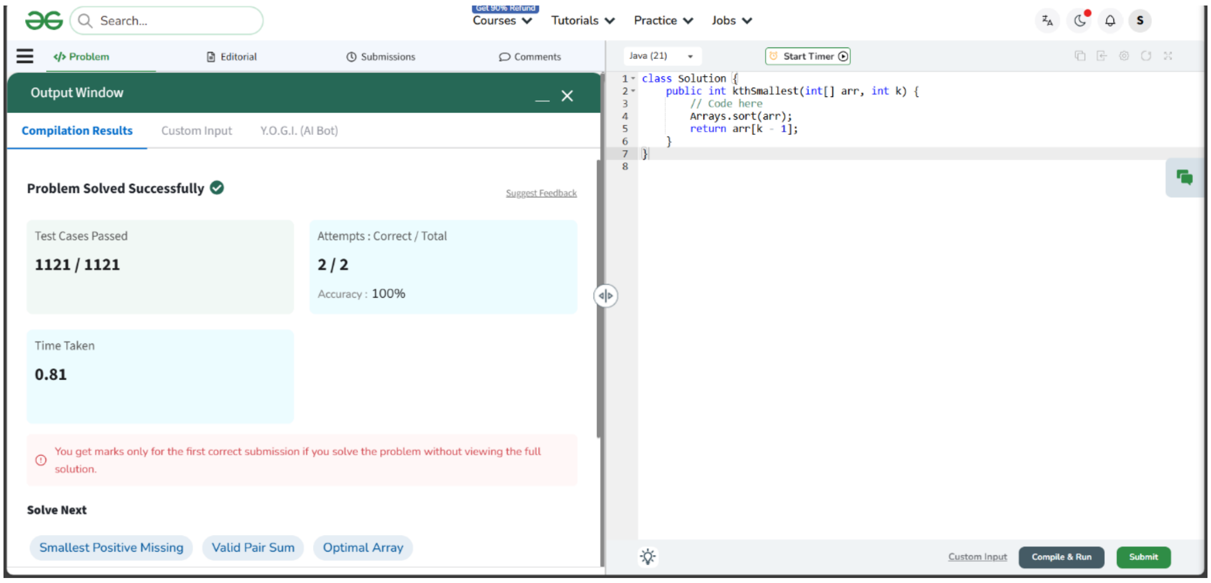
Explanation: 3rd smallest element in the given array is 7.

Constraints:

1 ≤ arr.size() ≤ 105

1 ≤ arr[i] ≤ 105

1 ≤ k ≤ arr.size()



Problem Solved Successfully

Test Cases Passed
1121 / 1121

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

Time Taken
0.81

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

Solve Next
Smallest Positive Missing
Valid Pair Sum
Optimal Array

```java
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

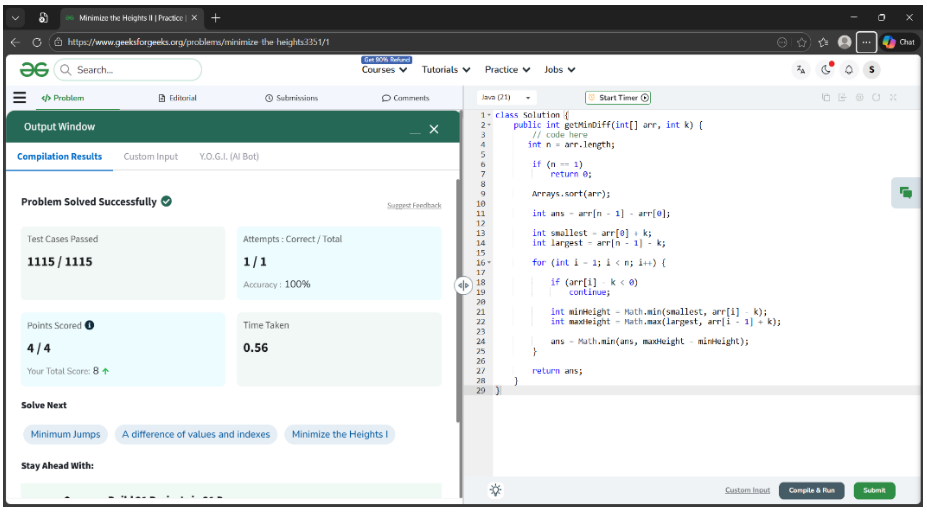
Explanation: The array can be modified as [3+k, 9+k, 12-k, 16-k, 20-k] = [6, 12, 9, 13, 17]. The difference between the largest and the smallest is 17-6 = 11.

Constraints

1 ≤ k ≤ 107

1 ≤ n ≤ 105

1 ≤ arr[i] ≤ 107



Problem Solved Successfully

Test Cases Passed
1115 / 1115

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

Points Scored
4 / 4
Your Total Score: 8

Time Taken
0.56

Solve Next

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

Stay Ahead With:

```java
class Solution {
    public int getMinDiff(int[] arr, int k) {
        // code here
       int n = arr.length;

        if (n == 1)
            return 0;

        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(largest, arr[i - 1] + k);

            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.
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]

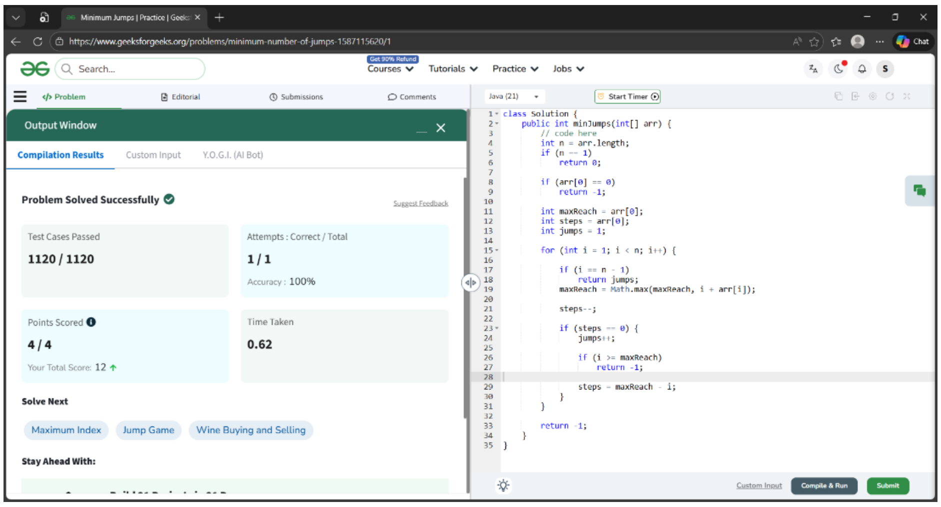
Output: -1

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: 12

Time Taken
0.62

Solve Next

Maximum Index · Jump Game · Wine Buying and Selling

Stay Ahead With:

```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 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

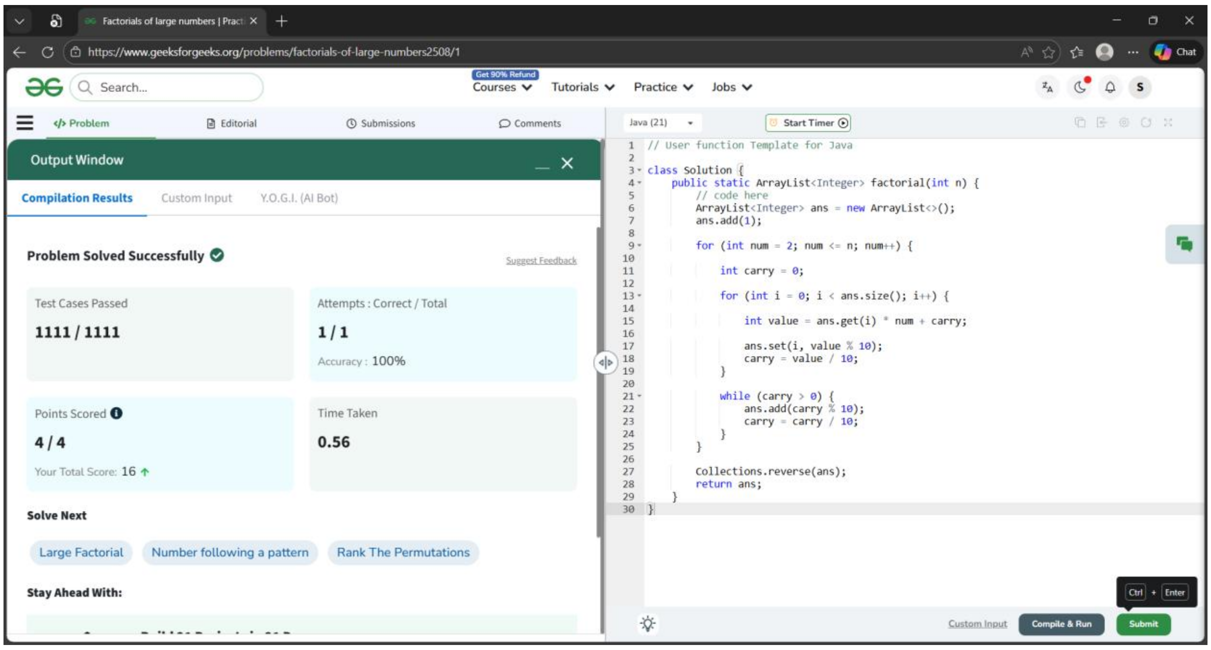
Output: [3, 6, 2, 8, 8, 0, 0]

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

Input: n = 1

Output: [1]

Explanation: 1! = 1



Problem Solved Successfully ✓

Test Cases Passed
1111 / 1111

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

Points Scored
4 / 4
Your Total Score: 16

Time Taken
0.56

Solve Next
Large Factorial
Number following a pattern
Rank The Permutations

Stay Ahead With:

```java
// User function Template for Java

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

        for (int num = 2; num <= n; num++) {

            int carry = 0;

            for (int i = 0; i < ans.size(); i++) {

                int value = ans.get(i) * num + carry;

                ans.set(i, value % 10);
                carry = value / 10;
            }

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

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