Experiment List for Programming Ability and Logic Building-2

Name : Ikshu Goel

Batch : 2CSE5

Roll Number : 2410030774

Week : 26/01/26 to 31/01/26

Lecture : 1 & 2

EXPERIMENT 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

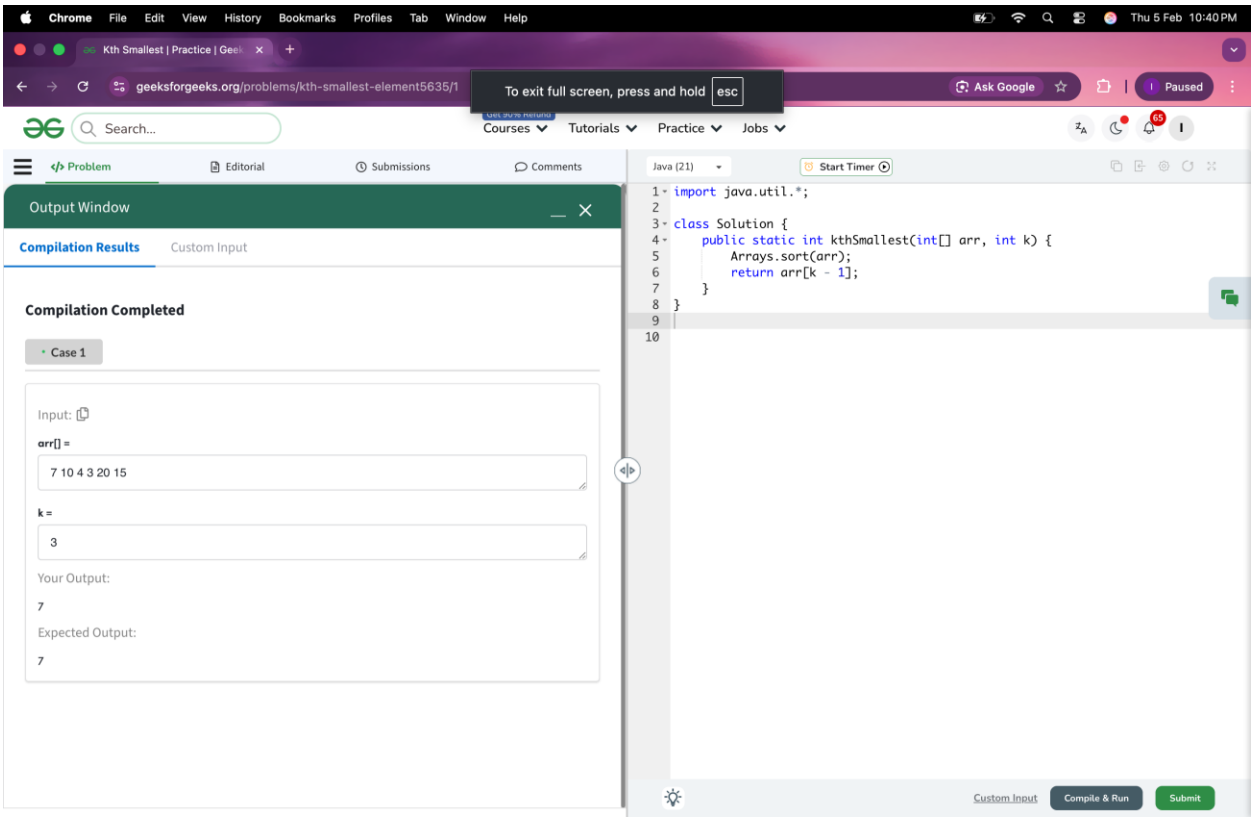
Explanation: 3rd smallest element in the given array is 7.

Constraints:

$1 \leq arr.size() \leq 105$

$1 \leq arr[i] \leq 105$

$1 \leq k \leq arr.size()$



EXPERIMENT 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 :

nput: 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

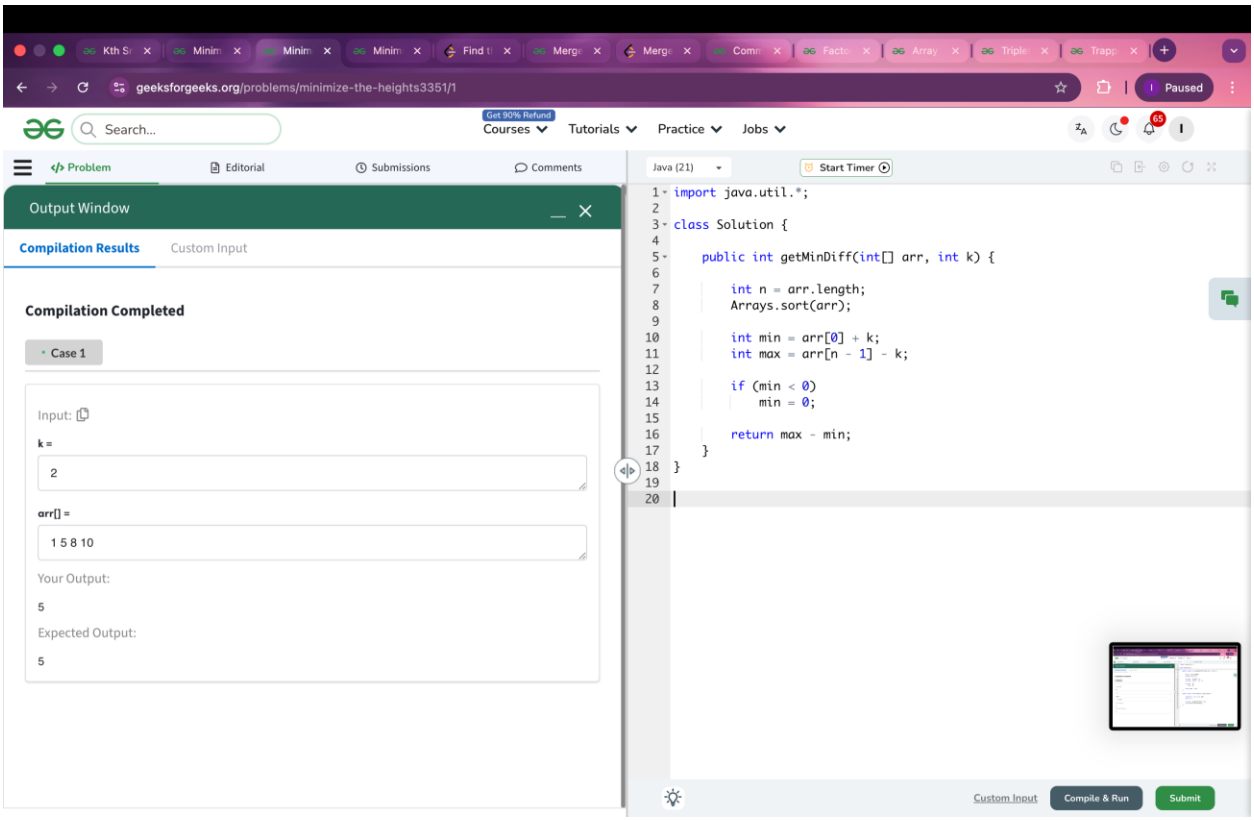
Explanation: The array can be modified as [3+k, 9+k, 12-k, 16-k, 20-k] = [6, 12, 9, 13, 17]. The difference between the largest and the smallest is 17-6 = 11.

Constraints

$1 \leq k \leq 107$

$1 \leq n \leq 105$

$1 \leq arr[i] \leq 107$



EXPERIMENT 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]

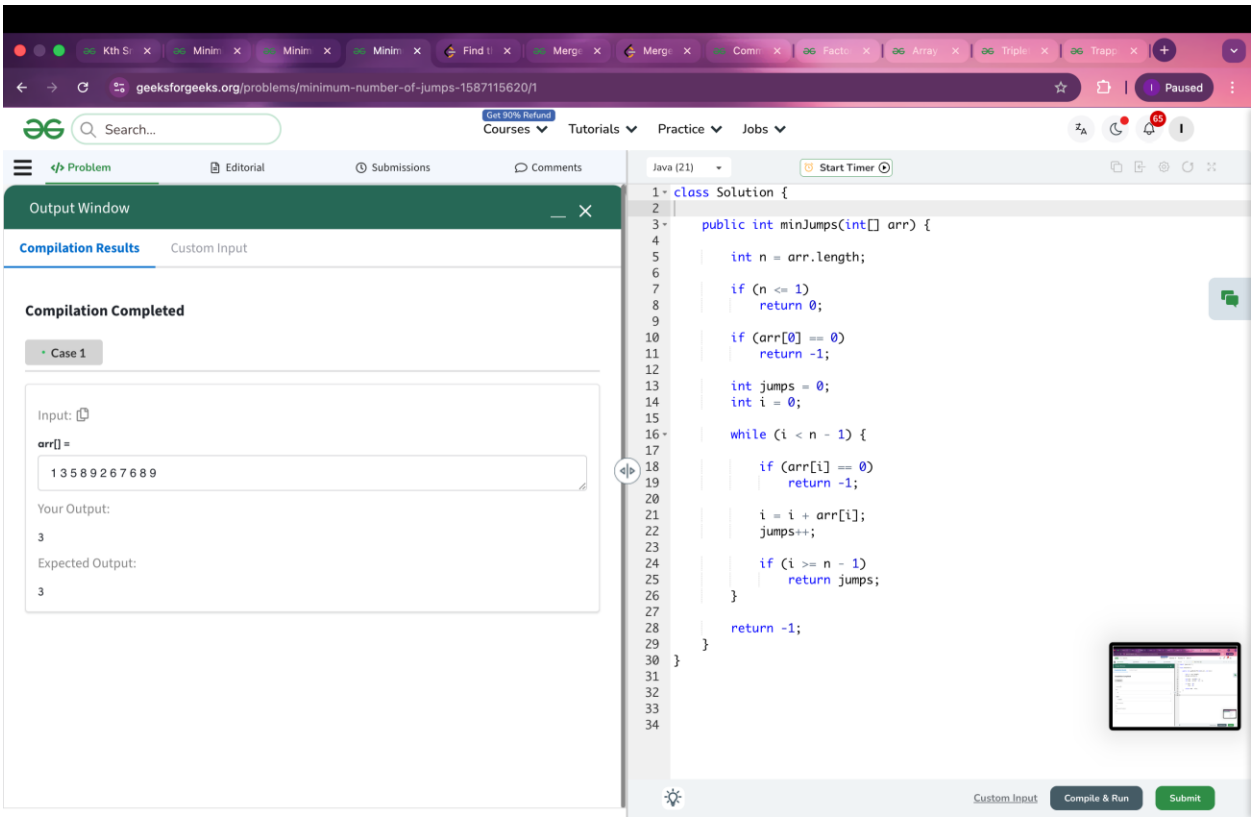
Output: -1

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

Constraints:

$2 \leq arr.size() \leq 105$

$0 \leq arr[i] \leq 105$



EXPERIMENT 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

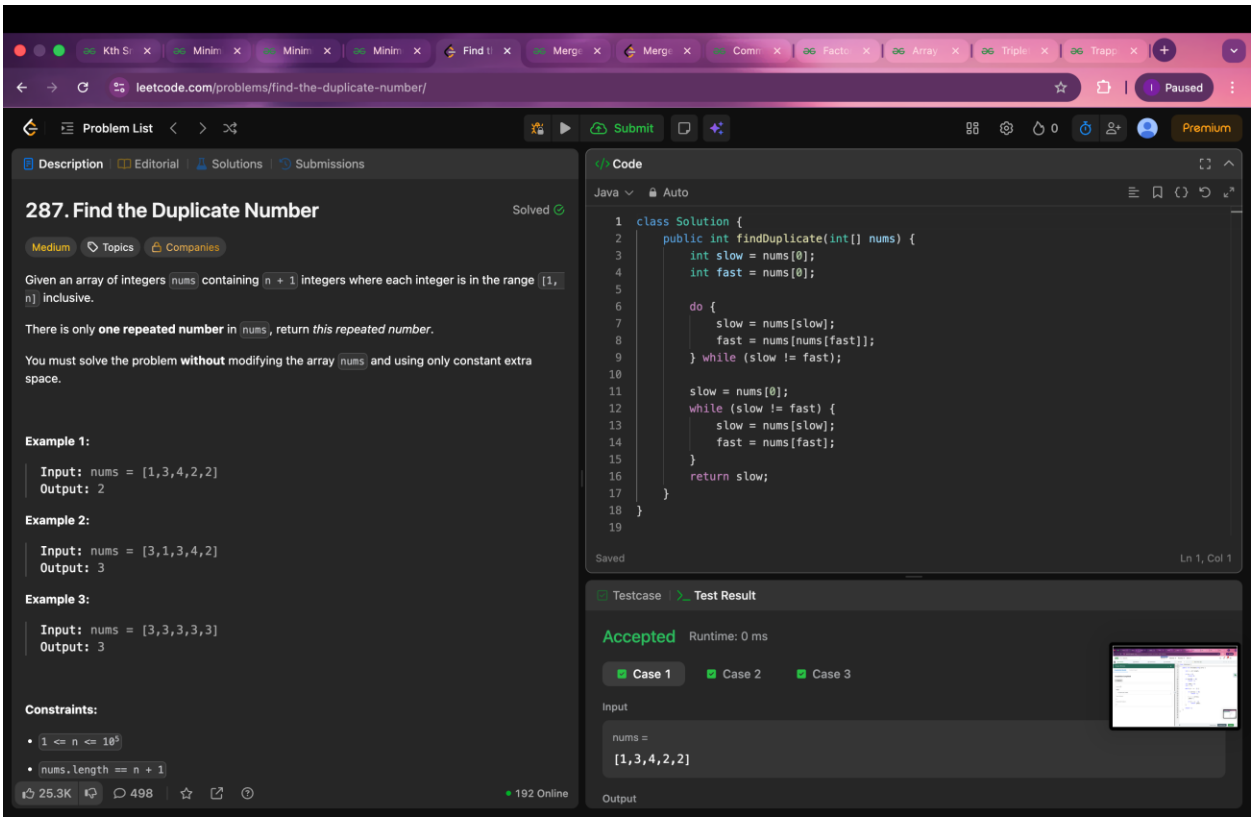
Constraints:

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.



EXPERIMENT 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]

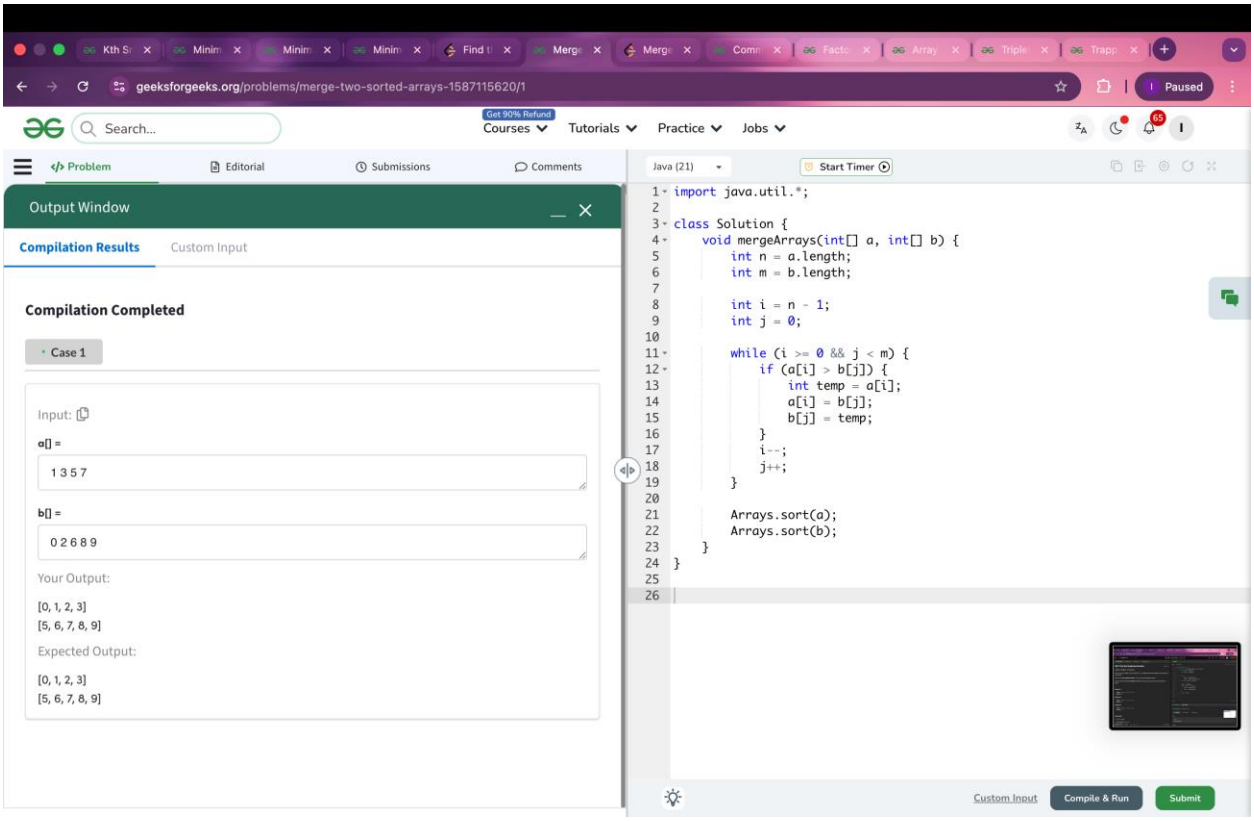
Output: a[] = [0, 1], b[] = [2, 3]

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

Constraints:

$1 \leq n, m \leq 105$

$0 \leq a[i], b[i] \leq 107$



EXPERIMENT 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]]

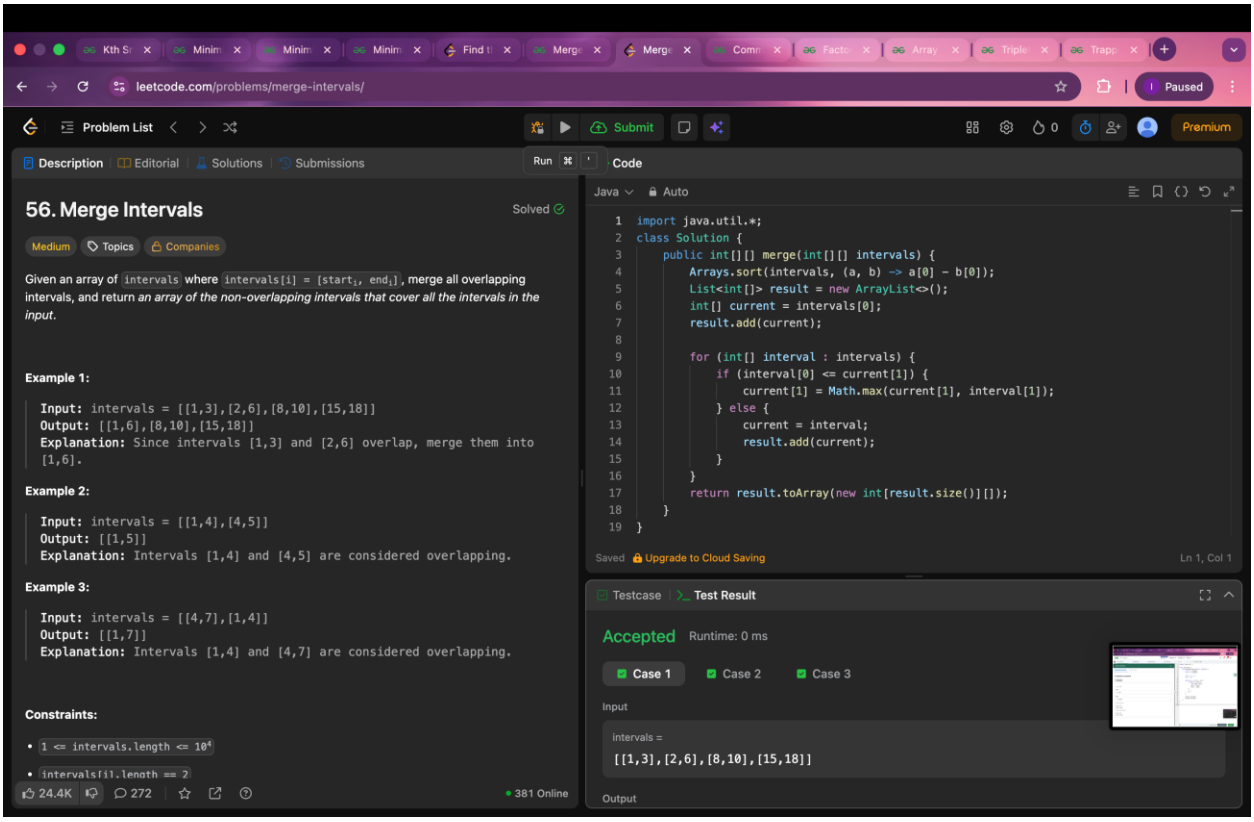
Explanation: Intervals [1,4] and [4,7] are considered overlapping.

Constraints:

1 <= intervals.length <= 10^4

intervals[i].length == 2

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



EXPERIMENT 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]

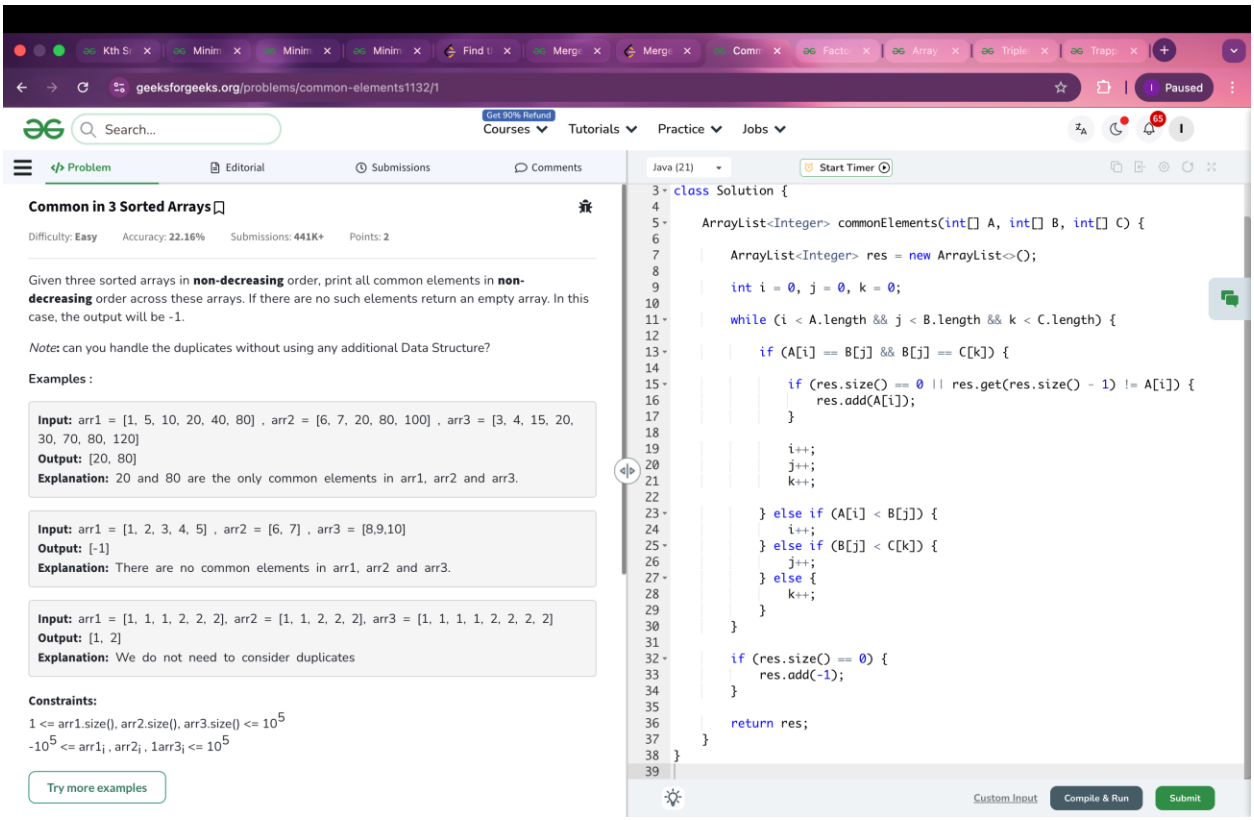
Output: [-1]

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



EXPERIMENT 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

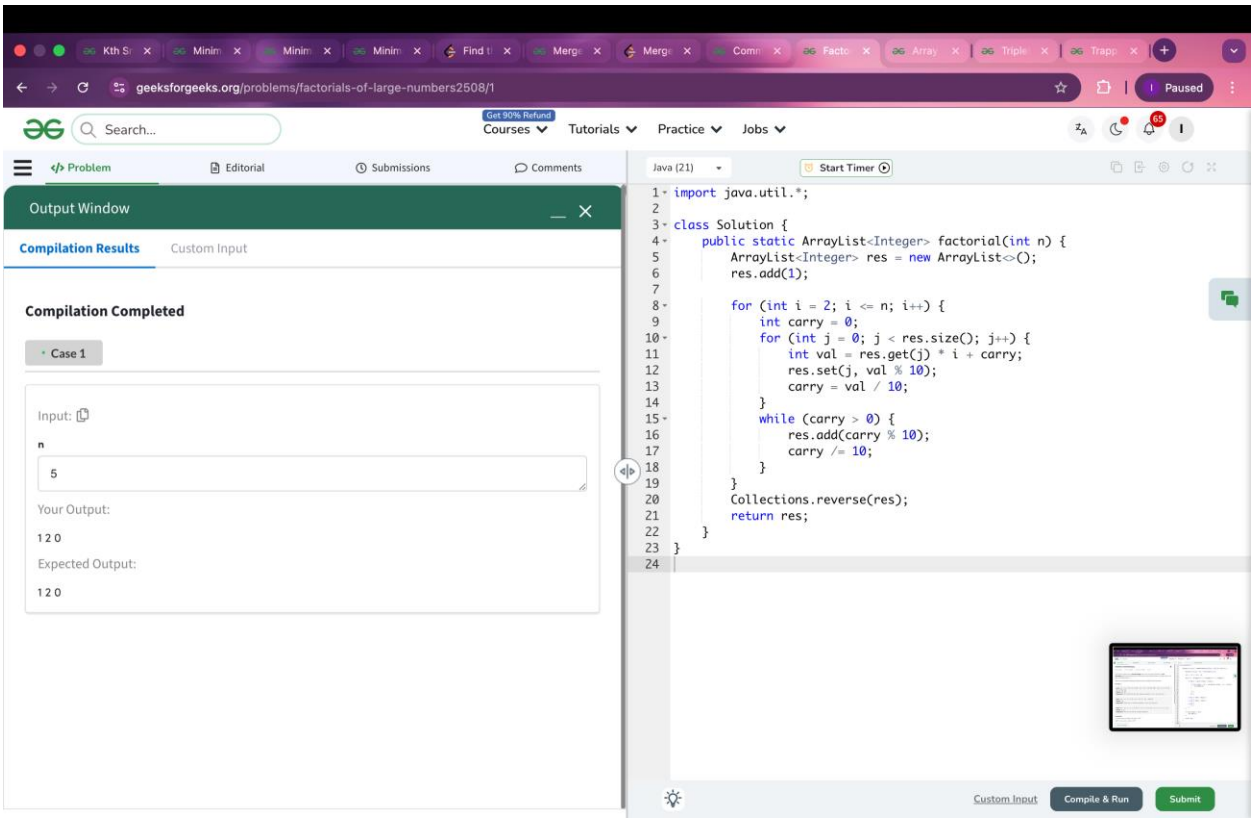
Output: [3, 6, 2, 8, 8, 0, 0]

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

Input: n = 1

Output: [1]

Explanation: 1! = 1



EXPERIMENT 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]

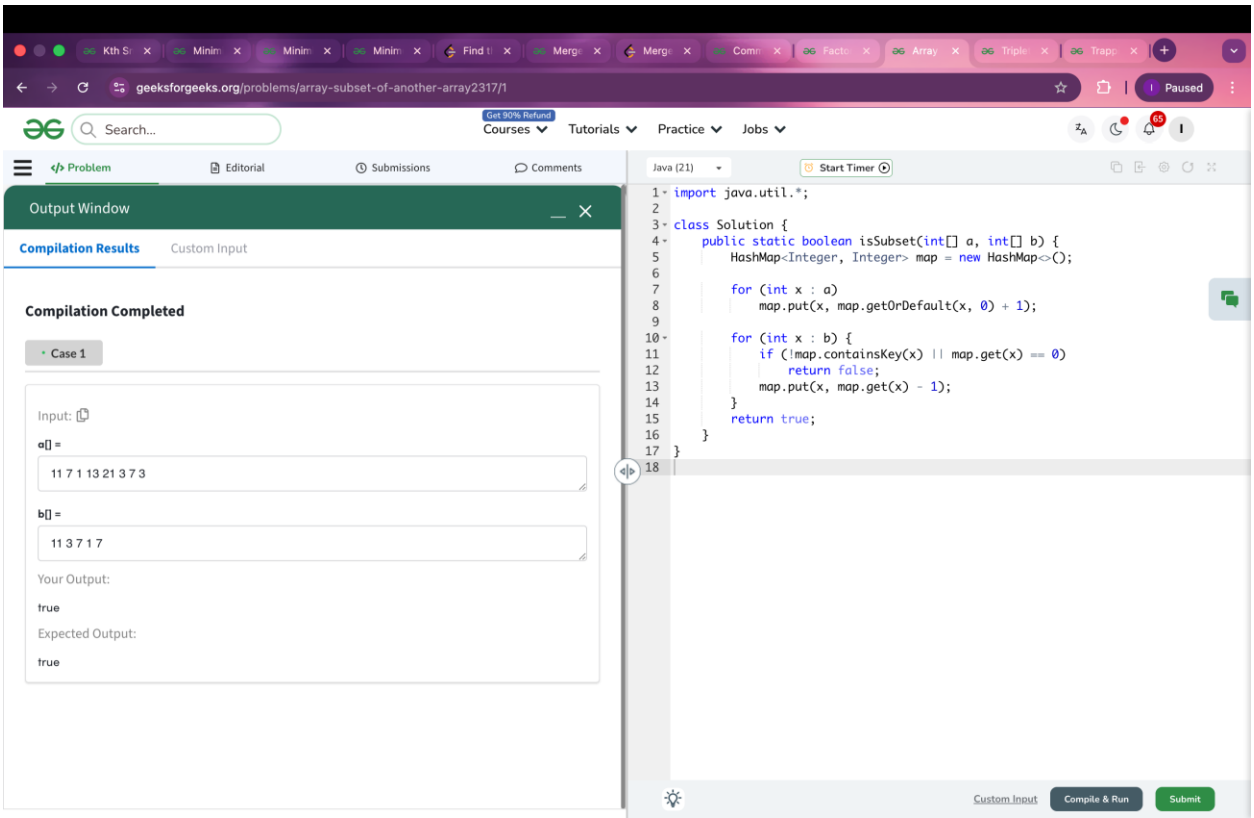
Output: true

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[]



EXPERIMENT 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

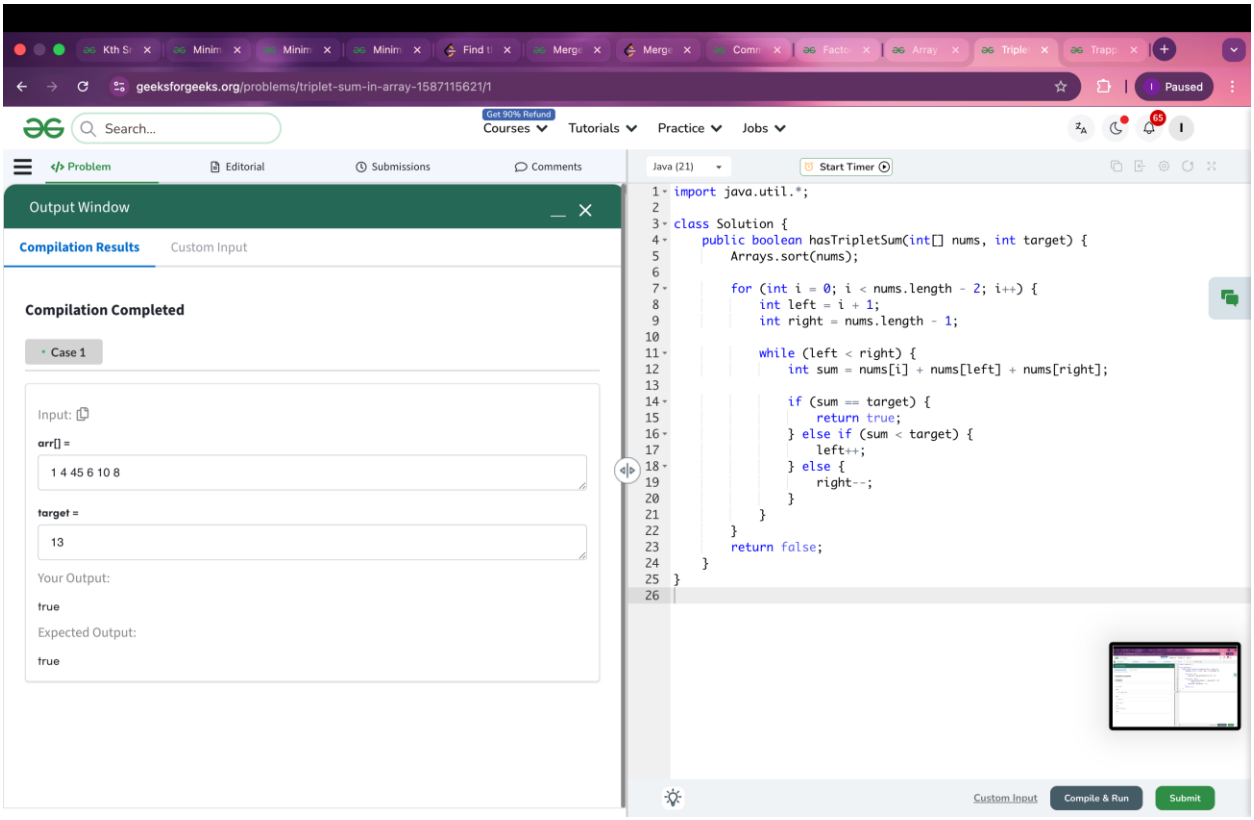
Output: true

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.



EXPERIMENT 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.

2

Take

Home

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]

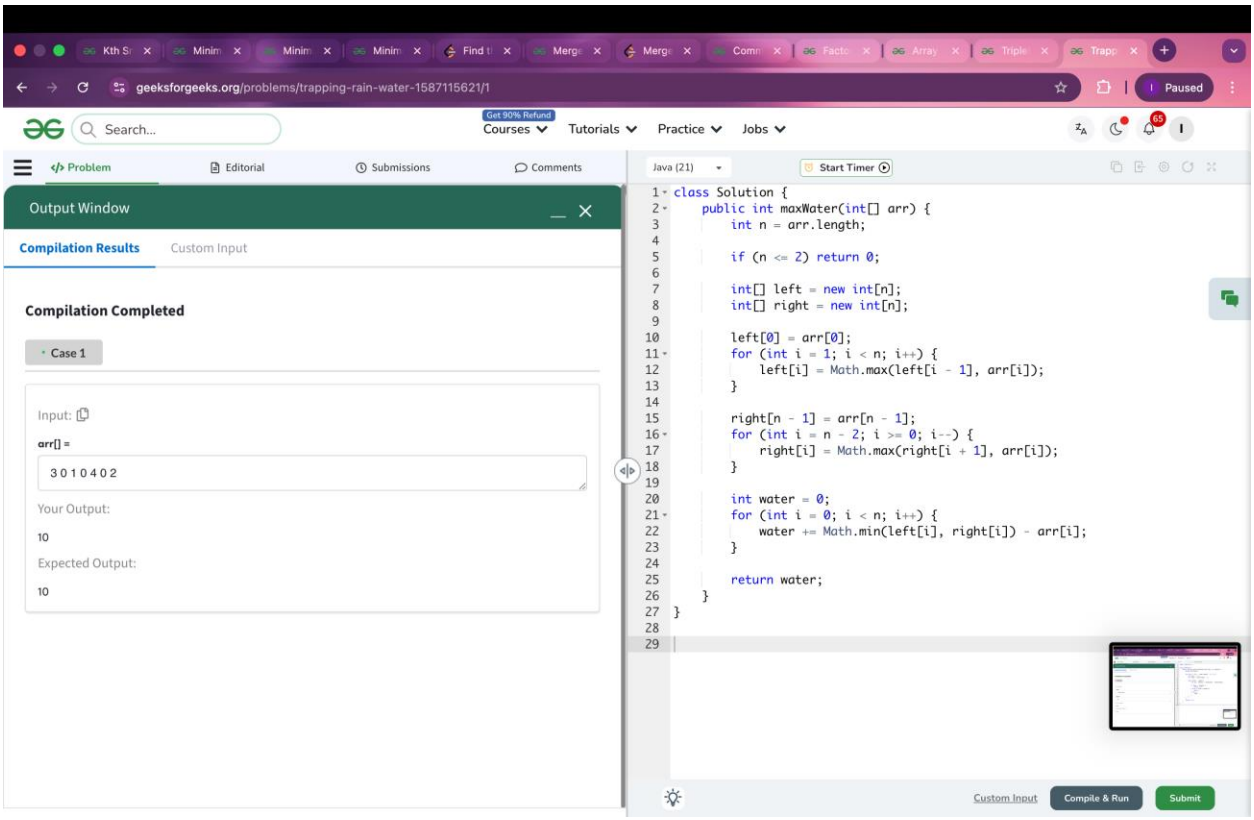
Output: 0

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.


geeksforgeeks.org/problems/trapping-rain-water-1587115621/1
Search...
Courses
Tutorials
Practice
Jobs
Problem
Editorial
Submissions
Comments
Output Window
Compilation Results
Custom Input
Compilation Completed
Case 1
Input:
arr[] =
3 0 1 0 4 0 2
Your Output:
10
Expected Output:
10
Java (21)
Start Timer
class Solution {
public int maxWater(int[] arr) {
int n = arr.length;
if (n <= 2) return 0;
int[] left = new int[n];
int[] right = new int[n];
left[0] = arr[0];
for (int i = 1; i < n; i++) {
left[i] = Math.max(left[i - 1], arr[i]);
}
right[n - 1] = arr[n - 1];
for (int i = n - 2; i >= 0; i--) {
right[i] = Math.max(right[i + 1], arr[i]);
}
int water = 0;
for (int i = 0; i < n; i++) {
water += Math.min(left[i], right[i]) - arr[i];
}
return water;
}
}
Custom Input
Compile & Run
Submit