

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 `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.

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

Output: 7

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

Constraints:

$1 \leq \text{arr.size()} \leq 105$

$1 \leq \text{arr}[i] \leq 105$

$1 \leq k \leq \text{arr.size}()$

The screenshot shows a web browser at `geeksforgeeks.org/problems/kth-smallest-element5635/1`. The IDE interface includes a top navigation bar with 'Courses', 'Tutorials', 'Practice', and 'Jobs'. The main area is split into two panes. The left pane, titled 'Output Window', shows 'Compilation Results' for 'Custom Input'. It indicates 'Compilation Completed' for 'Case 1'. The input is `arr[] = 7 10 4 3 20 15` and `k = 3`. The 'Your Output' is `7`, which matches the 'Expected Output' of `7`. The right pane shows the Java code for the solution:

```
1 import java.util.*;
2
3 class Solution {
4     public static int kthSmallest(int[] arr, int k) {
5         Arrays.sort(arr);
6         return arr[k - 1];
7     }
8 }
9
10
```

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

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 :

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

$$1 \leq k \leq 10^7$$

$$1 \leq n \leq 10^5$$

$$1 \leq \text{arr}[i] \leq 10^7$$

The screenshot displays the GeeksforGeeks online IDE interface. The browser address bar shows the URL `geeksforgeeks.org/problems/minimize-the-heights3351/1`. The IDE has a top navigation bar with links for Courses, Tutorials, Practice, and Jobs. The main editor area shows a Java solution for the 'Minimize the Heights' problem. The code is as follows:

```
1- import java.util.*;
2-
3- class Solution {
4-
5-     public int getMinDiff(int[] arr, int k) {
6-
7-         int n = arr.length;
8-         Arrays.sort(arr);
9-
10-        int min = arr[0] + k;
11-        int max = arr[n - 1] - k;
12-
13-        if (min < 0)
14-            min = 0;
15-
16-        return max - min;
17-    }
18- }
19-
20-
```

The Output Window on the left shows the compilation results. It indicates that the compilation was completed successfully. The input values are `k = 2` and `arr[] = 1 5 8 10`. The output shows `Your Output: 5` and `Expected Output: 5`.

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 $\text{arr}[i] = 3$, you can jump to index $i + 1$, $i + 2$, or $i + 3$ from position i .

If $\text{arr}[i] = 0$, 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: $\text{arr}[] = [1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9]$

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: $\text{arr} = [1, 4, 3, 2, 6, 7]$

Output: 2

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

Input: $\text{arr} = [0, 10, 20]$

Output: -1

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

Constraints:

$2 \leq \text{arr.size()} \leq 105$

$0 \leq \text{arr}[i] \leq 105$

The screenshot displays the GeeksforGeeks online IDE interface. The browser address bar shows the URL: [geeksforgeeks.org/problems/minimum-number-of-jumps-1587115620/1](https://www.geeksforgeeks.org/problems/minimum-number-of-jumps-1587115620/1). The IDE has a top navigation bar with links for Courses, Tutorials, Practice, and Jobs. The main editor area shows a Java solution for the 'Minimum Number of Jumps' problem. The code is as follows:

```
1 class Solution {
2
3 public int minJumps(int[] arr) {
4
5     int n = arr.length;
6
7     if (n <= 1)
8         return 0;
9
10    if (arr[0] == 0)
11        return -1;
12
13    int jumps = 0;
14    int i = 0;
15
16    while (i < n - 1) {
17
18        if (arr[i] == 0)
19            return -1;
20
21        i = i + arr[i];
22        jumps++;
23
24        if (i >= n - 1)
25            return jumps;
26    }
27
28    return -1;
29 }
30 }
31
32
33
34
```

The output window on the left shows the following details:

- Compilation Results:** Custom Input
- Compilation Completed**
- Case 1**
- Input:** `arr[] =`
`1 3 5 8 9 2 6 7 6 8 9`
- Your Output:** 3
- Expected Output:** 3

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

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:

$1 \leq n \leq 10^5$

`nums.length == n + 1`

$1 \leq \text{nums}[i] \leq n$

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

287. Find the Duplicate Number Solved

Medium Topics Companies

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:

- $1 \leq n \leq 10^5$
- `nums.length == n + 1`

25.3K 498 192 Online

```
1 class Solution {
2     public int findDuplicate(int[] nums) {
3         int slow = nums[0];
4         int fast = nums[0];
5
6         do {
7             slow = nums[slow];
8             fast = nums[nums[fast]];
9         } while (slow != fast);
10
11         slow = nums[0];
12         while (slow != fast) {
13             slow = nums[slow];
14             fast = nums[fast];
15         }
16         return slow;
17     }
18 }
19
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

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

Output:

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:

$1 \leq n, m \leq 105$

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

The screenshot shows the GeeksforGeeks website interface for the 'Merge Two Sorted Arrays' problem. The 'Output Window' on the left displays the 'Compilation Results' for 'Case 1'. It shows the input arrays `a[] = [1, 3, 5, 7]` and `b[] = [0, 2, 6, 8, 9]`, and the expected output arrays `[0, 1, 2, 3]` and `[5, 6, 7, 8, 9]`. The 'Code Editor' on the right shows the Java solution code, which implements a merge sort algorithm to merge the two arrays and then sorts them.

```
1 import java.util.*;
2
3 class Solution {
4     void mergeArrays(int[] a, int[] b) {
5         int n = a.length;
6         int m = b.length;
7
8         int i = n - 1;
9         int j = 0;
10
11         while (i >= 0 && j < m) {
12             if (a[i] > b[j]) {
13                 int temp = a[i];
14                 a[i] = b[j];
15                 b[j] = temp;
16             }
17             i--;
18             j++;
19         }
20
21         Arrays.sort(a);
22         Arrays.sort(b);
23     }
24 }
25
26
```

EXPERIMENT 6

Given an array of intervals where $\text{intervals}[i] = [\text{start}_i, \text{end}_i]$, merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

Example 1:

Input: $\text{intervals} = [[1,3],[2,6],[8,10],[15,18]]$

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

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

Example 2:

Input: $\text{intervals} = [[1,4],[4,5]]$

Output: $[[1,5]]$

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

Example 3:

Input: $\text{intervals} = [[4,7],[1,4]]$

Output: $[[1,7]]$

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

Constraints:

$1 \leq \text{intervals.length} \leq 10^4$

$\text{intervals}[i].\text{length} == 2$

$0 \leq \text{start}_i \leq \text{end}_i \leq 10^4$

56. Merge Intervals Solved

Medium Topics Companies

Given an array of intervals where $\text{intervals}[i] = [\text{start}_i, \text{end}_i]$, merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

Example 1:
Input: $\text{intervals} = [[1,3], [2,6], [8,10], [15,18]]$
Output: $[[1,6], [8,10], [15,18]]$
Explanation: Since intervals $[1,3]$ and $[2,6]$ overlap, merge them into $[1,6]$.

Example 2:
Input: $\text{intervals} = [[1,4], [4,5]]$
Output: $[[1,5]]$
Explanation: Intervals $[1,4]$ and $[4,5]$ are considered overlapping.

Example 3:
Input: $\text{intervals} = [[4,7], [1,4]]$
Output: $[[1,7]]$
Explanation: Intervals $[1,4]$ and $[4,7]$ are considered overlapping.

Constraints:

- $1 \leq \text{intervals.length} \leq 10^4$
- $\text{intervals}[i].\text{length} == 2$

24.4K 272 381 Online

Java

```
1 import java.util.*;
2 class Solution {
3     public int[][] merge(int[][] intervals) {
4         Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
5         List<int[]> result = new ArrayList<>();
6         int[] current = intervals[0];
7         result.add(current);
8
9         for (int[] interval : intervals) {
10             if (interval[0] <= current[1]) {
11                 current[1] = Math.max(current[1], interval[1]);
12             } else {
13                 current = interval;
14                 result.add(current);
15             }
16         }
17         return result.toArray(new int[result.size()][]);
18     }
19 }
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

$\text{intervals} = [[1,3], [2,6], [8,10], [15,18]]$

Output

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

The screenshot displays the GeeksforGeeks website interface. On the left, the problem 'Common in 3 Sorted Arrays' is detailed, including its difficulty (Easy), accuracy (22.16%), and submission count (441K+). The problem statement asks for common elements in three sorted arrays in non-decreasing order. Examples and constraints are provided. On the right, a Java solution is shown, implementing a three-pointer approach to find common elements across three arrays A, B, and C.

Common in 3 Sorted Arrays

Difficulty: Easy Accuracy: 22.16% Submissions: 441K+ Points: 2

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

Constraints:

$1 \leq \text{arr1.size()}, \text{arr2.size()}, \text{arr3.size()} \leq 10^5$
 $-10^5 \leq \text{arr1}_i, \text{arr2}_i, \text{arr3}_i \leq 10^5$

Try more examples

```

class Solution {
    ArrayList<Integer> commonElements(int[] A, int[] B, int[] C) {
        ArrayList<Integer> res = new ArrayList<>();
        int i = 0, j = 0, k = 0;
        while (i < A.length && j < B.length && k < C.length) {
            if (A[i] == B[j] && B[j] == C[k]) {
                if (res.size() == 0 || res.get(res.size() - 1) != A[i]) {
                    res.add(A[i]);
                }
                i++;
                j++;
                k++;
            } else if (A[i] < B[j]) {
                i++;
            } else if (B[j] < C[k]) {
                j++;
            } else {
                k++;
            }
        }
        if (res.size() == 0) {
            res.add(-1);
        }
        return res;
    }
}

```

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

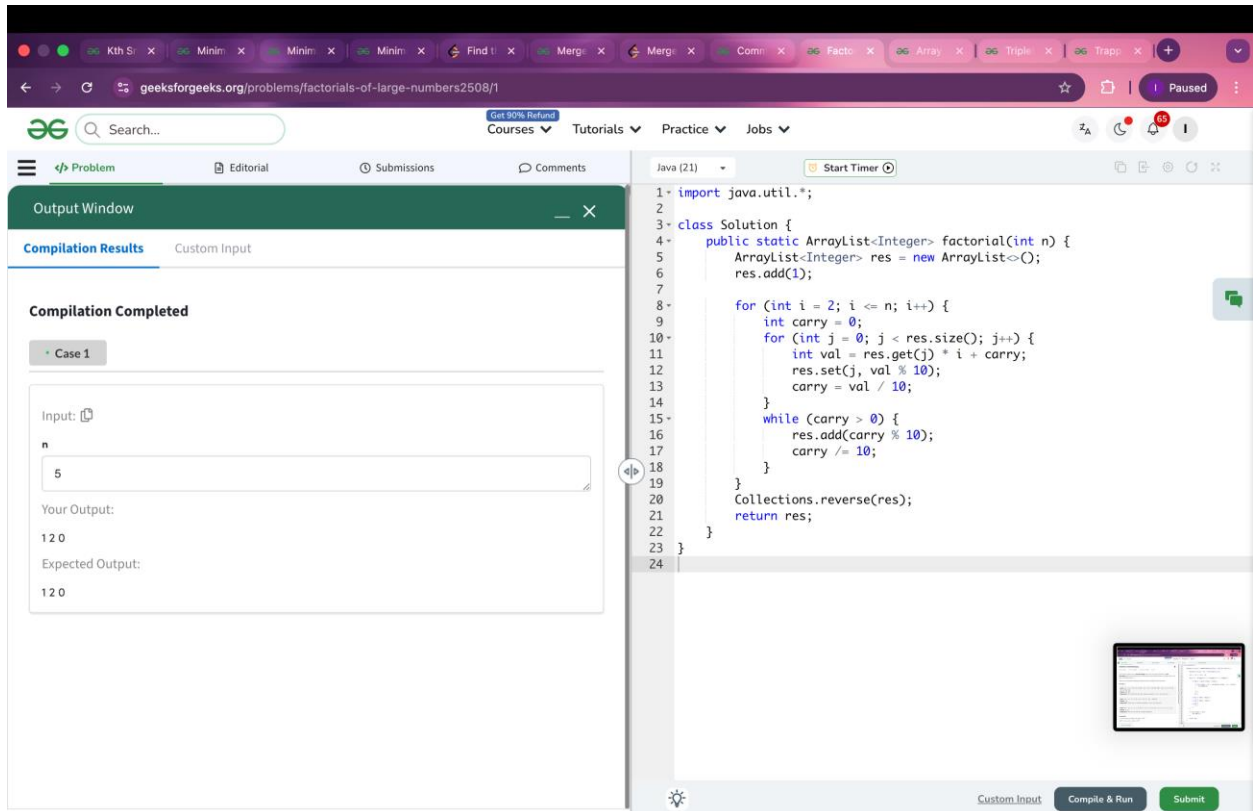
Output: [3, 6, 2, 8, 8, 0, 0]

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[]

The screenshot displays the GeeksforGeeks online IDE interface. On the left, the 'Output Window' shows 'Compilation Completed' for 'Case 1'. The input is defined as array `a[] = [11, 7, 1, 13, 21, 3, 7, 3]` and array `b[] = [11, 3, 7, 1, 7]`. The 'Your Output' is `true`, which matches the 'Expected Output'. On the right, the Java code (Java 21) is visible. It defines a `Solution` class with a static method `isSubset(int[] a, int[] b)`. This method uses a `HashMap<Integer, Integer>` to count the frequency of elements in array `a`. It then iterates through array `b`, checking if each element exists in the map with a count greater than zero. If any element is missing or its count is zero, it returns `false`; otherwise, it returns `true`.

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.

The screenshot shows the GeeksforGeeks online IDE interface. The browser address bar displays [geeksforgeeks.org/problems/triplet-sum-in-array-1587115621/1](https://www.geeksforgeeks.org/problems/triplet-sum-in-array-1587115621/1). The IDE has a top navigation bar with links for Courses, Tutorials, Practice, and Jobs. The main editor area shows a Java solution for the 'Triplet Sum in Array' problem. The code is as follows:

```
1- import java.util.*;
2-
3- class Solution {
4-     public boolean hasTripletSum(int[] nums, int target) {
5-         Arrays.sort(nums);
6-
7-         for (int i = 0; i < nums.length - 2; i++) {
8-             int left = i + 1;
9-             int right = nums.length - 1;
10-
11-             while (left < right) {
12-                 int sum = nums[i] + nums[left] + nums[right];
13-
14-                 if (sum == target) {
15-                     return true;
16-                 } else if (sum < target) {
17-                     left++;
18-                 } else {
19-                     right--;
20-                 }
21-             }
22-         }
23-         return false;
24-     }
25- }
```

On the left side, the 'Output Window' is open, showing 'Compilation Results' for 'Custom Input'. It indicates 'Compilation Completed' for 'Case 1'. The input is:

Input:

arr[] =
1 4 45 6 10 8

target =
13

Your Output:
true

Expected Output:
true

At the bottom right, there is a 'Submit' button and a 'Compile & Run' button.

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...

Get 90% Referral 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

```
1 class Solution {
2     public int maxWater(int[] arr) {
3         int n = arr.length;
4
5         if (n <= 2) return 0;
6
7         int[] left = new int[n];
8         int[] right = new int[n];
9
10        left[0] = arr[0];
11        for (int i = 1; i < n; i++) {
12            left[i] = Math.max(left[i - 1], arr[i]);
13        }
14
15        right[n - 1] = arr[n - 1];
16        for (int i = n - 2; i >= 0; i--) {
17            right[i] = Math.max(right[i + 1], arr[i]);
18        }
19
20        int water = 0;
21        for (int i = 0; i < n; i++) {
22            water += Math.min(left[i], right[i]) - arr[i];
23        }
24
25        return water;
26    }
27 }
28
29
```

Custom Input Compile & Run Submit