

## JAVA PRACTICE SHEET(26/01/26)

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:

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

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

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

Sol

The screenshot displays a coding environment with a Java solution for finding the `k`th smallest element in an array. The code is as follows:

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

The output window on the left shows the following results:

- Problem Solved Successfully ✓
- Test Cases Passed: 1121 / 1121
- Attempts: Correct / Total: 1 / 1
- Accuracy: 100%
- Points Scored: 4 / 4
- Time Taken: 0.67
- Your Total Score: 14 ↑

Below the results, there are buttons for "Solve Next" with options: "Smallest Positive Missing", "Valid Pair Sum", and "Optimal Array". At the bottom, there is a banner for "Build 21 Projects in 21 Days" with the text "Build real-world ML, Deep Learning & Gen AI projects".

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:  $k = 2$ ,  $\text{arr}[] = [1, 5, 8, 10]$

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:  $k = 3$ ,  $\text{arr}[] = [3, 9, 12, 16, 20]$

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

$1 \leq k \leq 10^7$

$1 \leq n \leq 10^5$

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

**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

```

1- import java.util.Arrays;
2- class Solution {
3-     public int getMinDiff(int[] arr, int k) {
4-         // code here
5-         int n = arr.length;
6-         Arrays.sort(arr);
7-         int ans = arr[n - 1] - arr[0];
8-         int smallest = arr[0] + k;
9-         int largest = arr[n - 1] - k;
10-        for (int i = 1; i < n; i++) {
11-            if (arr[i] - k < 0) {
12-                continue;
13-            }
14-            int minHeight = Math.min(smallest, arr[i] - k);
15-            int maxHeight = Math.max(arr[i - 1] + k, largest);
16-            ans = Math.min(ans, maxHeight - minHeight);
17-        }
18-        return ans;
19-    }
20- }
21-
22-
23-
24-

```

Custom Input **Compile & Run** **Submit**

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:

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

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

The screenshot shows a coding platform interface. On the left, the 'Output Window' displays 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. It states 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1120 / 1120', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Time Taken: 0.68'. There are buttons for 'Solve Next', 'Maximum Index', 'Jump Game', and 'Wine Buying and Selling'. At the bottom, it says 'Stay Ahead With: Build 21 Projects in 21 Days' with a small icon. On the right, the code editor shows a C++ solution for the 'Jump Game' problem. The code defines a class 'Solution' with a method 'minJumps' that takes an array 'arr' and returns the minimum number of jumps to reach the end. The code uses a greedy approach, maintaining 'maxReach' and 'steps' variables. The code is as follows:

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

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:

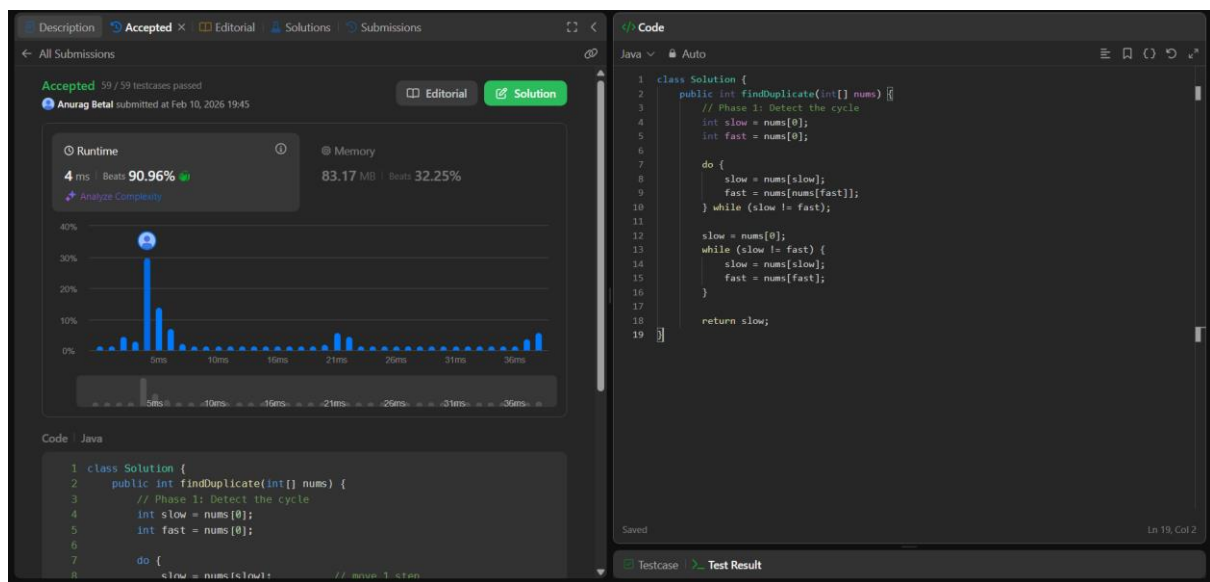
$1 \leq n \leq 105$

$\text{nums.length} == n + 1$

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

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

**Sol**



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

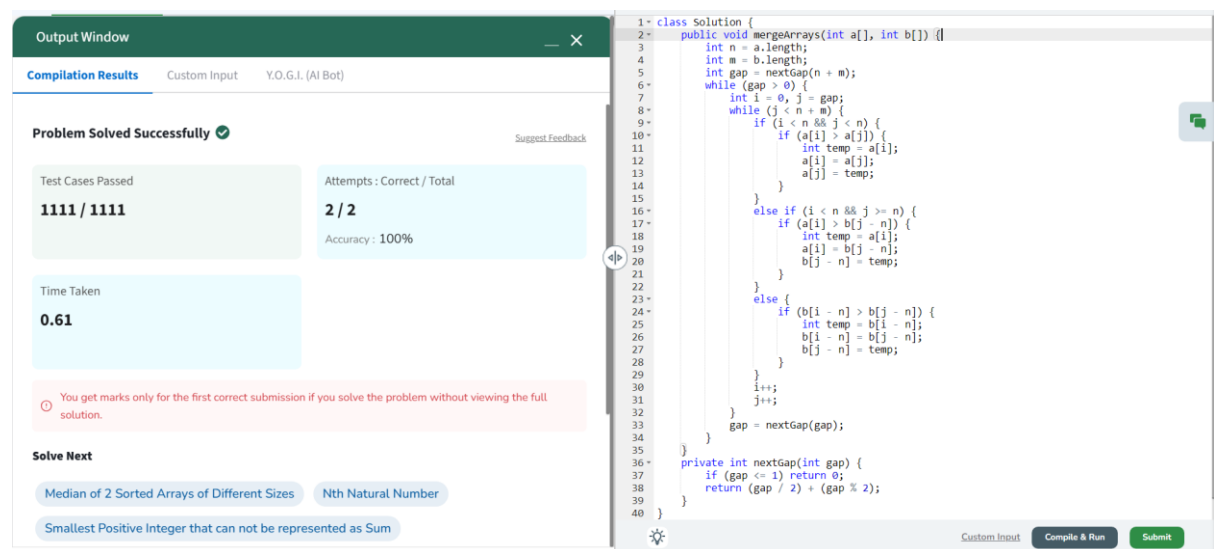
Explanation: After merging two sorted arrays we get  $[0, 1, 2, 3]$ .

Constraints:

$1 \leq n, m \leq 105$

$0 \leq a[i], b[i] \leq 10^7$

**Sol**



```
1 class Solution {
2     public void mergeArrays(int a[], int b[]) {
3         int n = a.length;
4         int m = b.length;
5         int gap = nextGap(n + m);
6         while (gap > 0) {
7             int i = 0, j = gap;
8             while (j < n + m) {
9                 if (i < n && j < n) {
10                     if (a[i] > a[j]) {
11                         int temp = a[i];
12                         a[i] = a[j];
13                         a[j] = temp;
14                     }
15                 } else if (i < n && j >= n) {
16                     if (a[i] > b[j - n]) {
17                         int temp = a[i];
18                         a[i] = b[j - n];
19                         b[j - n] = temp;
20                     }
21                 } else {
22                     if (b[i - n] > b[j - n]) {
23                         int temp = b[i - n];
24                         b[i - n] = b[j - n];
25                         b[j - n] = temp;
26                     }
27                 }
28                 i++;
29                 j++;
30             }
31             gap = nextGap(gap);
32         }
33     }
34     private int nextGap(int gap) {
35         if (gap <= 1) return 0;
36         return (gap / 2) + (gap % 2);
37     }
38 }
```

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

Output: `[[1,7]]`

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

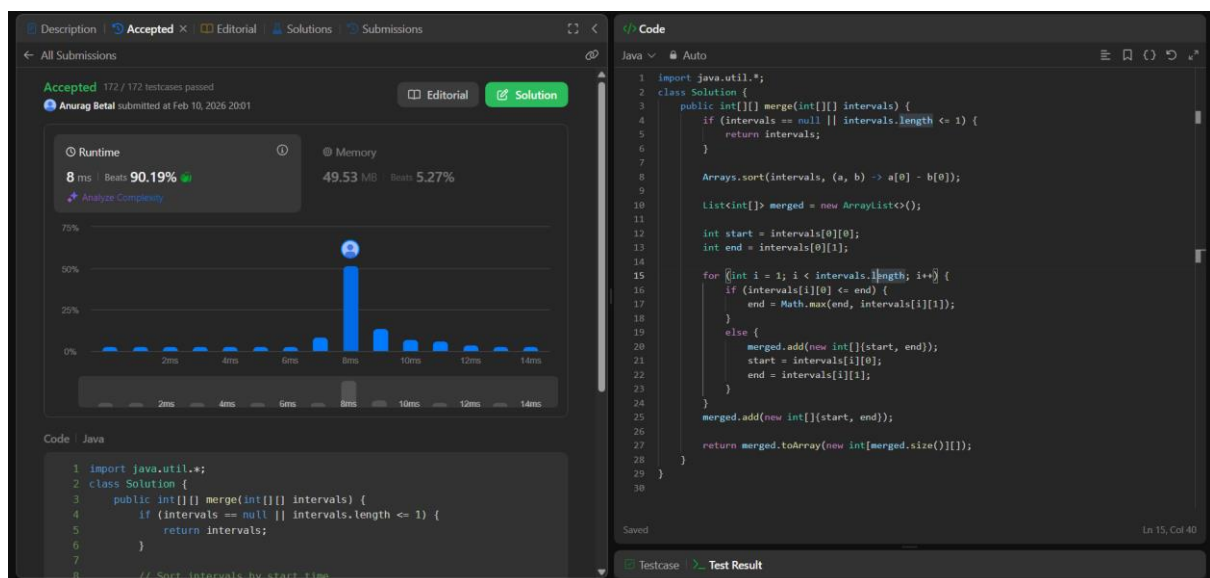
Constraints:

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

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

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

**Sol**



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

The screenshot displays a coding platform interface. On the left, the 'Output Window' shows 'Problem Solved Successfully' with 1215/1215 test cases passed, 2/2 points scored, and a time taken of 3.35. Below this, there are buttons for 'Solve Next' and 'Stay Ahead With: Build 21 Projects in 21 Days'. On the right, a code editor shows a Java solution for finding common elements in three arrays. The code uses nested loops to compare elements across three arrays and returns the common elements in a list.

```
1 import java.util.*;
2 class Solution {
3     // Function to find common elements in three arrays.
4     public List<Integer> commonElements(List<Integer> arr1, List<Integer> arr2,
5                                         List<Integer> arr3){
6
7         int i = 0, j = 0, k = 0;
8         List<Integer> result = new ArrayList<>();
9         int n1 = arr1.size();
10        int n2 = arr2.size();
11        int n3 = arr3.size();
12        while (i < n1 && j < n2 && k < n3) {
13            int a = arr1.get(i);
14            int b = arr2.get(j);
15            int c = arr3.get(k);
16            if (a == b && b == c) {
17                result.add(a);
18                while (i < n1 && arr1.get(i) == a) i++;
19                while (j < n2 && arr2.get(j) == a) j++;
20                while (k < n3 && arr3.get(k) == a) k++;
21            }
22            else {
23                int min = Math.min(a, Math.min(b, c));
24                if (a == min) i++;
25                else if (b == min) j++;
26                else k++;
27            }
28        }
29        if (result.isEmpty()) {
30            result.add(-1);
31        }
32        return result;
33    }
34 }
```

## 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**

The screenshot displays a coding platform interface. On the left, the 'Output Window' shows 'Compilation Results' for a problem solved successfully. It indicates that all test cases passed (1111 / 1111), with 1 correct attempt out of 1 total, 100% accuracy, 4 out of 4 points scored, and a time taken of 0.56 seconds. The user's total score is 26. Below this, there are buttons for 'Solve Next' (Large Factorial, Number following a pattern, Rank The Permutations) and a section 'Stay Ahead With: Build 21 Projects in 21 Days'. On the right, the code editor shows a Java solution for calculating the factorial of a number  $n$ . The code uses an ArrayList to store the digits of the factorial, calculates the factorial iteratively, and then reverses the list to return the correct order of digits.

```
1- import java.util.ArrayList;
2- import java.util.Collections;
3- // User function Template for Java
4-
5- class Solution {
6-     public static ArrayList<Integer> factorial(int n) {
7-         // code here
8-         ArrayList<Integer> result = new ArrayList<>();
9-         result.add(1); // 1! = 1
10-
11-         for (int x = 2; x <= n; x++) {
12-             int carry = 0;
13-             for (int i = 0; i < result.size(); i++) {
14-                 int prod = result.get(i) * x + carry;
15-                 result.set(i, prod % 10);
16-                 carry = prod / 10;
17-             }
18-
19-             while (carry > 0) {
20-                 result.add(carry % 10);
21-                 carry /= 10;
22-             }
23-
24-             Collections.reverse(result);
25-             return result;
26-         }
27-     }
28- }
29-
```

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**

Output Window

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

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:

```
1 import java.util.HashMap;
2 class Solution {
3     public boolean isSubset(int a[], int b[]) {
4         // Your code here
5         HashMap<Integer, Integer> map = new HashMap<>();
6         for (int x : a) {
7             map.put(x, map.getOrDefault(x, 0) + 1);
8         }
9         for (int x : b) {
10            if (!map.containsKey(x) || map.get(x) == 0) {
11                return false;
12            }
13            map.put(x, map.get(x) - 1);
14        }
15        return true;
16    }
17 }
18
19
20
```

Custom Input Compile & Run Submit

## 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:**

```

1 import java.util.*;
2 class Solution {
3     // code here
4     public boolean hasTripletSum(int arr[], int target) {
5         int n = arr.length;
6         if (n < 3) return false;
7         Arrays.sort(arr);
8         for (int i = 0; i < n - 2; i++) {
9             int left = i + 1;
10            int right = n - 1;
11
12            while (left < right) {
13                int sum = arr[i] + arr[left] + arr[right];
14
15                if (sum == target) {
16                    return true;
17                } else if (sum < target) {
18                    left++;
19                } else {
20                    right--;
21                }
22            }
23        }
24        return false;
25    }
26 }
27

```

Custom Input Compile & Run Submit

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**

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

8 / 8

Your Total Score: 45

Time Taken

0.26

Suggest Feedback


Solve Next

Longest Arithmetic Subsequence

Rod Cutting

Jump Game

Stay Ahead With:



Build 21 Projects in 21 Days

Build real-world ML, Deep Learning & Gen AI projects

```
1- class Solution {
2-     public int maxWater(int arr[]) {
3-         // code here
4-         int n = arr.length;
5-         if (n == 0) return 0;
6-
7-         int left = 0, right = n - 1;
8-         int leftMax = 0, rightMax = 0;
9-         int water = 0;
10-
11-         while (left < right) {
12-             if (arr[left] <= arr[right]) {
13-                 if (arr[left] >= leftMax) {
14-                     leftMax = arr[left];
15-                 } else {
16-                     water += leftMax - arr[left];
17-                 }
18-                 left++;
19-             } else {
20-                 if (arr[right] >= rightMax) {
21-                     rightMax = arr[right];
22-                 } else {
23-                     water += rightMax - arr[right];
24-                 }
25-                 right--;
26-             }
27-         }
28-         return water;
29-     }
30- }
31-
32-
33-
```

Custom Input

Compile & Run

Submit