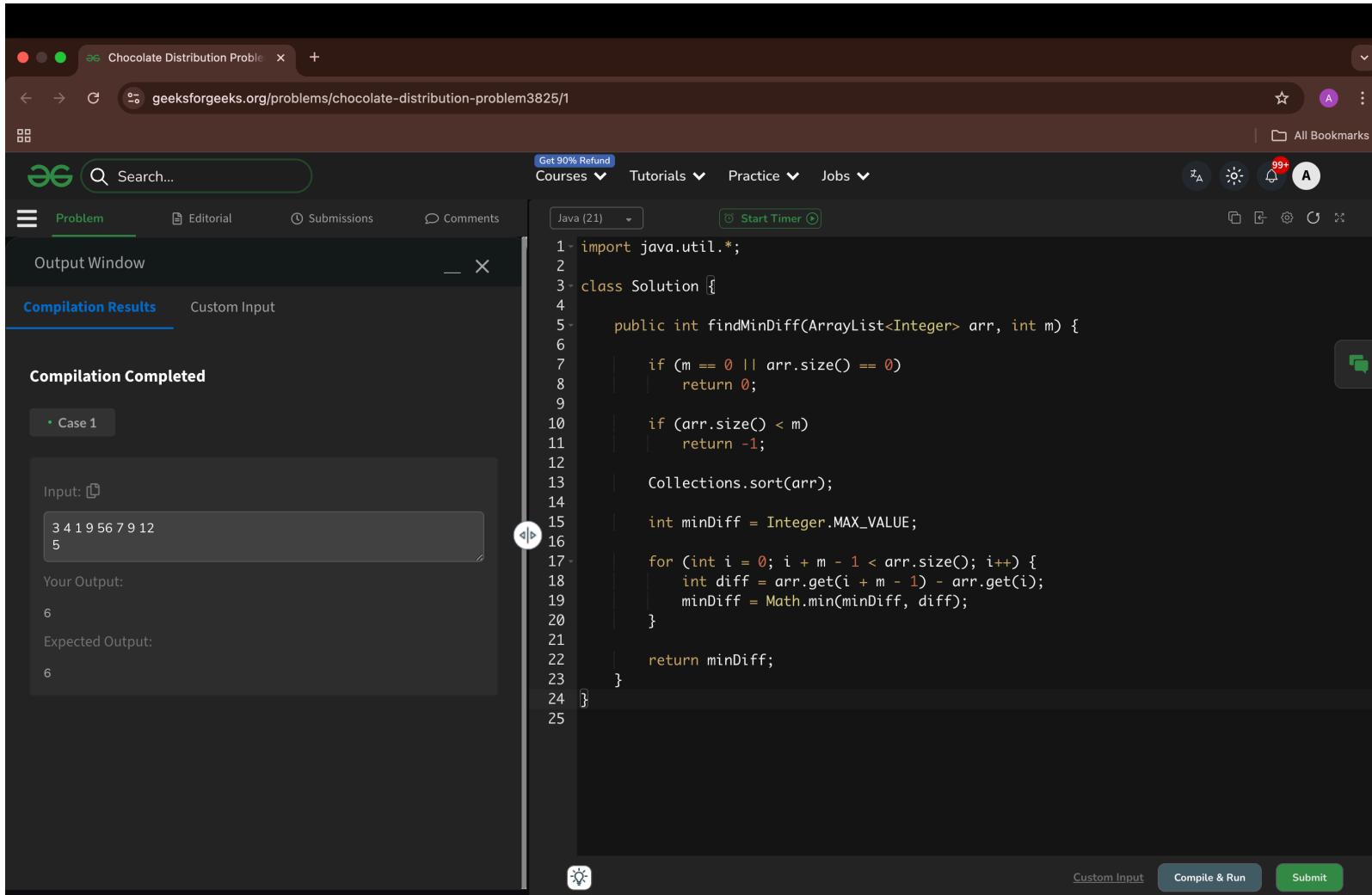


Name - Anisha Raj
Roll no. - 2410031455
Section - 2CSE8

Java Experiment-2



The screenshot shows a Java code editor interface on the GeeksforGeeks website. The title bar says "Chocolate Distribution Problem" and the URL is "geeksforgeeks.org/problems/chocolate-distribution-problem3825/1". The interface includes a search bar, navigation links for Courses, Tutorials, Practice, and Jobs, and a toolbar with various icons. The main area shows a Java code editor with the following code:

```
1 import java.util.*;
2
3 class Solution {
4
5     public int findMinDiff(ArrayList<Integer> arr, int m) {
6
7         if (m == 0 || arr.size() == 0)
8             return 0;
9
10        if (arr.size() < m)
11            return -1;
12
13        Collections.sort(arr);
14
15        int minDiff = Integer.MAX_VALUE;
16
17        for (int i = 0; i + m - 1 < arr.size(); i++) {
18            int diff = arr.get(i + m - 1) - arr.get(i);
19            minDiff = Math.min(minDiff, diff);
20        }
21
22        return minDiff;
23    }
24
25 }
```

The code is for a Java class named "Solution" that contains a single method "findMinDiff". The method takes an ArrayList of integers and an integer "m" as parameters. It first checks if "m" is 0 or if the array size is 0, in which case it returns 0. If the array size is less than "m", it returns -1. Otherwise, it sorts the array and then iterates through it to find the minimum difference between any two elements that are "m" units apart. The code uses the "Collections.sort" method and the "Math.min" method to achieve this. The code editor also shows an "Output Window" and a "Compilation Results" section which displays "Compilation Completed" and "Case 1". The "Input" field contains "3 4 1 9 5 6 7 9 12" and "5". The "Your Output" field shows "6" and the "Expected Output" field also shows "6".

Name - Anisha Raj
Roll no. - 2410031455
Section - 2CSE8

Java Experiment-2

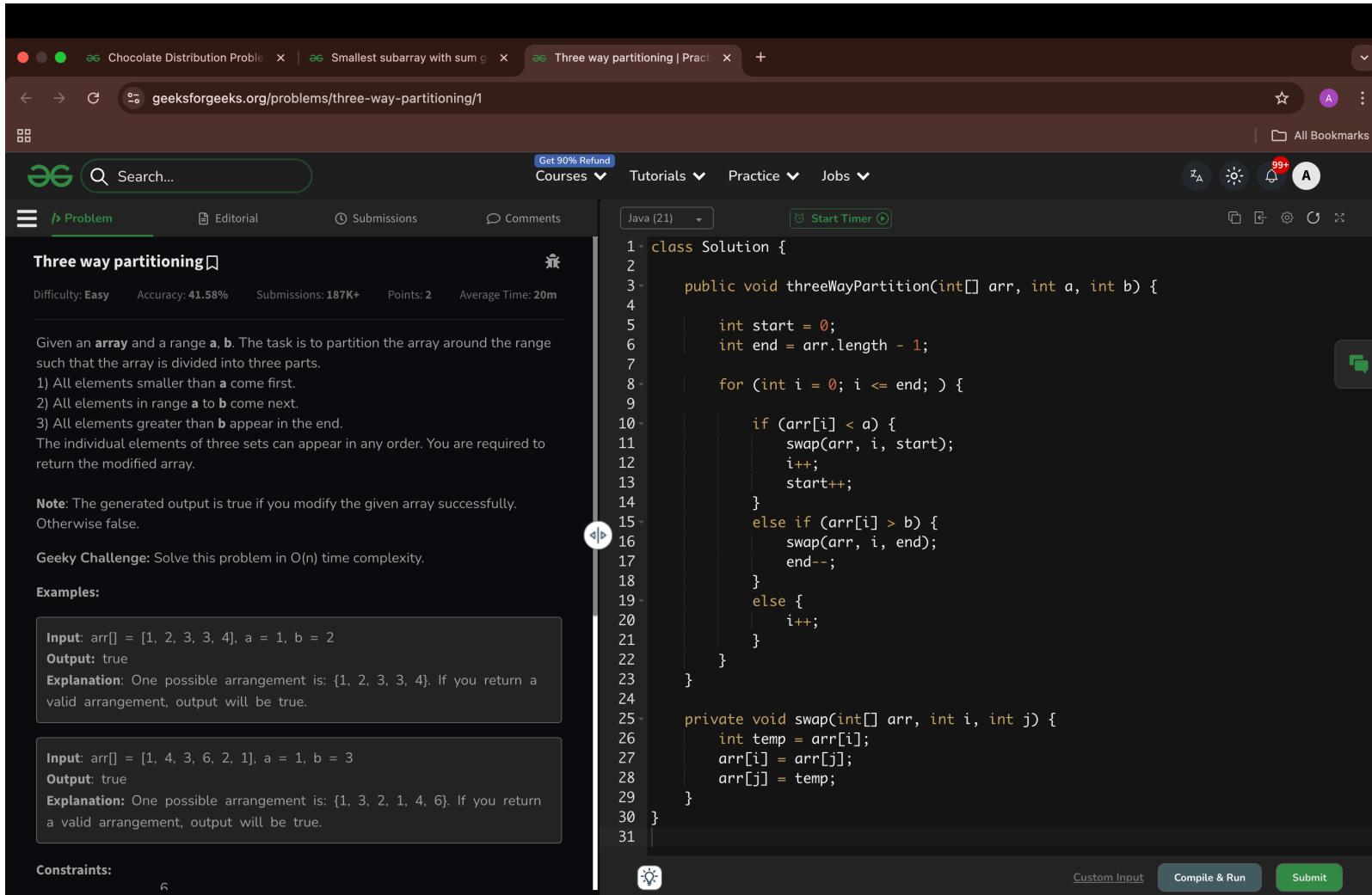
The screenshot shows a Java code editor on the GeeksforGeeks website. The problem is titled "Smallest subarray with sum greater than x". The code implements a sliding window approach to find the minimum length subarray with a sum greater than a given value x. The editor interface includes tabs for Java (21), Start Timer, and various tool icons. The code is as follows:

```
1 class Solution {
2     public int smallestSubWithSum(int x, int[] arr) {
3         int n = arr.length;
4         int minLength = Integer.MAX_VALUE;
5         int start = 0, currSum = 0;
6
7         for (int end = 0; end < n; end++) {
8             currSum += arr[end];
9
10            while (currSum > x) {
11                minLength = Math.min(minLength, end - start + 1);
12                currSum -= arr[start];
13                start++;
14            }
15
16            if (minLength == Integer.MAX_VALUE) {
17                return 0;
18            }
19        }
20
21        return minLength;
22    }
23 }
24 }
```

The problem description and examples are visible on the left side of the editor.

Name - Anisha Raj
Roll no. - 2410031455
Section - 2CSE8

Java Experiment-2



The screenshot shows a Java coding environment on the GeeksforGeeks website. The problem being solved is 'Three way partitioning'. The code is as follows:

```
1 class Solution {
2     public void threeWayPartition(int[] arr, int a, int b) {
3         int start = 0;
4         int end = arr.length - 1;
5
6         for (int i = 0; i <= end; ) {
7
8             if (arr[i] < a) {
9                 swap(arr, i, start);
10                i++;
11                start++;
12            }
13            else if (arr[i] > b) {
14                swap(arr, i, end);
15                end--;
16            }
17            else {
18                i++;
19            }
20        }
21
22    }
23
24
25    private void swap(int[] arr, int i, int j) {
26        int temp = arr[i];
27        arr[i] = arr[j];
28        arr[j] = temp;
29    }
30
31}
```

The code implements a three-way partitioning algorithm for an array. It uses three pointers: 'start' for elements less than 'a', 'end' for elements greater than 'b', and 'i' for the current element being processed. The algorithm moves elements less than 'a' to the start of the array, elements greater than 'b' to the end, and elements between 'a' and 'b' to their original positions.

Name - Anisha Raj
Roll no. - 2410031455
Section - 2CSE8

Java Experiment-2

The screenshot shows a Java code editor within a web browser window. The URL is geeksforgeeks.org/problems/minimum-swaps-required-to-bring-all-elements-less-than-or-equal-to-k-together4847/1. The code is a Java solution for the 'Minimum swaps and K together' problem, which involves bringing all elements less than or equal to a given value k to the front of an array with the minimum number of swaps. The code uses a sliding window approach to count the number of elements greater than k in each window and adjust the count as elements are moved. The code editor includes syntax highlighting for Java, a code editor interface with tabs, and a toolbar with various icons.

```
1 - class Solution {
2 -     public int minSwap(int[] arr, int k) {
3 -         int n = arr.length;
4 -         int good = 0;
5 -         // Count elements <= k
6 -         for (int i = 0; i < n; i++) {
7 -             if (arr[i] <= k)
8 -                 good++;
9 -         }
10 -        int bad = 0;
11 -        // Count bad elements in first window
12 -        for (int i = 0; i < good; i++) {
13 -            if (arr[i] > k)
14 -                bad++;
15 -        }
16 -        int ans = bad;
17 -        // Slide window
18 -        for (int i = 0, j = good; j < n; i++, j++) {
19 -            if (arr[i] > k)
20 -                bad--;
21 -            if (arr[j] > k)
22 -                bad++;
23 -        }
24 -        ans = Math.min(good, bad);
25 -        return ans;
26 -    }
27 -}
```

Minimum swaps and K together

Difficulty: Medium Accuracy: 26.0% Submissions: 141K+ Points: 4

Given an array **arr** and a number **k**. One can apply a swap operation on the array any number of times, i.e. choose any two index **i** and **j** ($i < j$) and swap $arr[i]$, $arr[j]$. Find the **minimum** number of swaps required to bring all the numbers less than or equal to **k** together, i.e. make them a contiguous subarray.

Examples:

Input: arr[] = [2, 1, 5, 6, 3], k = 3
Output: 1
Explanation: To bring elements 2, 1, 3 together, swap index 2 with 4 (0-based indexing), i.e. element arr[2] = 5 with arr[4] = 3 such that final array will be- arr[] = [2, 1, 3, 6, 5]

Input: arr[] = [2, 7, 9, 5, 8, 7, 4], k = 6
Output: 2
Explanation: To bring elements 2, 5, 4 together, swap index 0 with 2 (0-based indexing) and index 4 with 6 (0-based indexing) such that final array will be- arr[] = [9, 7, 2, 5, 4, 7, 8]

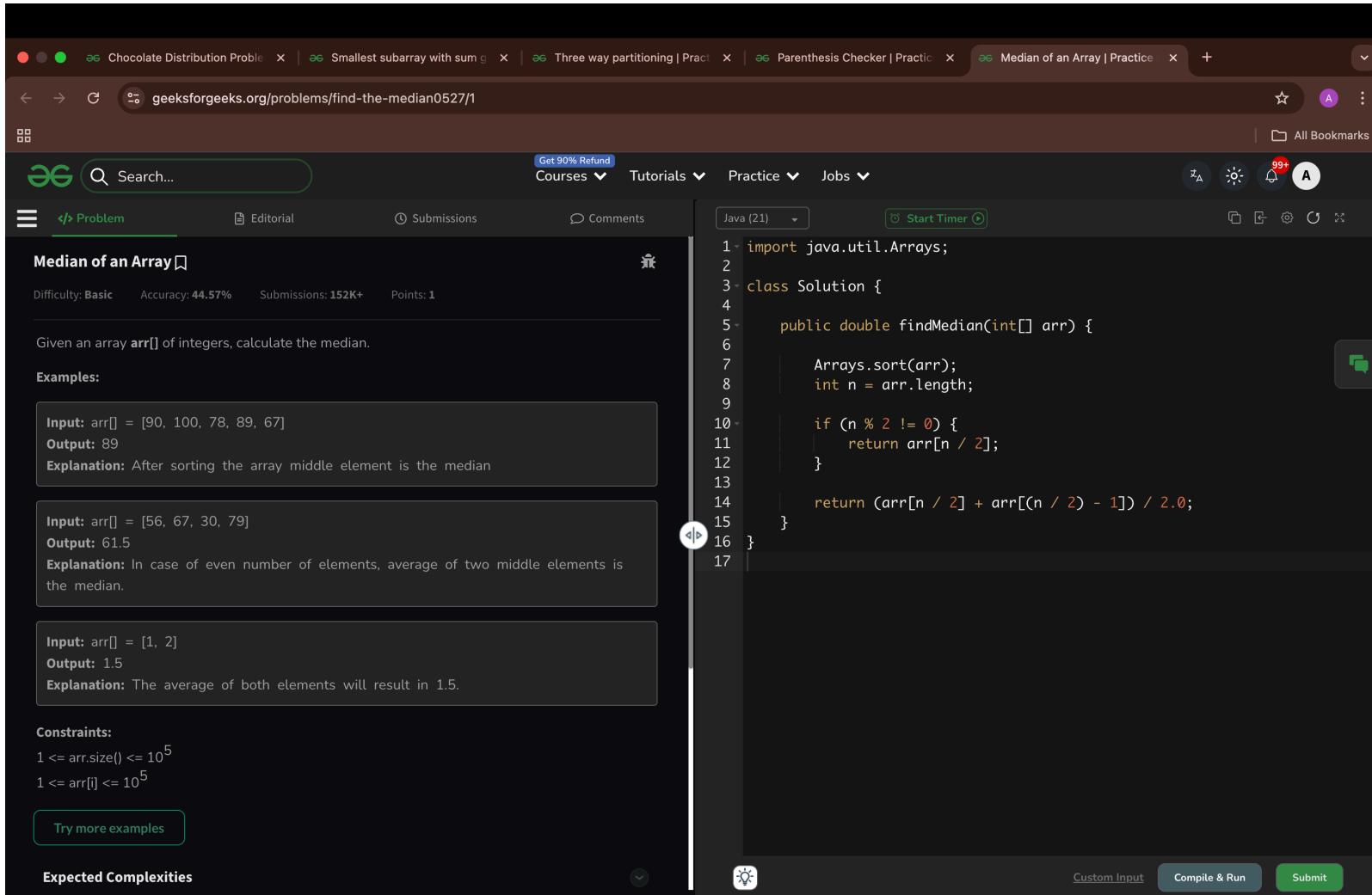
Input: arr[] = [2, 4, 5, 3, 6, 1, 8], k = 6
Output: 0

Constraints:

$1 \leq arr.size() \leq 10^6$
 $1 \leq arr[i] \leq 10^6$

Name - Anisha Raj
Roll no. - 2410031455
Section - 2CSE8

Java Experiment-2



The screenshot shows a web browser window with the GeeksforGeeks website open. The URL in the address bar is geeksforgeeks.org/problems/find-the-median0527/1. The page title is "Median of an Array".

The page content includes:

- Difficulty:** Basic **Accuracy:** 44.57% **Submissions:** 152K+ **Points:** 1
- Problem Statement:** Given an array `arr[]` of integers, calculate the median.
- Examples:**
 - Input:** `arr[] = [90, 100, 78, 89, 67]`
Output: 89
Explanation: After sorting the array middle element is the median
 - Input:** `arr[] = [56, 67, 30, 79]`
Output: 61.5
Explanation: In case of even number of elements, average of two middle elements is the median.
 - Input:** `arr[] = [1, 2]`
Output: 1.5
Explanation: The average of both elements will result in 1.5.
- Constraints:**
 - $1 \leq \text{arr.size()} \leq 10^5$
 - $1 \leq \text{arr}[i] \leq 10^5$
- Buttons:** Try more examples, Expected Complexities, Custom Input, Compile & Run, Submit

The right side of the screen shows a code editor with Java code for finding the median:

```
1 import java.util.Arrays;
2
3 class Solution {
4
5     public double findMedian(int[] arr) {
6
7         Arrays.sort(arr);
8         int n = arr.length;
9
10        if (n % 2 != 0) {
11            return arr[n / 2];
12        }
13
14        return (arr[n / 2] + arr[(n / 2) - 1]) / 2.0;
15    }
16
17 }
```

Name - Anisha Raj
Roll no. - 2410031455
Section - 2CSE8

Java Experiment-2

geeksforgeeks.org/problems/spirally-traversing-a-matrix-1587115621/1

Spirally traversing a matrix

Difficulty: Medium Accuracy: 35.2% Submissions: 343K+ Points: 4

You are given a rectangular matrix `mat[][]` of size $n \times m$, and your task is to return an array while **traversing** the matrix in **spiral** form.

Examples:

Input: `mat[][] = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]`
Output: [1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]

Explanation:

Example of matrix in spiral form

Matrix:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Output: 1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10

Input: `mat[][] = [[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12], [13, 14, 15, 16, 17, 18]]`
Output: [1, 2, 3, 4, 5, 6, 12, 18, 17, 16, 15, 14, 13, 7, 8, 9, 10, 11]
Explanation: Applying same technique as shown above.

Input: `mat[][] = [[32, 44, 27, 23], [54, 28, 50, 62]]`

```
import java.util.*;  
class Solution {  
    public ArrayList<Integer> spirallyTraverse(int[][] mat) {  
        ArrayList<Integer> result = new ArrayList<Integer>();  
        int n = mat.length;  
        int m = mat[0].length;  
        int top = 0, bottom = n - 1;  
        int left = 0, right = m - 1;  
  
        while (top <= bottom && left <= right) {  
            // Left → Right  
            for (int i = left; i <= right; i++) {  
                result.add(mat[top][i]);  
            }  
            top++;  
  
            // Top → Bottom  
            for (int i = top; i <= bottom; i++) {  
                result.add(mat[i][right]);  
            }  
            right--;  
  
            // Right → Left  
            if (top <= bottom) {  
                for (int i = right; i >= left; i--) {  
                    result.add(mat[bottom][i]);  
                }  
                bottom--;  
            }  
        }  
        return result;  
    }  
}
```

Custom Input Compile & Run Submit

Name - Anisha Raj
Roll no. - 2410031455
Section - 2CSE8

Java Experiment-2

The screenshot shows a Java code editor interface with the following details:

- Problem List:** The problem is titled "74. Search a 2D Matrix".
- Code:** The code is a Java class named "Solution" with a static method "searchMatrix". The code implements a binary search algorithm to search for a target value in a 2D matrix where each row is sorted in non-decreasing order and the first integer of each row is greater than the last integer of the previous row. The code uses 1D to 2D index conversion and handles the search range with "left" and "right" pointers.
- Input/Output:**
 - Example 1:** Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3. Output: true.
 - Example 2:** Input: matrix = [[1,3,5,7]], target = 1. Output: true.
- Statistics:** 17.7K views, 346 submissions, 239 online users.
- Editor Features:** Includes tabs for Description, Editorial, Solutions, Submissions, and a Premium section.

Name - Anisha Raj
Roll no. - 2410031455
Section - 2CSE8

Java Experiment-2

The screenshot shows a Java code editor on the GeeksforGeeks platform. The code is a solution for the problem 'Median in a row-wise sorted Matrix'. The code uses a binary search approach to find the median of a matrix where rows are sorted in ascending order. It includes input examples, explanations, and constraints.

```
1 class Solution {
2     int median(int[][] matrix) {
3         int r = matrix.length;
4         int c = matrix[0].length;
5
6         int low = 1;
7         int high = 2000; // according to constraints
8
9         while (low < high) {
10
11             int mid = (low + high) / 2;
12             int count = 0;
13
14             for (int i = 0; i < r; i++) {
15                 count += countSmallerThanOrEqual(matrix[i], mid);
16             }
17
18             if (count < (r * c + 1) / 2)
19                 low = mid + 1;
20             else
21                 high = mid;
22         }
23
24         return low;
25     }
26
27     int countSmallerThanOrEqual(int[] row, int target) {
28
29         int left = 0, right = row.length - 1;
30
31         while (left < right) {
32             int mid = (left + right) / 2;
33             if (row[mid] < target)
34                 left = mid + 1;
35             else
36                 right = mid;
37         }
38
39         if (row[left] < target)
40             return left + 1;
41         else
42             return left;
43     }
44 }
```

Median in a row-wise sorted Matrix

Difficulty: Medium Accuracy: 55.05% Submissions: 171K+ Points: 4

Given a **row-wise sorted** matrix `mat[][]` of size $n*m$, where the number of rows and columns is always **odd**. Return the **median** of the matrix.

Examples:

Input: `mat[][] = [[1, 3, 5], [2, 6, 9], [3, 6, 9]]`
Output: 5
Explanation: Sorting matrix elements gives us [1, 2, 3, 3, 5, 6, 6, 9, 9]. Hence, 5 is median.

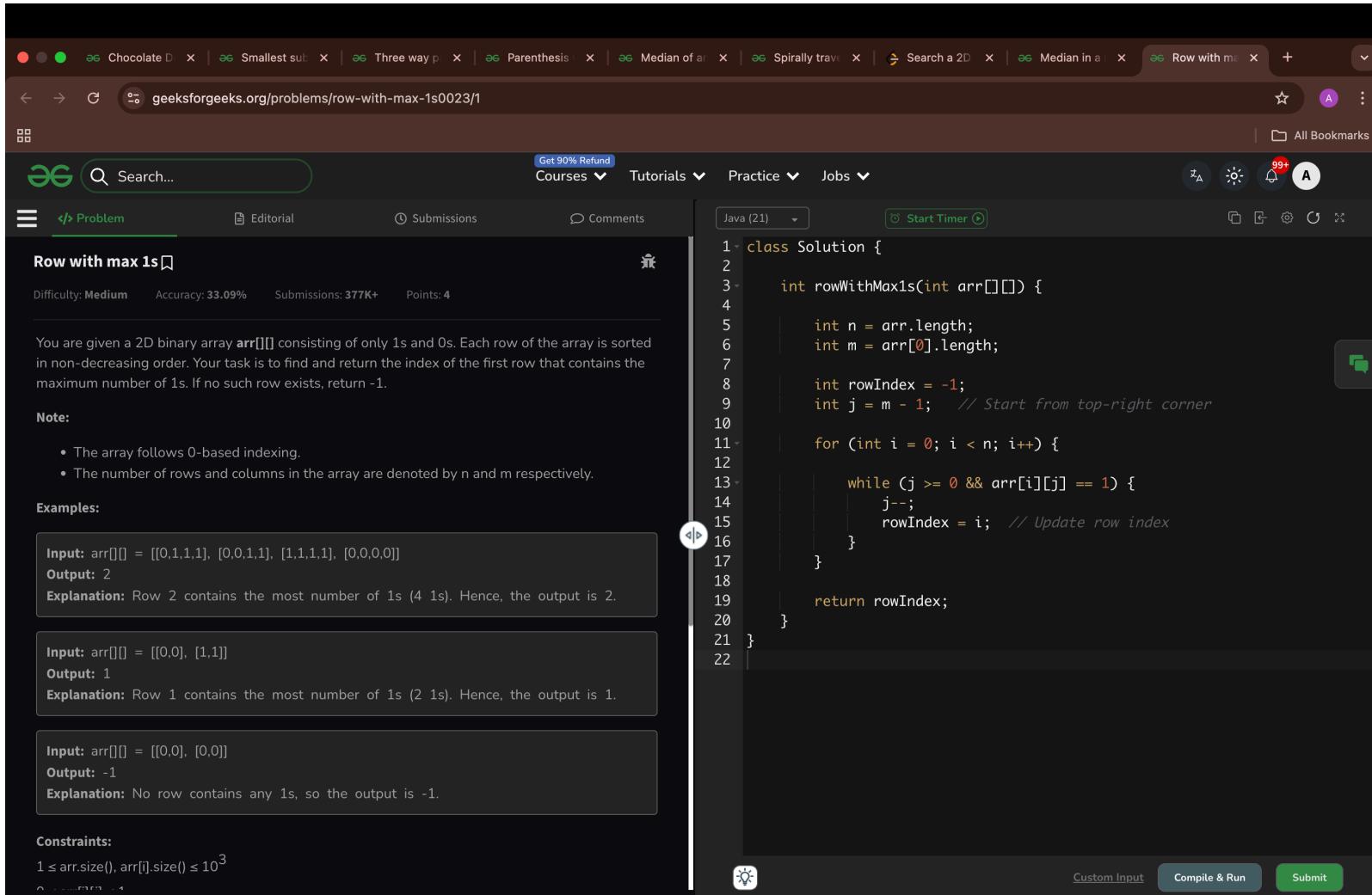
Input: `mat[][] = [[2, 4, 9], [3, 6, 7], [4, 7, 10]]`
Output: 6
Explanation: Sorting matrix elements gives us [2, 3, 4, 4, 6, 7, 7, 9, 10]. Hence, 6 is median.

Input: `mat = [[3], [4], [8]]`
Output: 4
Explanation: Sorting matrix elements gives us [3, 4, 8]. Hence, 4 is median.

Constraints:

Name - Anisha Raj
Roll no. - 2410031455
Section - 2CSE8

Java Experiment-2



The screenshot shows a Java programming environment on the GeeksforGeeks website. The problem is titled "Row with max 1s".

Problem Details:
Difficulty: Medium | Accuracy: 33.09% | Submissions: 377K+ | Points: 4

Description:
You are given a 2D binary array `arr[][]` consisting of only 1s and 0s. Each row of the array is sorted in non-decreasing order. Your task is to find and return the index of the first row that contains the maximum number of 1s. If no such row exists, return -1.

Note:

- The array follows 0-based indexing.
- The number of rows and columns in the array are denoted by n and m respectively.

Examples:

- Input:** `arr[][] = [[0,1,1,1], [0,0,1,1], [1,1,1,1], [0,0,0,0]]`
Output: 2
Explanation: Row 2 contains the most number of 1s (4 1s). Hence, the output is 2.
- Input:** `arr[][] = [[0,0], [1,1]]`
Output: 1
Explanation: Row 1 contains the most number of 1s (2 1s). Hence, the output is 1.
- Input:** `arr[][] = [[0,0], [0,0]]`
Output: -1
Explanation: No row contains any 1s, so the output is -1.

Constraints:
 $1 \leq \text{arr.size(), arr[i].size()} \leq 10^3$

Code Editor:

```
Java (21) Start Timer
1 class Solution {
2     int rowWithMax1s(int arr[][]) {
3         int n = arr.length;
4         int m = arr[0].length;
5
6         int rowIndex = -1;
7         int j = m - 1; // Start from top-right corner
8
9         for (int i = 0; i < n; i++) {
10
11             while (j >= 0 && arr[i][j] == 1) {
12                 j--;
13                 rowIndex = i; // Update row index
14             }
15         }
16
17         return rowIndex;
18     }
19 }
20 }
21 }
22 }
```

Custom Input | Compile & Run | Submit