

Experiment List for Programming Ability and Logic Building-2

Name : Ikshu Goel

Batch : 2CSE5

Roll Number : 2410030774

Week : 19/1/26 to 25/1/26

Lecture : 1 & 2

EXPERIMENT 1

Given an array $\text{arr}[]$ of positive integers, where each value represents the number of chocolates in a packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets among m students such that -

- i. Each student gets exactly one packet.

ii. The difference between maximum number of chocolates given to a student and minimum number of chocolates given to a student is minimum and return that minimum possible difference.

Examples:

Input: arr = [3, 4, 1, 9, 56, 7, 9, 12], m = 5

Output: 6

Explanation: The minimum difference between maximum chocolates and minimum chocolates is $9 - 3 = 6$ by choosing following m packets :[3, 4, 9, 7, 9].

Constraints:

$1 \leq m \leq \text{arr.size} \leq 105$

$1 \leq \text{arr}[i] \leq 109$

The screenshot shows a Java code editor on the GeeksforGeeks website. The code is a solution to the Chocolate Distribution Problem, which involves finding the minimum difference between the maximum and minimum values in a sorted array of size n, given that m packets are chosen from the array. The code uses a linear search approach to find the minimum difference.

```
1- import java.util.*;
2-
3- class Solution {
4-     public long findMinDiff(ArrayList<Integer> arr, int m) {
5-
6-         int n = arr.size();
7-         if (m > n)
8-             return -1;
9-
10-        Collections.sort(arr);
11-
12-        long minDiff = Long.MAX_VALUE;
13-
14-        for (int i = 0; i <= n - m; i++) {
15-            long diff = arr.get(i + m - 1) - arr.get(i);
16-
17-            if (diff < minDiff) {
18-                minDiff = diff;
19-            }
20-
21-        }
22-
23-        return minDiff;
24-    }
25-
26- }
```

The code editor interface includes tabs for Output Window, Compilation Results, and Custom Input. The Output Window shows the result for Case 1: Input [3, 4, 1, 9, 56, 7, 9, 12] and Output 6. The Compilation Results tab shows 'Compilation Completed'.

EXPERIMENT 2

Given a number x and an array of integers arr , find the smallest subarray with sum greater than the given value. If such a subarray does not exist return 0 in that case.

Examples:

Input: $x = 51$, $\text{arr}[] = [1, 4, 45, 6, 0, 19]$

Output: 3

Explanation: Minimum length subarray is $[4, 45, 6]$
Input: $x = 100$, $\text{arr}[] = [1, 10, 5, 2, 7]$

Output: 0

Explanation: No subarray exist

Constraints:

$1 \leq \text{arr.size}, x \leq 105$

$0 \leq \text{arr}[] \leq 104$

The screenshot shows a Java code editor interface on the website geeksforgeeks.org. The code is a Java program named Solution.java, which implements a function to find the smallest subarray with a sum greater than or equal to a given value x. The code uses a two-pointer approach with a sliding window to efficiently solve the problem.

```
1- class Solution {  
2-     public static int smallestSubWithSum(int x, int[] arr) {  
3-         int n = arr.length;  
4-         int minLen = n + 1;  
5-         int sum = 0, start = 0;  
6-  
7-         for (int end = 0; end < n; end++) {  
8-             sum += arr[end];  
9-  
10-            while (sum > x) {  
11-                minLen = Math.min(minLen, end - start + 1);  
12-                sum -= arr[start++];  
13-            }  
14-        }  
15-        return (minLen == n + 1) ? 0 : minLen;  
16-    }  
17- }  
18- }
```

The code editor includes a toolbar at the top with options like 'Courses', 'Tutorials', 'Practice', and 'Jobs'. Below the code, there are buttons for 'Custom Input', 'Compile & Run', and 'Submit'. On the left side, there is an 'Output Window' and a 'Compilation Results' section. The 'Compilation Completed' section shows the input values x=51 and arr=[1, 4, 45, 6, 0, 19] and the output 3, which matches the expected output.

EXPERIMENT 3

Given an array and a range a, b . The task is to partition the array around the range such that the array is divided into three parts.

- 1) All elements smaller than a come first.
- 2) All elements in range a to b come next.
- 3) All elements greater than b appear in the end.

The individual elements of three sets can appear in any order. You are required to return the modified array.

Note: The generated output is true if you modify the given array successfully. Otherwise false.

Geeky Challenge: Solve this problem in $O(n)$ time complexity.

Examples:

Input: $\text{arr[]} = [1, 2, 3, 3, 4]$, $a = 1$, $b = 2$

Output: true

Explanation: One possible arrangement is: $\{1, 2, 3, 3, 4\}$. If you return a valid arrangement, output will be true.

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

Output: true

Explanation: One possible arrangement is: $\{1, 3, 2, 1, 4, 6\}$. If you return a valid arrangement, output will be true.

Constraints:

$1 \leq \text{arr.size}() \leq 106$

$1 \leq \text{array}[i], a, b \leq 109$

The screenshot shows a Java code editor on the GeeksforGeeks website. The code is a solution for the 'Three way partitioning' problem. It defines a class 'Solution' with a static method 'threeWayPartition'. The method takes an integer array 'arr', an integer 'a', and an integer 'b'. It initializes 'low' to 0, 'mid' to 0, and 'high' to the length of the array minus 1. A while loop runs as long as 'mid' is less than or equal to 'high'. Inside the loop, it checks if the element at index 'mid' is less than 'a'. If so, it swaps it with the element at 'low' and increments both 'low' and 'mid'. If the element is greater than 'b', it swaps it with the element at 'high' and decrements 'high'. Otherwise, it increments 'mid'. After the loop, a swap function 'swap' is defined, which swaps elements at indices 'i' and 'j' in the array 'arr'. The code editor interface includes tabs for 'Java (21)', 'Start Timer', and buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

```
1+ class Solution {  
2+     public void threeWayPartition(int[] arr, int a, int b) {  
3+         int low = 0, mid = 0, high = arr.length - 1;  
4+  
5+         while (mid <= high) {  
6+             if (arr[mid] < a) {  
7+                 swap(arr, low++, mid++);  
8+             } else if (arr[mid] > b) {  
9+                 swap(arr, mid, high--);  
10+            } else {  
11+                mid++;  
12+            }  
13+        }  
14+    }  
15+  
16+    void swap(int[] arr, int i, int j) {  
17+        int temp = arr[i];  
18+        arr[i] = arr[j];  
19+        arr[j] = temp;  
20+    }  
21+ }
```

EXPERIMENT 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 $\text{arr}[i]$, $\text{arr}[j]$. 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:

$1 \leq \text{arr.size()} \leq 10^6$

$1 \leq \text{arr}[i] \leq 10^6$

$1 \leq k \leq 10^6$

The screenshot shows a Java code editor interface on the website geeksforgeeks.org. The code is a Java program named Solution that contains a method minSwap. The code logic involves counting elements less than or equal to k, then calculating the number of swaps needed to move elements greater than k to their correct positions. It uses two pointers, i and j, to iterate through the array, adjusting them based on the count of elements less than or equal to k.

```
1+ class Solution {  
2-     int minSwap(int[] arr, int k) {  
3-         int n = arr.length;  
4-  
5-         int count = 0;  
6-         for (int x : arr)  
7-             if (x <= k) count++;  
8-  
9-         int bad = 0;  
10-        for (int i = 0, j = count; j < n; i++, j++) {  
11-            if (arr[i] > k) bad--;  
12-            if (arr[j] > k) bad++;  
13-            ans = Math.min(ans, bad);  
14-        }  
15-    }  
16-  
17-    int ans = bad;  
18-    for (int i = 0, j = count; j < n; i++, j++) {  
19-        if (arr[i] > k) bad--;  
20-        if (arr[j] > k) bad++;  
21-        ans = Math.min(ans, bad);  
22-    }  
23-  
return ans;  
}
```

EXPERIMENT 5

Given an array `arr[]` of positive integers. Return true if all the array elements are palindrome otherwise, return false.

Examples:

Input: arr[] = [111, 222, 333, 444, 555]

Output: true

Explanation:

arr[0] = 111, which is a palindrome number.

arr[1] = 222, which is a palindrome number.

arr[2] = 333, which is a palindrome number.

arr[3] = 444, which is a palindrome number.

arr[4] = 555, which is a palindrome number.

As all numbers are palindrome so This will return true.
Input: arr[] = [121, 131, 20]

Output: false

Explanation: 20 is not a palindrome hence the output is false.

Expected Time Complexity: O(nlogn)

Expected Space Complexity: O(1)

Constraints:

1 <=arr.size<= 20

1 <=arr[i]<= 105

The screenshot shows a Java code editor interface on the website geeksforgeeks.org. The code is written in Java and defines a class named Solution. It contains two methods: isPalindromicArray and isPalindrome. The isPalindromicArray method iterates through an array of integers, calling the isPalindrome method on each element. The isPalindrome method checks if a single integer is a palindrome by reversing it and comparing it to the original number. The code uses a while loop to reverse the digits of the number.

```
1+ class Solution {  
2+     boolean isPalindromicArray(int[] arr) {  
3+         for (int num : arr) {  
4+             if (!isPalindrome(num)) {  
5+                 return false;  
6+             }  
7+         }  
8+         return true;  
9+     }  
10+  
11+     boolean isPalindrome(int num) {  
12+         int rev = 0;  
13+         int temp = num;  
14+  
15+         while (temp > 0) {  
16+             rev = rev * 10 + temp % 10;  
17+             temp /= 10;  
18+         }  
19+         return rev == num;  
20+     }  
21+ }  
22+ }  
23+ }
```

EXPERIMENT 6

Given an array $\text{arr}[]$ 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:

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

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

The screenshot shows a browser window on the GeeksforGeeks website. The URL is [geeksforgeeks.org/problems/find-the-median0527/1](https://www.geeksforgeeks.org/problems/find-the-median0527/1). The page title is "Median of an Array | Practice". The top navigation bar includes links for Courses, Tutorials, Practice, and Jobs. A search bar is present. The main content area has tabs for "Problem", "Editorial", "Submissions", and "Comments". The "Output Window" tab is active, showing the "Compilation Results" section which displays "Compilation Completed". Below it, there's a "Case 1" section with an "Input" field containing "90 100 78 89 67" and a "Your Output" field showing "89". The "Expected Output" field also shows "89". On the right side, the Java code for finding the median is displayed:

```
1 import
2
3 class Si
4 public Jobs Updates
5
6 int n = arr.length;
7
8 if (n % 2 == 1)
9     return arr[n / 2];
10 else
11     return (arr[n / 2] + arr[n / 2 - 1]) / 2.0;
12
13 }
```

Below the code, there are buttons for "Custom Input", "Compile & Run", and "Submit".

EXPERIMENT 7

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, 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]]`

Output: `[32, 44, 27, 23, 62, 50, 28, 54]`

Explanation: Applying same technique as shown above, output will be `[32, 44, 27, 23, 62, 50, 28, 54]`.

Constraints:

$1 \leq n, m \leq 1000$

$0 \leq \text{mat}[i][j] \leq 100$

The screenshot shows a Java code editor on the geeksforgeeks.org website. The code is a solution for spirally traversing a matrix. The code uses four pointers: top, bottom, left, and right, to traverse the matrix in a spiral order. It handles cases where the matrix is empty or has only one row or column. The code is written in Java and uses ArrayList<Integer> to store the result.

```
1+ import java.util.*;
2
3+ class Solution {
4+     ArrayList<Integer> spirallyTraverse(int[][] matrix) {
5+         ArrayList<Integer> res = new ArrayList<Integer>();
6+
7+         int r = matrix.length;
8+         int c = matrix[0].length;
9+
10+        int top = 0, bottom = r - 1, left = 0, right = c - 1;
11+
12+        while (top <= bottom && left <= right) {
13+            for (int i = left; i <= right; i++) {
14+                res.add(matrix[top][i]);
15+            }
16+            top++;
17+            for (int i = top; i <= bottom; i++) {
18+                res.add(matrix[i][right]);
19+            }
20+            right--;
21+
22+            if (top <= bottom) {
23+                for (int i = right; i >= left; i--) {
24+                    res.add(matrix[bottom][i]);
25+                }
26+                bottom--;
27+
28+                if (left <= right) {
29+                    for (int i = bottom; i >= top; i--) {
30+                        res.add(matrix[i][left]);
31+                    }
32+                    left++;
33+                }
34+            }
35+        }
36+
37+        return res;
38+    }
39+}
```

EXPERIMENT 8

You are given an $m \times n$ integer matrix matrix with the following two properties:

- . Each row is sorted in non-decreasing order.
- . The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true *if target is in* matrix *or* false *otherwise*.

You must write a solution in $O(\log(m * n))$ time complexity.

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],[10,11,16,20],[23,30,34,60]], target = 13

Output: false

The screenshot shows a LeetCode problem page for "Search a 2D Matrix".

Description: You are given an $m \times n$ integer matrix `matrix` with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

Given an integer `target`, return `true` if `target` is in `matrix` or `false` otherwise.

You must write a solution in $O(\log(m * n))$ time complexity.

Example 1:

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
Output: true

Example 2:

Code:

```

1 class Solution {
2     public boolean searchMatrix(int[][] matrix, int target) {
3         int n = matrix.length;
4         int m = matrix[0].length;
5
6         int low = 0, high = n * m - 1;
7
8         while (low <= high) {
9             int mid = (low + high) / 2;
10            int val = matrix[mid / m][mid % m];
11
12            if (val == target) return true;
13            else if (val < target) low = mid + 1;
14            else high = mid - 1;
15        }
16        return false;
17    }
18 }
```

Testcase: Accepted Runtime: 0 ms

Case 1 **Case 2**

Input:

```

matrix =
[[1,3,5,7],[10,11,16,20],[23,30,34,60]]
```

target =

17.7K 346 185 Online

EXPERIMENT 9

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:

$1 \leq n, m \leq 400$

$1 \leq \text{mat}[i][j] \leq 2000$

The screenshot shows a Java code editor on the GeeksforGeeks website. The code is a Java program named `Solution` that takes a 2D integer array `matrix` as input and returns its median. The code uses a `ArrayList` to store all elements of the matrix and then sorts it to find the middle element.

```
1+ import java.util.*;
2+
3+ class Solution {
4+     int median(int[][] matrix) {
5+         int r = matrix.length;
6+         int c = matrix[0].length;
7+
8+         ArrayList<Integer> list = new ArrayList<>();
9+
10+        for (int i = 0; i < r; i++) {
11+            for (int j = 0; j < c; j++) {
12+                list.add(matrix[i][j]);
13+            }
14+        }
15+
16+        Collections.sort(list);
17+        return list.get(list.size() / 2);
18+    }
19+
20+ }
```

The code editor interface includes tabs for "Custom Input" and "Compile & Run". The status bar at the bottom shows "Custom Input" and "Compile & Run" buttons.

EXPERIMENT 10

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.

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.

1.Input: arr[][] = [[0,0], [0,0]]

Output: -1

Explanation: No row contains any 1s, so the output is -1.

Constraints:

$1 \leq \text{arr.size}(), \text{arr}[i].size() \leq 103$

$0 \leq \text{arr}[i][j] \leq 1$

To exit full screen, press and hold esc

geeksforgeeks.org/problems/row-with-max-1s0023/1

Courses Tutorials Practice Jobs

Output Window

Compilation Results Custom Input

Compilation Completed

Case 1

Input: arr[] = [[0,1,1,1], [0,0,1,1], [1,1,1,1], [0,0,0,0]]

Your Output: 2

Expected Output: 2

```
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 maxRow = -1;
7-         int maxCount = 0;
8-
9-         for (int i = 0; i < n; i++) {
10-             int count = 0;
11-             for (int j = 0; j < m; j++) {
12-                 if (arr[i][j] == 1) count++;
13-             }
14-             if (count > maxCount) {
15-                 maxCount = count;
16-                 maxRow = i;
17-             }
18-         }
19-         return maxRow;
20-     }
21- }
22-
23-
```

Custom Input Compile & Run Submit