Row with max 1s

Difficulty: **Medium** Accuracy: **33.09%** Submissions: **376K+** Points: **4**

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.

Java (21) Start Timer

```java
class Solution {
    int rowWithMax1s(int[][] arr) {
        int n = arr.length, m = arr[0].length;
        int row = -1, j = m - 1;

        for (int i = 0; i < n; i++) {
            while (j >= 0 && arr[i][j] == 1) {
                j--;
                row = i;
            }
        }
        return row;
    }
}
```

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.

Java (21) Start Timer

```java
class Solution {
    public int median(int[][] mat) {
        int n = mat.length, m = mat[0].length;
        int low = 1, high = 2000;

        while (low <= high) {
            int mid = (low + high) / 2;
            int count = 0;

            for (int i = 0; i < n; i++) {
                int l = 0, r = m;
                while (l < r) {
                    int md = (l + r) / 2;
                    if (mat[i][md] <= mid) l = md + 1;
                    else r = md;
                }
                count += l;
            }

            if (count <= (n * m) / 2) low = mid + 1;
            else high = mid - 1;
        }
        return low;
    }
}
```

Problem List | Submit | Premium

Description | Editorial | Solutions | Submissions

74. Search a 2D Matrix

Solved

Medium | Topics | Companies

You are given an `m x 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:

Code

Java | Auto

```java
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int m = matrix.length, n = matrix[0].length;
        int l = 0, r = m * n - 1;

        while (l <= r) {
            int mid = l + (r - l) / 2;
            int val = matrix[mid / n][mid % n];

            if (val == target) return true;
            if (val < target) l = mid + 1;
            else r = mid - 1;
        }
        return false;
    }
}
```

Saved

Ln 1, Col 1

Testcase | Test Result

Problem | Editorial | Submissions | Comments

Median of an Array

Difficulty: **Basic** Accuracy: **44.57%** Submissions: **151K+** Points: **1**

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:

$1 <= arr.size() <= 10^5$

$1 <= arr[i] <= 10^5$

Java (21) Start Timer

```java
class Solution {
    public double findMedian(int[] arr) {
        Arrays.sort(arr);
        int n = arr.length;

        // Odd number of elements
        if (n % 2 != 0) {
            return arr[n / 2];
        }

        // Even number of elements
        return (arr[n / 2 - 1] + arr[n / 2]) / 2.0;
    }
}
```

Problem | Editorial | Submissions | Comments

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:

Java (21) Start Timer

```java
        int good = 0;
        for (int num : arr) {
            if (num <= k) {
                good++;
            }
        }

        // If no good elements or all are good
        if (good == 0 || good == n) {
            return 0;
        }

        // Count bad elements in the first window of size 'good'
        int bad = 0;
        for (int i = 0; i < good; i++) {
            if (arr[i] > k) {
                bad++;
            }
        }

        int minSwaps = bad;

        // Slide the window
        for (int i = 0, j = good; j < n; i++, j++) {
            if (arr[i] > k) bad--;
            if (arr[j] > k) bad++;
            minSwaps = Math.min(minSwaps, bad);
        }

        return minSwaps;
    }
}
```

Custom Input | Compile & Run | Submit

Three way partitioning

Difficulty: **Easy** Accuracy: **41.58%** Submissions: **187K+** Points: **2** Average Time: **20m**

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: 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: 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.

Java (21) Start Timer

```java
class Solution {
    // Function to partition the array around the range
    // such that array is divided into three parts.
    public void threeWayPartition(int arr[], int a, int b) {
        int low = 0, mid = 0;
        int high = arr.length - 1;

        while (mid <= high) {
            if (arr[mid] < a) {
                // Move element smaller than a to left
                int temp = arr[low];
                arr[low] = arr[mid];
                arr[mid] = temp;
                low++;
                mid++;
            }
            else if (arr[mid] > b) {
                // Move element greater than b to right
                int temp = arr[mid];
                arr[mid] = arr[high];
                arr[high] = temp;
                high--;
            }
            else {
                // Element lies between a and b
                mid++;
            }
        }
    }
}
```

Problem | Editorial | Submissions | Comments

Smallest subarray with sum greater than x

Difficulty: **Easy** Accuracy: **37.07%** Submissions: **154K+** Points: **2** Average Time: **20m**

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 do not exist return 0 in that case.

Examples:

Input: x = 51, arr[] = [1, 4, 45, 6, 0, 19]
Output: 3
Explanation: Minimum length subarray is [4, 45, 6]

Input: x = 100, arr[] = [1, 10, 5, 2, 7]
Output: 0
Explanation: No subarray exist

Constraints:

$1 \leq arr.size, x \leq 10^5$

$0 \leq arr[] \leq 10^4$

Try more examples

Expected Complexities

Java (21) Start Timer

```java
class Solution {
    public static int smallestSubWithSum(int x, int[] arr) {
        int n = arr.length;
        int start = 0, sum = 0;
        int minLen = Integer.MAX_VALUE;

        for (int end = 0; end < n; end++) {
            sum += arr[end];

            // Try to shrink the window while sum > x
            while (sum > x) {
                minLen = Math.min(minLen, end - start + 1);
                sum -= arr[start];
                start++;
            }
        }

        // If no subarray found
        return (minLen == Integer.MAX_VALUE) ? 0 : minLen;
    }
}
```

Problem | Editorial | Submissions | Comments

Given an array **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].

Input: arr = [7, 3, 2, 4, 9, 12, 56], m = 3
Output: 2
Explanation: The minimum difference between maximum chocolates and minimum chocolates is 4 - 2 = 2 by choosing following m packets :[3, 2, 4].

Input: arr = [3, 4, 1, 9, 56], m = 5
Output: 55
Explanation: With 5 packets for 5 students, each student will receive one packet, so the difference is 56 - 1 = 55.

Constraints:

$1 \le m <= arr.size \le 10^5$

$1 \le arr[i] \le 10^9$

Java (21) | Start Timer

```java
// User function Template for Java

class Solution {
    public int findMinDiff(ArrayList<Integer> arr, int m) {
        int n = arr.size();

        // Edge case
        if (m == 0 || n < m) {
            return 0;
        }

        // Sort the packets
        Collections.sort(arr);

        int minDiff = Integer.MAX_VALUE;

        // Sliding window of size m
        for (int i = 0; i + m - 1 < n; i++) {
            int diff = arr.get(i + m - 1) - arr.get(i);
            minDiff = Math.min(minDiff, diff);
        }

        return minDiff;
    }
}
```

Custom Input | Compile & Run | Submit