

# JAVA PRACTICE SHEET (02/02/2026 – 07/02/2026)

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:**

1. **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.

## Solution-

The screenshot displays a Java IDE interface. On the left, the 'Output Window' shows 'Problem Solved Successfully' with test cases passed (1111/1111), attempts (1/1), accuracy (100%), points scored (2/2), and time taken (0.22). The main editor shows a Java class 'Solution' with a method 'threeWayPartition' that implements the partitioning logic using a while loop and conditional swaps. The code is as follows:

```
1 class Solution {
2     // Function to partition the array around the range such
3     // that array is divided into three parts.
4     public void threeWayPartition(int arr[], int a, int b) {
5         // code here
6         int low = 0;
7         int mid = 0;
8         int high = arr.length - 1;
9
10        while (mid <= high) {
11
12            if (arr[mid] < a) {
13                int temp = arr[mid];
14                arr[mid] = arr[low];
15                arr[low] = temp;
16
17                low++;
18                mid++;
19            }
20            else if (arr[mid] > b) {
21                int temp = arr[mid];
22                arr[mid] = arr[high];
23                arr[high] = temp;
24
25                high--;
26            }
27            else {
28                mid++;
29            }
30        }
31    }
32 }
```

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

2.

## Solution-

The screenshot displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Compilation Results' for a submission by 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with 1112/1112 test cases passed, 1/1 attempts correct, 100% accuracy, 4/4 points scored, and a time taken of 0.45 seconds. Below this, 'Solve Next' buttons for 'Rearrange Array Alternately', 'Count Number', and 'Subarray Inversions' are visible. The main editor on the right shows a Java solution for 'Java (21)'. The code defines a 'Solution' class with a 'minSwap' method that counts the minimum swaps required to group elements less than or equal to 'k' together. The method uses two passes: one to count 'good' elements (≤ k) and another to count 'bad' elements (> k), then calculates the minimum swaps as the maximum of these counts.

```
1 // User function Template for Java
2
3 class Solution {
4     // Function for finding maximum and value pair
5     int minSwap(int[] arr, int k) {
6         // Complete the function
7         int n = arr.length;
8
9         int good = 0;
10        for (int i = 0; i < n; i++) {
11            if (arr[i] <= k) good++;
12        }
13
14        if (good == 0 || good == 1) return 0;
15
16        int bad = 0;
17        for (int i = 0; i < good; i++) {
18            if (arr[i] > k) bad++;
19        }
20
21        int ans = bad;
22
23        int i = 0, j = good;
24        while (j < n) {
25
26            if (arr[i] > k) bad--;
27            if (arr[j] > k) bad++;
28
29            ans = Math.min(ans, bad);
30
31            i++;
32            j++;
33        }
34
35        return ans;
36    }
37 }
```

3. You are given an  $m \times n$  integer 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:**

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 = 13

**Output:** false

**Solution-**

Problem List

Submit

Premium

Description

Accepted

Editorial

Solutions

Submissions

All Submissions

Accepted 133 / 133 testcases passed

harshittiwari07 submitted at Feb 08, 2026 21:44

Editorial

Solution

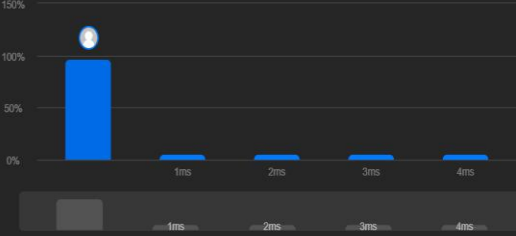
Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

43.84 MB | Beats 78.40%



Code

Java

Auto

```
1 class Solution {
2     public boolean searchMatrix(int[][] matrix, int target) {
3         int m = matrix.length;
4         int n = matrix[0].length;
5
6         int low = 0;
7
8     }
```

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Input

matrix =

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

target =

3

Output

true

Expected

true

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

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

## Solution-

The screenshot displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Problem Solved Successfully' with a green checkmark. Below this, statistics are listed: 'Test Cases Passed: 1111/1111', 'Attempts: Correct / Total: 1/1', 'Accuracy: 100%', 'Points Scored: 4/4', and 'Time Taken: 0.81'. A 'Solve Next' section offers links to 'Max sum in the configuration', 'Boolean Matrix', and 'Row with Minimum 1s'. At the bottom left, a banner promotes 'Build 21 Projects in 21 Days'. The main area on the right shows a Java code editor with a solution for the problem. The code uses a binary search approach to find the row with the maximum number of 1s. It initializes variables for the number of rows (*n*), columns (*m*), the maximum number of ones found (*maxOnes*), and the index of the row with the maximum ones (*ansRow*). A loop iterates over each row, and within each row, a binary search is performed to find the first occurrence of 1. The row index is updated if a row with more ones is found. The code is as follows:

```
1 // User function Template for Java
2
3 class Solution {
4     public int rowWithMaxIs(int arr[][]){
5         // code here
6         int n = arr.length;
7         int m = arr[0].length;
8
9         int maxOnes = 0;
10        int ansRow = -1;
11
12        for (int i = 0; i < n; i++) {
13
14            int low = 0, high = m - 1;
15            int firstOne = -1;
16
17            while (low <= high) {
18                int mid = low + (high - low) / 2;
19
20                if (arr[i][mid] == 1) {
21                    firstOne = mid;
22                    high = mid - 1;
23                } else {
24                    low = mid + 1;
25                }
26            }
27
28            if (firstOne != -1) {
29                int ones = m - firstOne;
30
31                if (ones > maxOnes) {
32                    maxOnes = ones;
33                    ansRow = i;
34                }
35            }
36        }
37
38        return ansRow;
39    }
40 }
```