LABSHEET - 4

CS F211 - Data Structures and Algorithms
Academic Year: 2025-2026 (2nd Semester)

General Instructions for this Labsheet:

- Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.
- Question 3 should be done using sorting only.
- Questions 2, 5, 9 can be done using any suitable data structure.
- Questions 1, 4, 6, 7, 8, 10 should be done using binary search. Using linear search is not allowed.

Question 1: Search in a sorted 2D matrix

Problem Description:

Given an M x N matrix of distinct integers where:

1. Each row is sorted in ascending order from left to right.
2. Each column is sorted in ascending order from top to bottom.

Write an algorithm to search for a key value 'K' within this matrix.
If the element exists, return its row and column indices (i, j). If the element is not present, print -1.

Solve this using an efficient algorithm in O(M + N) time, or use binary search.
***Hint: Start iterating from the top-right corner of the matrix for efficient search.

Input Format:

Line 1: Two integers M, N (Rows, Columns)
Next M lines: N integers each, representing the matrix rows
Last Line: One integer (Value of key K)

Output Format:

Two space-separated integers representing the row index and column index **(0-based indexing)**. If not found, output -1.

Sample Test Case 1:

Input:

```
2 3
1 3 5
2 4 6
7
```

Output:

```
-1
```

Sample Test Case 2:

Input:

```
4 4
10 20 30 40
15 25 35 45
27 29 37 48
32 33 39 50
29
```

Output:

```
2 1
```

Note: Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.

Question 2: Zig-Zag Group Sorting

Problem Description:

Given an array of N(≥ 2) distinct integers, you must process and sort the elements in groups of increasing size, alternating between the front and the back of the available elements.

PROCESS:

1. Start with group size K = 1.
2. Take K elements from the front of the current available elements.
3. Sort these K elements in ascending order in their current positions.
4. Increment K by 1 (K = 2).
5. Take K elements from the back of the current available elements.
6. Sort these K elements in ascending order in their current positions.
7. Increment K by 1 (K = 3).
8. Repeat the process (alternating Front and Back) until all elements have been assigned to a group.
9. If the number of remaining elements is less than the current K, the final group will simply consist of all leftover elements.

Input Format:

Line 1: Integer N (Size of the array, N ≥ 2)
Line 2: N integers, representing the array elements

Output Format:

Final processed array of elements

Sample Test Case 1:

Input:
10
10 9 8 7 6 5 4 3 2 1

Output:
10 7 8 9 3 4 5 6 1 2

Sample Test Case 2:

Input:
8
5 3 2 9 6 1 7 4

Output:
5 2 3 9 1 6 4 7
===> Underlined elements are in the same group.

Note: Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.

Question 3: Same Element beyond K

Problem Description:

Given an array of N elements, find if there exists two equal elements such that the distance* between them is greater than K(> 0).

*Distance between two indices is defined as the absolute difference.
Eg: If i = 1, j = 4, distance between i and j = |1 - 4| = 3.
Solve the question using sorting. O(n^2) solution will **NOT** be accepted.

Input Format:

Line 1: Two integers N and K (Size of the array, integer K(> 0))
Line 2: N integers, representing the array elements

Output Format:

If such a pair of elements exists, output any one possible pair of indices, else output -1.

Sample Test Case 1:

Input:
6 2
1 2 3 1 4 5

Output:
0 3

Explanation: Possible pair -> (0, 3)

Sample Test Case 2:

Input:
5 3
1 4 1 3 4

Output:
-1

Explanation: No such possible pair exists

Note: Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.

Question 4: where is gROOT?

Problem Description:

Peter and Rocket are finding Groot. Reportedly, he has been hiding on a planet numbered X, where X is the largest integer whose product with itself is less than or equal to K. Peter, unaware of better searching algorithms, plans to find him through all the planets sequentially starting from Planet 1. But Rocket wants to find him before Peter by using Binary Search. Help him do so by giving the planet number on which Groot is hiding.

Input Format:

Line 1: Integer K ($0 < K \leq 1{,}000{,}000$)

Output Format:

The planet X

Sample Test Case 1:

Input:
148000

Output:
384

Sample Test Case 2:

Input:
331776

Output:
576

Note: Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.

Question 5: Bitwise Strength Sort

Problem Description:

The "Strength" S of a number is the total count of 1s in its binary form. For example, the number 6 in binary is 110, so S(6) = 2. Your task is to sort a list of N unique non-negative integers based on these rules:

1. The number with higher Strength will come first.
2. If two numbers have the same Strength, the larger numerical value must come first.

Input Format

Line 1: An integer N (the number of elements).

Line 2: N space-separated unique integers. (Integers lie in the range [1,255])

Output Format

A single line containing the sorted integers separated by spaces.

Sample Test Case 1:

Input:

4

3 5 7 8

Output:

7 5 3 8

Explanation: S(7) = 3; S(5) = S(3) = 2; S(8) = 1

Sample Test Case 2:

Input:

5

10 2 8 4 7

Output:

7 10 8 4 2

Explanation: S(7) = 3; S(10) = 2; S(8) = S(4) = S(2) = 1

Note: Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.

Question 6: Finding the Middle Score

Problem Description:

You are given a table (matrix) of scores with R rows and C columns. In this table, each row is already sorted from lowest to highest. Your goal is to find the median score of the entire table. The median is the number that would be in the exact middle if you took every single number from the table and put them into one long sorted list.

Note:

1. All scores are positive integers.
2. You are not allowed to create a new list or array to store all the numbers. You must find the median by searching the table using a binary search approach.
3. The total number of rows and columns is odd so R*C will always be an odd number, so there is only one middle value.

Input Format:
First Line: Two integers R (rows) and C (columns).

Next R Lines: Each line has C sorted positive integers.

Output Format

A single integer (the median).

Sample Test Case 1:
Input:
3 3
1 10 20
2 15 25
3 30 40

Output:
15
Explanation: If you list all scores in order, you get: 1, 2, 3, 10, **15**, 20, 25, 30, 40. The middle one is 15.

Sample Test Case 2:
Input:
3 3
5 5 5
1 2 3
8 9 10

Note: Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.

Output:
5
Explanation: If you list all scores in order, you get: 1, 2, 3, 5, **5**, 5, 8, 9, 10. The middle one is 5.

Question 7: The K-th Missing Frequency

Problem Description:

A telecommunications tower transmits signals at specific, unique frequencies. These frequencies are stored in a sorted list. However, some frequencies are "missing" from the sequence. Your task is to find the K-th missing positive integer starting from 1.
For example, if the existing frequencies are [2, 3, 4, 7, 11]:

- The missing numbers are: 1, 5, 6, 8, 9, 10, 12, 13, ...
- The 1st missing number is 1.
- The 5th missing number is 9.

Note:

1. All frequencies in the input and the missing numbers are positive integers.
2. The input array is strictly increasing and contains no duplicate numbers.
3. You must solve this using Binary Search.

Input Format

First Line: Two space-separated integers, **N** (number of existing frequencies) and **K** (the rank of the missing number you need).
Second Line: N space-separated unique integers in ascending order.

Output Format

A single integer representing the K-th missing frequency.

Sample Test Case 1:

Input:
5 5
2 3 4 7 11
Output:
9
Explanation: Missing numbers are 1, 5, 6, 8, 9... The 5th one is 9.

Sample Test Case 2:

Input:
4 2
1 2 3 4
Output:
6
Explanation: No numbers are missing between 1 and 4. The missing sequence starts after 4: 5, 6, 7.... The 2nd missing is 6.

Note: Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.

Question 8: The Minimum Timeline for Painting

Problem Description:

Imagine you are a manager in charge of painting a long row of boards. Each board has a different length, and they are all lined up in a specific order that cannot be changed. You have a team of painters and a few rules to follow. Each painter can only work on a continuous section of boards—this means a painter cannot skip a board in the middle of their assigned section. Every painter works at the same speed, taking exactly 1 minute to paint 1 unit of board length. You cannot split a single board between two different painters; one person must finish the entire board they start. Since all painters start working at the same time, the total time it takes to finish the whole project is determined by the "busiest" painter. Your goal is to figure out a way to assign the boards so that this maximum workload is as small as possible, effectively finding the minimum time needed to complete the job.

Input Format

Line 1: Two integers **N** (the number of boards) and **K** (the number of painters)(K<=N).
Line 2: N space-separated integers representing the lengths of the boards in the order they are lined up.(Length of any board>=1)

Output Format

A single integer representing the minimum possible time to finish the project.

Sample Test Case 1:

Input:
4 2
10 20 30 40
Output:
60
Explanation: Painter 1 handles [10, 20, 30] (Total: 60) and Painter 2 handles [40] (Total: 40); the maximum time taken is 60.

Sample Test Case 2:

Input:
6 3
40 10 30 10 20 30
Output:
50

Explanation: The boards are split as [40, 10], [30, 10], and [20, 30]. The workloads are 50, 40, and 50. The minimum possible maximum time is 50.

Note: Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.

Question 9: The Memory Map Validator

Problem Description

You are developing a memory management tool that tracks how a computer's memory is divided into various "Allocation Intervals". Each interval represents a block of memory defined by a Start Address and an End Address. Because multiple processes can request memory in different ways, these intervals can overlap, be perfectly adjacent, or even be proper subsets of one another (where one memory block exists entirely inside another). Your goal is to determine if the total assigned memory space is Contiguous (no gaps) or Fragmented (contains gaps).

To validate the map, you must follow these steps:

1. Sorting: Organize all intervals by their Start Address in ascending order. If two or more intervals share the same Start Address, sort them by their End Address in ascending order.
2. Connectivity Check: Starting from the first interval, memory is considered "Contiguous" if every subsequent interval begins at or before the furthest memory address reached by any of the previous intervals. If any interval starts at an address higher than the maximum reach of all preceding blocks, a gap exists.
   Contiguity is evaluated relative to the smallest start address among the given intervals, gaps before this address are ignored.

Note: For every interval, Start < End always holds. Intervals can be identical, overlapping, or subsets of each other. A single interval is always contiguous.

Note: Two intervals [a, b] and [b, c] are considered continuous and all intervals are closed.

Input Format

- **Line 1:** An integer **N**, representing the number of memory intervals.
- **Next N Lines:** Two space-separated integers representing the **Start** and **End** addresses of an interval.

Output Format

- **N Lines:** The sorted list of intervals, each on a new line.
- **Last Line:** The string Contiguous if no gaps exist, otherwise Fragmented.

Sample Test Case 1

Input:
4
10 25
0 30
5 15
0 10

Output:
0 10
0 30
5 15
10 25
Contiguous

Explanation: The interval [0, 30] covers everything else; since all other blocks start before 30, there are no gaps.

Sample Test Case 2

Input:
3
10 20
40 50
0 10

Output:
0 10
10 20
40 50
Fragmented

Explanation: 1st and 2nd intervals are continuous, but after address 20, there is a gap until address 40, making the memory fragmented.

Note: Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.

Question 10: The Fair Tax Cap Optimizer

Problem Description

A government needs to collect a target revenue of G units from N citizens. Each citizen has a different income. To maintain fairness, the government implements a Tax Cap (C). The rules for collection are: Any citizen earning an income less than or equal to C pays their entire income as tax. Any citizen earning an income greater than C pays exactly C units and keeps the remainder.
Your goal is to find the smallest integer Cap C that results in a total collected tax of at least G. If the target G cannot be reached even by taxing the entire income of every citizen output -1.

Input Format

- **Line 1:** Two integers, **N** (number of citizens) and **G** (target revenue).
- **Line 2: N** space-separated integers representing individual incomes.

Output Format

- A single integer representing the **minimum Cap C**, or **-1** if the target is unreachable.

Sample Case 1

Input:
3 25
10 20 30
Output: 9
Explanation: With C = 8, tax is 8+8+8=24 (low). With C = 9, tax is 9+9+9=27 (meets target).

Sample Case 2

Input:
4 50
5 10 15 10
Output: -1
Explanation: Total income is 40. Even with the highest possible Cap, the target of 50 cannot be met.

Sample Case 3

Input:
5 45
10 5 20 15 30

Output:
10

Note: Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.