|  |
| --- |
| **CS204(2025)**  **Array**  Instructions for Implementation   * Write the program in C or C++ using standard input/output. * Follow the input/output format strictly. * Ensure your code:   + Handles **edge cases** correctly.   + Meets the **time and space complexity constraints**. * **Do not use STL containers like** vector**,** set**, or** map **in C++.** Use raw arrays unless explicitly allowed. * You are encouraged to write clean, modular, and well-documented code. |

### Q1. Left Rotate Array by n Positions

#### Problem:

Given an integer array of size n, your task is to rotate the array to the left by k positions. This means each element will shift to its left k times, and the first k elements will move to the end of the array. Perform this rotation in-place without using any extra array.

#### Input Format:

* First line: Integer size (size of array)
* Second line: size space-separated integers
* Third line: Integer k (number of positions to rotate)

#### Output Format:

* Print the rotated array elements space-separated on a single line.

#### Constraints:

* 1 ≤ size ≤ 106
* 0 ≤ arr[i] ≤106
* 0 ≤n < size
* **Expected Time Complexity:** O(n)
* **Expected Space Complexity:** O(1)

#### Sample Input 1:

10

1 2 3 4 5 6 7 8 9 10

3

#### Sample Output 1:

4 5 6 7 8 9 10 1 2 3

#### Sample Input 2:

5

10 20 30 40 50

2

#### Sample Output 2:

30 40 50 10 20

### Q2. Interchange Primary and Secondary Diagonal of Matrix

#### Problem:

Given a square matrix of size n x n, write a program to interchange the elements of the primary diagonal (from top-left to bottom-right) with the secondary diagonal (from top-right to bottom-left). Print the updated matrix after the operation.

#### Input Format:

* First line: Integer n (dimension of square matrix)
* Next n lines: Each line has n space-separated integers

#### Output Format:

* Print the modified matrix, each row on a new line

#### Constraints:

* 1 ≤n ≤ 1000
* **Time Complexity:** O(n)
* **Space Complexity:** O(1)

#### Sample Input 1:

3

1 2 3

4 5 6

7 8 9

#### Sample Output 1:

3 2 1

4 5 6

9 8 7

#### Sample Input 2:

2

1 2

3 4

#### Sample Output 2:

2 1

4 3

### Q3. Convert 2D Array to 1D (Row-Major)

#### Problem:

Given a 2D array (matrix) of size m x n, write a program to flatten it into a 1D array using row-major order. That is, the elements should be stored in the 1D array row by row, left to right.

#### Input Format:

* First line: Two integers m and n (matrix dimensions)
* Next m lines: Each with n space-separated integers

#### Output Format:

* Print row-major flattened array

#### Constraints:

* 1 ≤ m , n ≤ 1000
* **Time Complexity:** O(n)
* **Space Complexity:** O(1) (modifying in-place or pointer arithmetic)

#### Sample Input 1:

2 3

1 2 3

4 5 6

#### Sample Output 1:

1 2 3 4 5 6

#### Sample Input 2:

1 4

10 20 30 40

#### Sample Output 2:

10 20 30 40

### Q4. Leaders in an Array

#### Problem:

You are given an array of n integers. An element is called a leader if it is greater than or equal to all elements to its right. Write a program to find and print all such leaders in the array.

#### Input Format:

* First line: Integer n
* Second line: n space-separated integers

#### Output Format:

* Print all leaders in one line.

#### Constraints:

* 1 ≤ n ≤ 106
* −106 ≤ arr[i] ≤ 106
* **Time Complexity:** O(n)

#### Sample Input 1:

6

16 17 4 3 5 2

#### Sample Output 1:

17 5 2

#### Sample Input 2:

5

1 2 3 4 5

#### Sample Output 2:

5

### Q5. Merge Two Sorted Arrays

#### Problem:

Given two sorted arrays of sizes n1 and n2, write a program to merge them into a single sorted array of size n1 + n2. The resulting array should also be in non-decreasing order. You may not use any in-built sort function.

#### Input Format:

* First line: Integer n1
* Second line: n1 sorted integers
* Third line: Integer n2
* Fourth line: n2 sorted integers

#### Output Format:

* One line with the merged sorted array

#### Constraints:

* 1 ≤ n1, n2 ≤ 106
* **Time Complexity:** O(n1 + n2)
* **Space Complexity:** O(n1 + n2)

#### Sample Input 1:

3

1 3 5

3

2 4 6

#### Sample Output 1:

1 2 3 4 5 6

#### Sample Input 2:

2

5 6

3

1 2 3

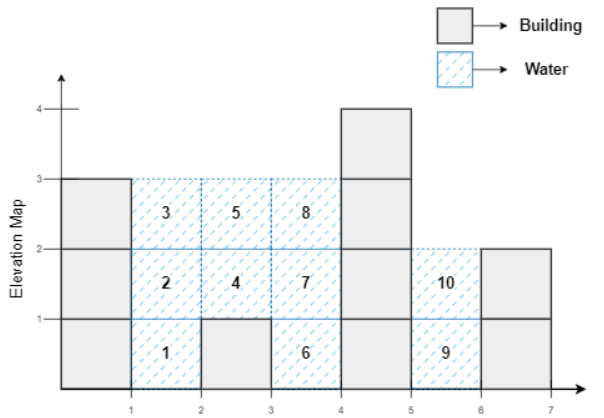
#### Sample Output 2:

1 2 3 5 6

### Q6. Trapping Rainwater

#### Problem:

Given an array of size n representing the heights of blocks where each block has a width of 1, write a program to compute how much rainwater can be trapped after raining. The water is trapped between the bars if there are taller bars on both sides. See fig. and sample 1.



#### Input Format:

* First line: Integer n
* Second line: n space-separated non-negative integers

#### Output Format:

* Print an integer — total trapped water

#### Constraints:

* 1 ≤ n ≤ 106
* 0 ≤ arr[i] ≤ 106
* **Time Complexity:** O(n)
* **Space Complexity:** O(1)

#### Sample Input 1:

5

3 0 1 0 4 0 2

#### Sample Output 1:

10

#### Sample Input 2:

4

1 2 3 4

#### Sample Output 2:

0

### Q7. Convert Sparse Matrix to Triplets

#### Problem:

A sparse matrix is a matrix in which most of the elements are zero. Given a 2D sparse matrix of size m×n, write a program to convert it into a list of triplets, where each triplet contains (row\_index, col\_index, value) of a non-zero element.

#### Input Format:

* First line: Two integers m and n
* Next m lines: Each with n integers

#### Output Format:

* List of triplets, one per line. If there is no triplets print 0 (ie. if all are zeros).

#### Constraints:

* 1 ≤ m,n ≤ 1000

#### Sample Input 1:

2 2

0 1

0 0

#### Sample Output 1:

0 1 1

#### Sample Input 2:

2 2

0 0

0 0

#### Sample Output 2:

0

### Q8. Reconstruct Matrix from Triplets

#### Problem:

Given a list of triplets (row\_index, col\_index, value), reconstruct the original 2D matrix of size m x n. All elements that are not present in the triplets should be assumed to be zero. Print the reconstructed matrix.

#### Input Format:

* First line: Two integers m and n
* Next line: Integer k (number of triplets)
* Next k lines: Each with row col value

#### Output Format:

* Reconstructed matrix (each row on a new line)

#### Constraints:

* 1 ≤ m,n ≤ 1000

#### Sample Input 1:

2 2

2

0 1 7

1 1 8

#### Sample Output 1:

0 7

0 8

#### Sample Input 2:

2 2

0

#### Sample Output 2:

0 0

0 0

### Q9. Find Missing Number in Range 1 to n

#### Problem:

You are given an array of size n-1 containing distinct integers from 1 to n. One number from this range is missing. Write a program to find the missing number using an efficient approach (preferably O(n) time and O(1) space).

#### Input Format:

* First line: Integer n
* Second line: n-1 space-separated integers

#### Output Format:

* Print the missing number.

#### Constraints:

* 2 ≤ n ≤ 106
* All numbers are distinct and within 1 to n
* **Time Complexity:** O(n)
* **Space Complexity:** O(1)

#### Sample Input 1:

4

1 2 4

#### Sample Output 1:

3

#### Sample Input 2:

5

2 3 1 5

#### Sample Output 2:

4

### Q10. Subarray with Given Sum

#### Problem:

Given an array of size n of non-negative integers and a target sum S, determine if there exists a contiguous subarray whose elements sum up exactly to S. If such a subarray exists, print its starting and ending indices (0-based). If not, print "-1".

#### Input Format:

* First line: Integer n
* Second line: n space-separated integers
* Third line: Integer S (target sum)

#### Output Format:

* **Print its starting and ending indices (0-based). If not, print -1.**

#### Constraints:

* 1 ≤ n ≤ 106
* 0 ≤ arr[i] ≤ 106
* **Time Complexity:** O(n)
* **Space Complexity:** O(1)

#### Sample Input 1:

7

1 4 20 3 10 5 17

33

#### Sample Output 1:

2 4

#### Sample Input 2:

6

1 3 4 8 7 9

13

#### Sample Output 2:

-1