Task naming format: fullID_T01L03_2A.c/CPP

Task-01:

Assume your program will take a set of positive integers(1~9) as input and store them in an array. It will stop taking input for any negative number or zero.
Now, the program will show how many times each of the numbers have occurred. (Don't show anything for a number that didn't occur.)

Input	**Output**
3 4 5 7 8 9 2 3 3 4 4 5 0	3 occurs 3 times 4 occurs 3 times 5 occurs 2 times 7 occurs 1 time 8 occurs 1 time 9 occurs 1 time 2 occurs 1 time
5 2 2 3 3 3 1 1 9 -1	3 occurs 3 times 2 occurs 2 times 1 occurs 2 times 5 occurs 1 time 9 occurs 1 time

Note: Output is arranged based on the occurrence of each number. If the frequency of two numbers is the same, it's not mandatory to sort based on the relative ordering.

Note: STL libraries are not allowed for this task. Implement the sort function manually.

Task-02:

Write a program that takes a positive integer number(n) as input and calculates the factorial(n!) value of the number.

$$n! = n \times (n-1) \times (n-2) \times \ldots \ldots \times 1$$

The factorial value should be calculated in two different ways using the following function:

- Factorial_using_iteration (int n): returns int
- Factorial_using_recursion (int n): returns int

Input	**Output**
5	120 (using iteration) 120 (using recursion)
10	3628800 (using iteration) 3628800 (using recursion)
0	1 (using iteration) 1 (using recursion)

Task-03

‘Binary-search’ is an algorithm that is based on a compare and split mechanism. This algorithm is also known as ‘half-interval search’, ‘logarithmic search’, or ‘binary chop’.
The objective is to search for the position of the target value in a sorted array. It compares the target value with the middle element of the array. If the element is equal to the target element, then the algorithm returns the index of the found element. Otherwise, the searching algorithm uses a half section of that array, based on the comparison value, it uses either the first half (when the value is less than the middle) or the second half (when the value is greater than the middle). Then, it does the same for the next half.

Your task is to implement this algorithm in two ways using the following functions:

- Iterative_binary_search(int x, int array[]): returns index
- Recursive_binary_search(int x, int array[]): returns index

Input	**Output**
0 2 6 11 12 18 34 45 55 99 -1 55	8 (using iteration) 8 (using recursion)
25 33 37 43 55 60 -1 42	-1 (using iteration) -1 (using recursion)
2 3 7 13 15 20 25 33 -1 -1	-1 (using iteration) -1 (using recursion)

[**Note**:

- The input array is given in a sorted sequence.
- **Using iterations** means using a loop inside the function that checks for the equality of the middle element.
- **Using recursion** means the function calls itself again and again.
- Array indexing starts from 0.
- It will stop taking input for any negative number.]

Task 04:
Given two integer arrays num1 and num2, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in any order.

Input	**Output**	**Clarification**
1 2 2 1 -1 2 2 -1	2 2	
4 9 5 -1 9 8 9 8 4 -1	4 9	(9 4) is also acceptable.