Data Structures and Algorithms Lab

Lab 02 **Marks 10**

Instructions

- Work in this lab individually. Follow the best coding practices and include comments to explain the logic where necessary.
- You can use your books, notes, handouts, etc. but you are not allowed to borrow anything from your peer student.
- Do not use any **AI** tool for help; doing so will be considered cheating and may result in lab cancellation and possible disciplinary action.
- Test your program thoroughly with various inputs to ensure proper functionality and error handling.
- Show your work to the instructor before leaving the lab to get some or full credit.

Finding Common Elements in Two Arrays

You are given two arrays, both containing ***n*** elements. Write functions to determine the **intersection** (i.e., common elements) of these two arrays, based on the **time complexity** requirements given below. You can assume that there are **no duplicates** in either of the two arrays.

Task 01

Implement the function:

```
int intersection1 (int A[], int B[], int C[], int n)
```

which takes three integer arrays of size ***n*** as parameters. The intersection of arrays **A** and **B** will be stored in array **C**. This function will **return the number of elements** that were stored in array **C**.

The **worst-case time complexity** of this function should be $\boldsymbol{O(n^2)}$. You are not allowed to create any new array(s).

Example 1	**Example 2**	**Example 3**
Input:	**Input:**	**Input:**
A = [7, 1, 5, 3]	A = [1, 6, 9, 2]	A = [2, 1, 4, 3]
B = [4, 3, 8, 5]	B = [3, 7, 8, 4]	B = [3, 2, 1, 4]
n = 4	n = 4	n = 4
Output:	**Output:**	**Output:**
C = [5, 3] // Order is based on appearance in A	C = []	C = [2, 1, 4, 3] // Follows order of A
Return Value: 2	Return Value: 0	Return Value: 4

Task 02

Now, you can assume that both the input arrays contain the numbers which are **sorted** in increasing order. Implement the function:

```
int intersection2 (int A[], int B[], int C[], int n)
```

The **worst-case time complexity** of this function should be $\boldsymbol{O(nlgn)}$. You are not allowed to create any new array(s).

Example 1	**Example 2**	**Example 3**
Input:	**Input:**	**Input:**
A = [1, 3, 5, 7]	A = [2, 4, 6, 8]	A = [10, 20, 30]
B = [3, 5, 8, 9]	B = [1, 3, 5, 7]	B = [10, 20, 30]
n = 4	n = 4	n = 3
Output:	**Output:**	**Output:**
C = [3, 5] // Naturally sorted due to input being sorted	C = []	C = [10, 20, 30]
Return Value: 2	Return Value: 0	Return Value: 3

Task 03: Find the k^{th} Smallest Element

Write a function to determine the k^{th} smallest element of an array containing **n** elements. The prototype of your function should be:

```
int findKthSmallest (int A[], int n, int k)
```

- **A[]** is the array containing n integers.
- **k** is the position (1-based index) of the smallest element you want to find.
- You are strictly **not allowed to sort the array**.

Also, determine the **time complexity** of your function and ensure it is clearly mentioned in your implementation.

Example 1	**Example 2**	**Example 3**
Input: A = [7, 10, 4, 3, 20, 15] n = 6 k = 3 **Output:** 4	**Input:** A = [1, 2, 3, 4, 5] n = 5 k = 2 **Output:** 2	**Input:** A = [9, 3, 2, 8, 5, 6, 7] n = 7 k = 5 **Output:** 7