*Friday, March 25, 2022.*

# Data Structures and Algorithms LAB – Spring 2022
(BS-IT-F20 Morning & Afternoon)

# Lab # 3

## ***Instructions:***

- Attempt the following tasks exactly **in the given order**.
- You must complete all tasks individually. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating would result in an **"F"** grade in this course and lab.
- **Indent** your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for all input/output.
- Do NOT use any **global** or **static** variable(s). However, global named constants may be used.
- Make sure that there are **NO** ***dangling pointers*** or ***memory leaks*** in your programs.

## **Task # 1.1**

Implement the following member function of the **`UnsortedList`** class:

- **`UnsortedList UnsortedList::intersection (const UnsortedList& list2) const;`**

This function should determine the intersection of the current list object with the object list2. This function should create (and return) a new **`UnsortedList`** object containing all the common elements of the two lists. You can assume that there are no duplicates in either of the two lists, and there should be no duplicates in the resultant list as well. The time complexity of your function should be $\boldsymbol{O(n^2)}$ where $\boldsymbol{n}$ is the number of elements in the larger of the two lists.

## **Task # 1.2**

Implement the following member function of the **`SortedList`** class:

- **`SortedList SortedList::intersection (const SortedList& list2) const;`**

This function should determine the intersection of the current list object with the object list2. This function should create (and return) a new **`SortedList`** object containing all the common elements of the two lists. You can assume that there are no duplicates in either of the two lists, and there should be no duplicates in the resultant list as well. The time complexity of your function should be $\boldsymbol{O(n)}$ where $\boldsymbol{n}$ is the number of elements in the larger of the two lists.

## Task # 2

Write a global C++ function to determine and return the **k**th largest element of an *unsorted* array containing **n** integers. The prototype of your function should be:

```
int findKthLargest (const int* arr, int n, int k)
```

In the above prototype, **arr** is an *unsorted* array containing **n** integers from which we want to find the **k**th largest element. Note that:

- You can assume that all elements of the array **arr** are **unique** (i.e. there are no duplicates).
- In your function, you are NOT allowed to modify the contents of the array **arr**.
- The **Space complexity** of your function should be $O(1)$ i.e. you are NOT allowed to declare/use any other array in your implementation.

Also determine the **exact step count** of your implemented function and also determine its **time complexity** (in Big Oh notation).

## Task # 3.1

Implement the following member function of the **UnsortedList** class:

- `bool UnsortedList::isSubset (const UnsortedList& list2) const;`

This function should determine (true or false) whether the current list object (on which this function has been called) is a subset of the object **list2**. You can assume that there are no duplicates in either of the two lists. Also, note that an empty set is a subset of every set. The time complexity of your function should be $O(n^2)$ where $n$ is the number of elements in the larger of the two lists.

## Task # 3.2

Implement the following member function of the **SortedList** class:

- `bool SortedList::isSubset (const SortedList& list2) const;`

This function should determine (true or false) whether the current list object (on which this function has been called) is a subset of the object **list2**. You can assume that there are no duplicates in either of the two lists. Also, note that an empty set is a subset of every set. The time complexity of your function should be linear i.e. $O(n)$ where $n$ is the number of elements in the larger of the two lists.

## Task # 4.1

Implement the following member function of the **`UnsortedList`** class:

- **`UnsortedList UnsortedList::difference (const UnsortedList& list2) const;`**

  This function should determine the difference of the current list object with the object **`list2`**. Remember that, given two sets $\boldsymbol{A}$ and $\boldsymbol{B}$, their difference $\boldsymbol{A-B}$ is defined to be the set of all those elements that are present in $\boldsymbol{A}$ but NOT present in $\boldsymbol{B}$. This function should create (and return) a new **`UnsortedList`** object containing the difference of the two UnsortedLists. You can assume that there are no duplicates in either of the two lists, and there should be no duplicates in the resultant list as well. The time complexity of your function should be $\boldsymbol{O(n^2)}$ where $\boldsymbol{n}$ is the number of elements in the larger of the two lists.

## Task # 4.2

Implement the following member function of the **`SortedList`** class:

- **`SortedList SortedList::difference (const SortedList& list2) const;`**

  This function should create (and return) a new **`SortedList`** object containing the difference of the two SortedLists. You can assume that there are no duplicates in either of the two lists, and there should be no duplicates in the resultant list as well. The time complexity of your function should be $\boldsymbol{O(n)}$ where $\boldsymbol{n}$ is the number of elements in the larger of the two lists.

## Task # 5.1 and 5.2

Union ☺

**VERY IMPORTANT**

In the next Lab, you will need some or all of the functions from Today's Lab. So, make sure that you have the working implementation of **ALL** the functions of Today's Lab, when you come to the next Lab.