Data Structures and Algorithms LAB – Spring 2026

(BS-IT-F24 Morning & Afternoon)

Lab # 3 (Online Lab)

Submission Deadline: ***Saturday, February 14, 2026. (till 6:00 PM)***

Instructions:

- You must complete all tasks individually. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating or usage of AI tools will result in an **"F"** grade in this course and lab.
- Attempt the following tasks exactly **in the given order**.
- **Indent** your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for all input/output.
- Make sure that there are **NO** ***dangling pointers*** or ***memory leaks*** in your programs.
- Late submissions will NOT be accepted, whatever your excuse may be.

Submission Procedure:

i. There are 8 tasks in this lab. You are required to submit C++ programs for the **first 4 tasks**. The remaining tasks are for your practice and you are not required to submit them. But I will assume in quizzes and exams that you have solved all of these questions/tasks ☺

ii. Create 4 empty folders. The folder names MUST be **Task-1**, **Task-2**, **Task-3**, and **Task-4**. Put all of the ***.CPP and .H files*** of each task in its appropriate folder. Do NOT include any other files in your submission, otherwise your submission will NOT be graded. Don't forget to mention your Name, Roll Number and Section in comments at the top of each CPP file.

iii. Now, put all the folders created in the previous step into another folder.

iv. Now, compress the folder (that you created in previous step) in **.RAR** or **.ZIP** format. Then rename the RAR or ZIP file ***exactly*** according to the following format:

`Mor-Lab3-BITF24M123` (for Morning section) OR

`Aft-Lab3-BITF24A456` (for Afternoon section),

where the text shown in **BLUE** should be your **complete Roll Number**.

v. Carefully check your **.RAR** or **.ZIP** file to ensure that all above instructions have been followed.

vi. Finally, submit the *single* **.RAR** or **.ZIP** file through **Google Classroom**. Make sure that you press the **SUBMIT** button after uploading your file.

Note: *If you do not follow the above submission procedure, your Lab will NOT be graded and you will be awarded ZERO marks.*

Important Note:

For each task, you must also write a driver (main) program to illustrate the working of your function on at least 2 to 3 different inputs. At the time of submission, also include the .CPP file of this main function in the submission folder of each task.

Task # 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 $O(n^2)$ where n is the number of elements in the larger of the two lists.

Task # 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 $O(n)$ where n is the number of elements in the larger of the two lists.

Task # 3

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 # 4

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 # 5

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 # 6

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 # 7 and 8

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.