INFORMATION
TECHNOLOGY
UNIVERSITY

Software Engineering Department - ITU
SE200L: Data Structures & Algorithms Lab

Course Instructor: Usama Bin Shakeel	**Dated: 04/09/2024**
Teaching Assistant: Zainab Bashir & Ryan Naveed	**Semester: Fall 2024**
Lab Engineer: Sadia Ijaz	**Batch: BSSE2023B**

Lab 3. Expanding implementation of Array class to support sorting

Name	**Roll number**	**Report (out of 35)**

Checked on: ______________________________

Signature: ________________________________

1.1 Objective

The objective of this lab is to practice problems related to pointers and dynamic memory allocation.

1.2 Equipment and Component

Component Description	Value	Quantity
Computer	Available in lab	1

1.3 Conduct of Lab

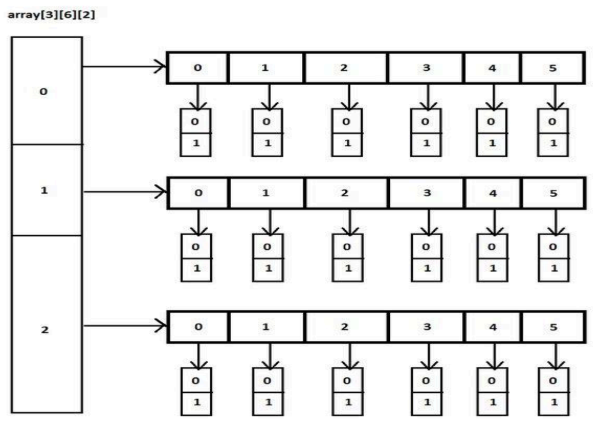
1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

1.4 Theory and Background

A pointer is a variable that stores the memory address of another variable. The type of data a pointer can point to must match the data type of the variable it references. Arrays can be created dynamically, allowing their size to be modified during runtime. Dynamic arrays allocate a contiguous block of memory for their elements, which can be resized during the program's execution.

A **3D dynamic array** extends this concept to three dimensions. It can be visualized as an array of 2D arrays. In a 3D dynamic array, each layer (or "slice") of the array is a 2D array that can be dynamically resized, providing a flexible structure for handling multi-dimensional data. Each layer can have a different number of rows and columns, allowing for a more complex data representation.

A **3D pointer** is a pointer that points to a pointer to a pointer. This multi-level indirection is used to manage a 3D array dynamically. It allows you to create and handle a dynamic structure where each layer is itself a 2D array, and each 2D array can have dynamically varying numbers of rows and columns. This setup facilitates the creation of complex multi-dimensional data structures with flexible sizing.



Templates are a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one.

Lab Tasks

Task 1

Extend the **MyArray** class from Assignment 2 to include sorting functionalities for a 3D array. The sorting should be done row-wise across the entire 3D matrix. The three sorting algorithms to implement are:

1. **Insertion Sort**
2. **Selection Sort**
3. **Merge Sort**

Example:

Consider a 3D matrix before and after sorting:

Before:		**After:**	
Layer 1:	Layer 2:	Layer 1:	Layer 2:
2 1 3	3 23 5	1 2 3	3 5 23
6 5 4	8 0 4	4 5 6	0 4 8
0 8 7	33 4 2	0 7 8	2 4 33

Data Members:

1. **int d0, d1, d2 //columns, rows, layers**
2. **T*** arr3D**
3. **T** arr2D**
4. **T* arr1D**

Member Functions:

1. **Insertion Sort Function**: Sorts the given 1D array using Insertion sort
```
void insertionSort1D(T *& array){
}
```
2. **Selection sort Function**: Sorts the given 1D array using Selection sort
```
void selectionSort1D(T *& array){
}
```
3. **Merge sort Function**: Sorts the given 1D array using Merge sort
```
void mergeSort1D(T *& array){
}
```

4. **Insertion Sort Function**: Sorts the 2D array using 1D Insertion sort function

```
void insertionSort2D(){
}
```

5. **Selection sort Function**: Sorts the 2D array using 1D Selection sort function

```
void selectionSort2D(){
}
```

6. **Merge sort Function**: Sorts the 2D array using 1D Merge sort function

```
void mergeSort2D(){
}
```

7. **Insertion Sort Function**: Sorts the 3D array using 1D Insertion sort function

```
void insertionSort3D(){
}
```

8. **Selection sort Function**: Sorts the 3D array using 1D Selection sort function

```
void selectionSort3D(){
}
```

9. **Merge sort Function**: Sorts the 3D array using 1D Merge sort function

```
void mergeSort3D(){
}
```

10. **Get row Function**: takes input the row number and layer number and returns the row pointer

```
T* getRow3D(int row_num, int layer_num){
}
```

11. **Get row Function**: takes input the row number and returns the row pointer

```
T* getRow2D(int row_num){
}
```

12. **Set Element and Get Element functions** from assignment.

```
void setElement(T value, int index1, int index2 = -1, int index3 = -1)
}
void getElement(int index1, int index2 = -1, int index3 = -1)
}
```

Assessment Rubric for Lab

Performance metric	**CLO**	**Able to complete the task over 80% (4-5)**	**Able to complete the task 50-80% (2-3)**	**Able to complete the task below 50% (0-1)**	**Marks**
1. Realization of experiment	1	Executes without errors excellent user prompts, good use of symbols, spacing in output. The testing has been completed.	Executes without errors, user prompts are understandable,minimum use of symbols or spacing in output. Some testing has been completed.	Does not execute due to syntax errors, runtime errors, user prompts are misleading or non- existent. No testing has been completed.	
2. Conducting experiment	1	Able to make changes and answer all questions.	Partially able to make changes and few incorrect answers.	Unable to make changes and answer all questions.	
3. Computer use	2	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	3	Actively engages and cooperates with other group member(s) in an effective manner.	Cooperates with other group member(s) in a reasonable manner but conduct can be improved.	Distracts or discourages other group members from conducting the experiment	
5. Laboratory safety and disciplinary rules	3	Code comments are added and do help the reader to understand the code.	Code comments are added and do not help the reader to understand the code.	Code comments are not added.	
6. Data collection	3	Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap.	Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables.	Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy.	
7. Data analysis	4	Solution is efficient, easy to understand, and maintain.	A logical solution that is easy to follow but it is not the most efficient.	A difficult and inefficient solution.	
Total (out of 35):					