CSCE4133/5133 – Algorithms
Fall 2023

Assignment 4
Sorting Algorithms

Out Date: Oct. 30, 2023
Due Date: **Nov. 10, 2023**

Instructions:

- **Written Format & Template:** Students can use either Google Doc or Latex
- Write your full name, email address and student ID in the report.
- Write the number of "Late Days" the student has used in the report.
- Submission via BlackBoard
- **Policy:** Can be solved in an individual or a team of two & review the late-day policy.
- **Submissions:** Your submission should be a PDF for the written section & a zip file with your source code implementation.
- For more information, please visit the course website for the homework policy.

Sorting Algorithms

1. Overview

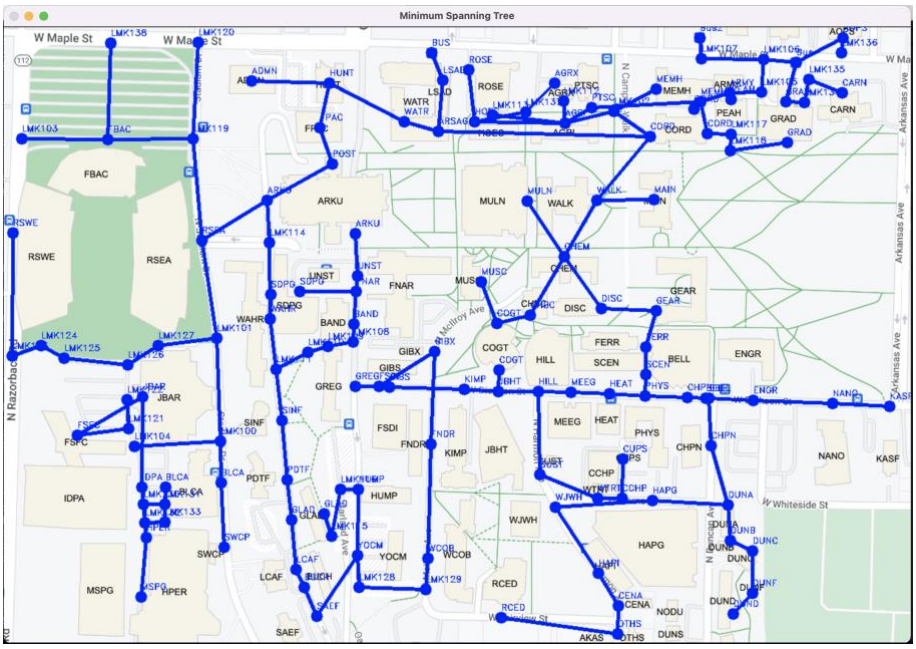

Figure 1: The Minimum Spanning Tree of the Campus Map

In Homework 3, we have studied about the modified breadth first search for the shortest path algorithm and implemented the binary search tree to speed up the searching process. In this homework, we study a new problem related to the sorting algorithms. Now, assume that our university needs to install internet cables between buildings so that all building can connect to each other. The cost to set up a cable between two building is equivalent to the distance between building (the weight of edge). Because we would like to save money for our university, we need to choose a set of connections (edges) between buildings to install so that all buildings can connect to each other and the cost for the installation is as small as possible. In the graph theory, this problem is known as Minimum Spanning Tree (MST). Figure 1 illustrates an example of MST. To solve this problem, we can use the following greedy algorithms:

```
let E be a list of edges in the graph G
sort E in the ascending order of the edge's weight
let MST be the graph with n nodes of G and empty edge
for each edge e in E do
    let u and v be two vertices of e
    if u cannot reach to v on MST then
       add e into MST
    end if
end for
```

The above algorithm is known as the Kruskal's algorithm. In this Homework 4, the implementation of the Kruskal's algorithm is provided. Your task is to sort the list of edges in the ascending order of edge's weight.

In this assignment, you are ***REQUIRED*** to implement THREE sorting algorithms, i.e., ***Quick Sort***, ***Merge Sort***, and ***Heap Sort***. If you think the homework is too long, don't worry, it is long because we provide you with the detail of the descriptions. There are some hints that may help to be success in this homework:

- You should start your homework early.
- You should read the descriptions of the homework carefully before you start working on it.
- Before implementing the required functions, you should try to compile the source code first. Basically, the source code with an empty implementation can be compiled successfully. This step will help you to understand the procedure of each problem.
- If you forgot the sorting algorithms, you are encouraged to revise the lecture slides.

The source code is organized as follows:

File or Folder	***Required Implementation***	***Purpose***
`assets/`	No	Contain the data of homework
`include/`	No	Contain headers files
`src/`	No	Contain source code files
`bin/`	No	Contain an executable file
`include/linked_list.hpp`	No	Define the linked list structure
`include/queue.hpp`	No	Define the queue structure
`include/stack.hpp`	No	Define the stack structure
`include/graph.hpp`	No	Define the graph structure
`include/search.hpp`	No	Define the header of BFS
`include/bst.hpp`	No	Define the header of BST and AVL
`Include/sort.hpp`	No	Define the header of sorting algorithms
`src/data_structures`	No	Implement data structures (i.e., linked list, queue, stack, bst, and avl)
`src/algorithms`	No	Implement the algorithms (i.e., bfs, minimum spanning tree)
`src/sort`	No	Contain the source files of sorting algorithms
`src/sort/qsort.cpp`	Yes	Implement the Quick Sort algorithm
`src/sort/msort.cpp`	Yes	Implement the Merge Sort algorithm
`src/sort/hsort.cpp`	Yes	Implement the Heap Sort algorithm
`src/main.cpp`	No	Implement the main program
`Makefile.windows`	No	Define compilation rules used on Windows
`Makefile.linux`	No	Define compilation rules used on Linux/Mac OS

2. Implementation

In the header file, you are giving the definition of the template sorting function as follows:

```
#include <iostream>
#include <string>

extern std::string sortAlgName;

template<class T>
void sort(std::vector<T> &array, int l, int r);
```

The variable ***sortAlgName*** defines the name of the sorting algorithm and will be initialized in the implementation. The sorting function contains three arguments, i.e., the $array$ will be sorted, l and r defines the index range that will be sorted (i.e., $a[l..r]$). You have to implement three sorting algorithms, i.e., Quick Sort in ***src/sort/qsort.cpp***, Merge Sort in ***src/sort/msort.cpp***, and Heap Sort in ***src/sort/hsort.cpp***. In your code, to compare two array values, you can simplify declare as

```
array[i] < array[j] (array[i] less than array[j])
array[i] > array[j] (array[i] greater than array[j])
array[i] == array[j] (array[i] equal to array[j])
array[i] >= array[j] (array[i] not less than array[j])
array[i] <= array[j] (array[i] not greater than array[j])
```

In this homework, you must implement these algorithms as optimal as possible. The expected complexity of these algorithms is $O(n \log n)$ where n the size of an array.

3. Compilation and Testing

In this homework 4, we use the same environment (Make, GCC/G++, OpenCV Library) used in the previous ones to compile the source code.

In this homework, ***you use only a single Makefile***. It will automatically detect your Operator System (e.g., Linux, MacOS, or Windows) and configurate the compilation rules. For Windows users, you have to make sure that you set the correct paths for MINGW_BIN and OPENCV_DIR:
MINGW_BIN=<path to mingw64>/bin
OPENCV_DIR=<path to opencv>/build/install
For example, if we stored mingw64 and opencv in C:/, we edit these paths as follows:
MINGW_BIN=C:/mingw64/bin
OPENCV_DIR=C:/opencv/build/install

OpenCV Library: In our homework, OpenCV is OPTIONAL and only used for the visualization purpose, you can work on your homework WITHOUT installing and using OpenCV. If you do not want to use OpenCV for visualization, open file ***Makefile*** and edit line ***OPENCV=1*** to ***OPENCV=0.***

To compile and test your source code, you open the terminal and go to the source code folder and type the following commands:

- **make qsort (**or **mingw32-make qsort** on Windows): To compile and run quick sort
- **make msort (**or **mingw32-make msort** on Windows): To compile and run merge sort
- **make hsort (**or **mingw32-make hsort** on Windows): To compile and run heap sort

If you compile and run successfully, you should pass the unit test of the sorting algorithms and see an output as follows:

```
Perform unit test on the sorting algorithm
You are using Quick Sort algorithm
Your sorting implementation is correct
Perform unit test on your implementation with graph
Minimum Spanning Tree:
Edge: 1 0. Cost: 1
Edge: 2 1. Cost: 2
Edge: 3 1. Cost: 3
Edge: 4 2. Cost: 4
Edge: 5 4. Cost: 6
Total Cost: 16
```

For Windows users, if you would like to compile with OpenCV, you have to copy the following files to your Homework folder:

- libopencv_core3413.dll
- libopencv_highgui3413.dll
- libopencv_imgcodecs3413.dll
- libopencv_imgproc3413.dll