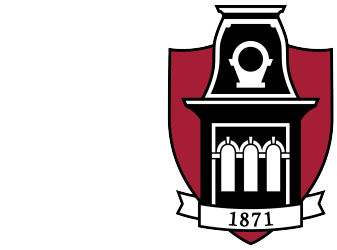

UNIVERSITY OF
ARKANSAS

CSCE 4133/5133 Algorithms

Homework Assignment #4

Submission Deadline: Noon (12PM), April 15, 2024

Instructions:

- **Submissions**: Students can use either Google Doc or LaTEX for writing the report. Your submission should be a zip file of your report and the source code, including your implementation. Name the zip file as lastname_studentID.zip. Submission via BlackBoard.
- If you have any questions please contact us via Teams or email: khoavoho@uark.edu and anhbui@uark.edu.
- **Policy**: Students must start working on this assignment independently & review the late-day policy.

I. Overview

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 edges 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.

For **Graduate students**, you are also required to include a variable to track computational operations, knowing that fundamental operations like comparisons, assignments, and variable reads have a time complexity of O(1). 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 the homework early.
- You should read the descriptions of homework carefully before you start working on the homework.
- 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 binary search tree, 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, DFS, and recursive DFS algorithms
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
src/algorithms	No	Implement the algorithms
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

II. 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 logn) where n the size of an array.

III. 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.

There are two files named Makefile.windows and Makefile.linux. If you are using Windows, rename Makefile.windows to Makefile. If you are using Linux or Mac OSX, rename Makefile.linux to Makefile. 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 implement all methods correctly, the unit test function should return "true" and you should receive an output as follows:

```
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


Minimum Spanning Tree:
Edge: 24 136. Cost: 20
Edge: 29 72. Cost: 21
Edge: 118 117. Cost: 22
Edge: 130 56. Cost: 23
Edge: 74 131. Cost: 23
Edge: 108 75. Cost: 24
Edge: 116 1. Cost: 24
Edge: 134 17. Cost: 25
Edge: 47 65. Cost: 26
Edge: 133 132. Cost: 27
Edge: 110 109. Cost: 28
```

```
Edge: 107 98. Cost: 28
Edge: 131 130. Cost: 29
Edge: 92 44. Cost: 29
Edge: 42 16. Cost: 29
Edge: 57 96. Cost: 29
Edge: 112 11. Cost: 30
Edge: 134 135. Cost: 31
Edge: 115 89. Cost: 31
Edge: 3 23. Cost: 32
Edge: 117 94. Cost: 32
Edge: 58 0. Cost: 33
Edge: 34 93. Cost: 33
Edge: 97 8. Cost: 33
Edge: 109 108. Cost: 34
Edge: 124 125. Cost: 35
Edge: 83 92. Cost: 37
Edge: 22 37. Cost: 38
Edge: 76 69. Cost: 38
Edge: 26 87. Cost: 39
Edge: 121 122. Cost: 39
Edge: 10 7. Cost: 40
Edge: 124 123. Cost: 41
Edge: 11 55. Cost: 41
Edge: 65 79. Cost: 42
Edge: 129 85. Cost: 42
Edge: 7 105. Cost: 43
Edge: 128 20. Cost: 43
Edge: 94 10. Cost: 43
Edge: 106 105. Cost: 44
Edge: 106 36. Cost: 44
Edge: 52 82. Cost: 44
Edge: 75 29. Cost: 44
Edge: 96 40. Cost: 46
Edge: 54 35. Cost: 46
Edge: 74 133. Cost: 47
Edge: 5 2. Cost: 47
Edge: 111 110. Cost: 48
Edge: 73 97. Cost: 48
Edge: 59 135. Cost: 48
Edge: 44 62. Cost: 48
Edge: 126 127. Cost: 48
Edge: 15 83. Cost: 50
Edge: 28 95. Cost: 51
Edge: 22 21. Cost: 51
Edge: 62 52. Cost: 52
Edge: 77 0. Cost: 53
Edge: 36 17. Cost: 54
Edge: 30 18. Cost: 54
Edge: 96 82. Cost: 54
Edge: 12 137. Cost: 55
Edge: 100 67. Cost: 56
Edge: 66 72. Cost: 57
Edge: 69 8. Cost: 57
Edge: 9 114. Cost: 57
Edge: 44 16. Cost: 57
Edge: 102 43. Cost: 59
```

```
Edge: 35 53. Cost: 59
Edge: 42 90. Cost: 60
Edge: 95 41. Cost: 61
Edge: 81 102. Cost: 64
Edge: 42 84. Cost: 64
Edge: 115 116. Cost: 64
Edge: 11 102. Cost: 67
Edge: 30 47. Cost: 67
Edge: 27 56. Cost: 68
Edge: 111 93. Cost: 68
Edge: 5 87. Cost: 69
Edge: 121 39. Cost: 69
Edge: 6 111. Cost: 69
Edge: 60 45. Cost: 70
Edge: 137 60. Cost: 70
Edge: 24 36. Cost: 71
Edge: 91 15. Cost: 74
Edge: 32 29. Cost: 76
Edge: 19 38. Cost: 77
Edge: 71 86. Cost: 77
Edge: 50 118. Cost: 77
Edge: 101 127. Cost: 79
Edge: 6 18. Cost: 80
Edge: 27 64. Cost: 80
Edge: 84 73. Cost: 83
Edge: 106 107. Cost: 84
Edge: 58 25. Cost: 84
Edge: 91 80. Cost: 86
Edge: 126 125. Cost: 86
Edge: 67 70. Cost: 87
Edge: 71 80. Cost: 87
Edge: 51 3. Cost: 88
Edge: 1 20. Cost: 89
Edge: 46 80. Cost: 90
Edge: 128 129. Cost: 90
Edge: 54 80. Cost: 91
Edge: 11 113. Cost: 98
Edge: 20 79. Cost: 98
Edge: 9 28. Cost: 100
Edge: 78 40. Cost: 101
Edge: 73 4. Cost: 102
Edge: 33 39. Cost: 102
Edge: 9 63. Cost: 103
Edge: 114 93. Cost: 103
Edge: 41 49. Cost: 105
Edge: 88 21. Cost: 106
Edge: 90 19. Cost: 107
Edge: 25 82. Cost: 112
Edge: 71 43. Cost: 112
Edge: 41 2. Cost: 113
Edge: 61 119. Cost: 115
Edge: 103 61. Cost: 116
Edge: 100 104. Cost: 116
Edge: 11 60. Cost: 122
Edge: 56 33. Cost: 122
Edge: 99 51. Cost: 125
```

```
Edge: 138 61. Cost: 130
Edge: 120 119. Cost: 130
Edge: 88 4. Cost: 131
Edge: 101 63. Cost: 133
Edge: 119 63. Cost: 137
Edge: 101 100. Cost: 139
Edge: 10 55. Cost: 147
Edge: 99 85. Cost: 154
Edge: 37 14. Cost: 158
Edge: 48 123. Cost: 166
Edge: 5 43. Cost: 286
Total Cost: 9290
```

IV. Submission

Write a report detailing the key ideas for implementing each required function and include the implemented functions themselves. You must also include a screenshot of the final output (from either command prompt or terminal) after running the program.

Write your full name, email address and student ID in the report. Your submission should be a zip file of your report and the source code, including your implementation. Name the zip file as 'lastname_studentID.zip'. Submission via BlackBoard.