CSC 130
Data Structures and Algorithm Analysis

Final Project
250 Points

Complete various tasks related to Data Structures / Algorithms
The purpose of this project is to test that you can apply the knowledge learned in this course while actually coding in Java. Many assignments give you a "blank slate" but I wanted to prepare you for industry which tends to have you work within a pre-existing framework instead of a clean coding canvas. Follow the steps / specifications below for credit.

Steps:

1) Set up a new project in Eclipse (or whatever IDE you use) and create a new package in the project called "Main" (to work with my code files)

2) Download ALL of the files available on the assignment page on Canvas and add these to your project in the Main package (if using Eclipse, you can simply drag and drop.) The file "Main.java" is provided as a courtesy for you so you can see the test file I am using to grade your assignment. Therefore, you know if this file works as expected on your end, it will look good on my end as well.

3) Read the assignment specifications below and do everything mentioned if you wish to receive full credit. I will deduct points for parts that are not implemented (or not done correctly.) *Your program must run without errors or warnings for full credit.*

Specifications
(*60 points*) – In the BST.java file, you need to complete the body of the methods for the three binary tree traversals:

- preorderToString()
- postorderToString()
- levelorderToString()

Each one of these is worth 20 points. Display the node data in the same way as the toString() does (e.g. [3,4,5,6,7,8,9]) as far as formatting is concerned (brackets and commas dividing node numbers).

If you are unsure what preorder, postorder, or levelorder are, you should research these concepts. They all represent different ways to display the data.

Note: You are welcome to make helper methods to aid in solving these problems but make sure to make these methods private (e.g. Abstraction).

(*100 points*) – In the Graph.java file, you need to complete the body of the methods for:

- addWeightedEdge()

- shortestPath()

To receive the 100 points for this portion, both must work as expected. The shortestPath must be formatted similar to the BST traversal (e.g. [3,4,5,6,7,8]) regarding having brackets around the set and having commas divide the node numbers.

The shortestPath must take into consideration to weights of the edges and not simply the number of edges. Therefore, a path could be the shortest path if there are more edges between two points AS LONG AS the total weights of all edges is less in the shortest path than the longer paths.

Think of this as a map where you are finding a short path and the weights being distance.

(*50 points*) – In the Sort.java file, complete the body of the method for:

- MergeSort()

If you are unsure of how MergeSort works, research the topic. It is a recursive sorting method and you MUST use recursion to solve this (for credit).

You are welcome to create helper methods provided you make these methods private.

MergeSort is an optimal algorithm which means it is O(n log n) in the worst case. Since linearithmic is the best we can possibly do for a general-purpose sort in the worst case, merge sort is considered optimal. Your code must be O(n log n) in the worst case.

(*40 points*) – Solve the 3-sum problem. In class, we had done the 2-sum problem which was to find two numbers in an array that equaled a given sum. We didn't have to find all possible combinations but only one. This took $O(n^2)$ time since we used a nested loop to solve.

When you solve the 3-sum problem, you MUST use a HashMap (Java API version) and your solution must be better than $O(n^3)$ time. Using a HashMap, you should be able to solve this problem without using three nested loops.

Finish the body of the method in the Sum.java file for:

- threeSum()

Submission
Zip up a copy of your entire project from and submit via Canvas by the due date. **Only .zip files will be accepted!**

If you have any questions, let me know.