Project 3 – Design Description

Tony Shen

1. Overview of Classes

For this project, I used class Quad that held member variables that were pointers to the 4 directions in our quadtree. Quad also held a pointer n, which stores the address of a struct Node. Node contains all of the information about the city/location in its own member variables, namely: longitude/latitude, population, cost of living, and salary.

Quad has a number of member functions, all pertaining to the specific functions described in the project description. We start with **i**, which simply inserts a node into the quadtree recursively, handling failures when the position is already occupied. This function takes in an instance of Quad alongside its corresponding node, ensures that it is not already in the tree, and inserts it in the correct position. It will return a bool, indicating whether or not the node was successfully inserted.

Quad		
n	Node*	
NW, NE, SW, SE	Quad*	
i	(Node* _n, Quad* quad)	Bool
s	(double _x, _y, Quad* quad)	Quad*
q_max, q_min, q_total	(string _d, _attr, Quad* quad)	Long
print, clear	()	Void
size	()	Long

The next function **s** takes given longitude/latitude values in addition to an instance of Quad* to find a specific city through a similar recursive approach as i. The return value also being of type Quad* means that all elements of this city can be accessed through search. Specifically, it will output the name of the city a specified location. This function is used by a few other functions in our code.

The next function is **q_max**. This will input a direction, attribute (corresponding), as well as a Quad* that will contain a place on the "grid". Starting at this point, we take the direction and find the maximum value of one attribute (p, r, s) facing in that direction from the first. Our next **q_min** is very similar, simply returning the smallest value instead. However this time, we must consider the possibility that one of our directional pointers is null, so we use the ternary conditional operator. Just like the next function **q_total**, the parameters and return type is the same. This last function of the three will add up all values of a specified attribute facing in the direction starting from a location. All 3 functions first set a traversal pointer temp position and direction specified, then traverse through recursively to get an answer (max, min, total).

The next functions, **print, clear,** and **size** all take no parameters, with only size returning an int representing the number of nodes in the tree. Again, the tree is traversed in a simple recursive manner.

2. Constructors/Destructor/Operator overloading
   Starting with struct Node, I utilized a simple constructor to allow initialization of a node to given values (eg. those to be inserted). The constructor for class Quad uses the return of the previous constructor, forming a full tree. Used together, this allows us to initialize a tree very easily, to be used to insert/for traversal. I put in a destructor class for Quad, as both the root and branch trees must be properly dereferenced and deleted recursively.

3. Test Cases
   Test 1: Plotting points on a graph and evaluating the accuracy of functions q_max, q_min, and q_total in specified directions.

   Test 2: Ensuring the above functions were functional in both directions (eg. when nodes inserted NE of one another, try starting at the end and setting direction = SW)

   Test 3: Searching up actual cities' information and plugging them into the program

   Test 4: Only inserting in one direction (eg. NW) and then setting direction to another and ensuring we get "failure" (ensuring we can get to each "failure" statement)

4. Performance
   Looking at the implementation of function i, we can clearly see that if the tree is balanced there will be a time complexity of O(lg(n)) due to its recursive nature. Such a balanced tree resembles an AVL tree, whose complexity as we know is O(lg(n)).

   For clear we must visit every node in the tree and delete/dereference the memory there, thus it is known that the time complexity is O(n).