# P3- Design Document

## CLASS DESIGN:

I have used one class (quadtree class) and one struct(quadTreeNode) in my program. The quadtree class contains the root of quadTreeNode struct (private), constructor and all other member functions(public). My quadTreeNode struct contains all the city information, the direction pointers from the node and the member functions that are necessary to modify the nodes in the quadTree class. My program is implemented using a recursive approach.

## MEMBER VARIABLES & MEMBER FUNCTIONS:

My quadtree class has member variables root of quadTreeNode pointer type and size under the private section. My struct(quadTreeNode) has all the city information and the direction pointers as member variables.

**Member Functions:**

**bool insertRoot(string a, double x, double y, int pop, int cost, int sal)**
**bool insertNode(string a, double x, double y, int pop, int cost, int sal)**

- The 1st function inserts root node when root is empty, and the 2nd function inserts a node in the tree by comparing the latitude and longitude of the cities. The functions return the result recursively.
- 1st and 2nd function: return type is bool and the parameter type is double, string and int.

**quadTreeNode* newNode(string a, double x, double y,int pop, int cost, int sal)**
**quadTreeNode* newRoot(string a,double x, double y,int pop, int cost, int sal)**

- The 1st function creates a new root node when root, assigns the city informations and points to the direction pointers of the root to NULL.The 2nd function creates a new node evrytime the insert function is called.
- 1st and 2nd function: return type is quadTreeNode pointer and the parameter types are double, string and int.

**void sizeTree()**

- Function return the size of the tree. I use variable counter to count the number of nodes in the tree and return it.
- Function return type is void and there are no parameters passed.

**void printTree();**
**void printNode()**

- The 1st function checks if root is empty, and the 2nd function performs an inorder traversal to print the nodes in the order NE, NW, ROOT, SW, SE.
- 1st and 2nd function: return type is void and there are no parameters.

**int searchTree(double x, double y)**
**int searchNode(double x, double y)**

- The 1st function checks if root is empty or size is already zero, and then passes on to the 2nd function which locates the node by comparing the latitudes and longitudes of cities.
- 1st and 2nd function: return type is int and the parameter type is double.

**int maxTree(double x, double y, string direction, string attr)**
**int findMaxNode(double x, double y, string direction, string attr)**
**int findMax(string attr)**

- The 1st function checks if root is empty, and then passes on to the 2nd function which finds the node by comparing the latitudes and longitudes of cities. After finding the particular node, it checks for the direction and

passes it to the 3rd function which recursively compares the maximum of the attribute passed from a bottom to top approach.

- 1st, 2nd function: return type is int and the parameter types are double and string.
- 3rd function: return type is int and the parameter type is string.

**int minTree(double x, double y, string direction, string attr)**
**int findMinNode(double x, double y, string direction, string attr)**
**int findMin(string attr)**

- The 1st function checks if root is empty, and then passes on to the 2nd function which finds the node by comparing the latitudes and longitudes of cities. After finding the node, it checks for the direction and passes it to the 3rd function which recursively compares the minimum of the attribute passed from a bottom to top approach.
- 1st, 2nd function: return type is int and the parameter types are double and string.
- 3rd function: return type is int and the parameter type is string.

**int totalTree(double x, double y, string direction, string attr)**
**int findTotalNode(double x, double y, string direction, string attr)**
**int findTotal(string attr)**

- The 1st function checks if root is empty, and then passes on to the 2nd function which finds the node by comparing the latitudes and longitudes of cities. After finding the node, it checks for the direction and passes it to the 3rd function which recursively adds the sum of the attribute passed from a bottom to top approach.
- 1st, 2nd , 3rd function: return type is int and the parameter types are double and string.

**void clear()**
**void clearNode()**

- The 1st function checks if root is empty or not and assigns the root to NULL. The 2nd function performs a post order tree traversal and deletes the direction pointers in that order and clears the tree. It also decrements the counter every time it deletes a node.
- 1st and 2nd function: return type is void and there are no parameters.

## CONSTRUCTORS:

quadTree() - I have initialized a constructor for my quadTreeNode class which assigns the root of tree to NULL, size and counter to zero.
quadTreeNode() - I have initialized a constructor for my quadtree class which assigns the direction pointers to NULL and the city to zero as well.
**DESTRUCTORS:** I have not initialized any destructors in my program.

## RUNNING TIME:

1. Insert:
   The time complexity for Insert is O(lg(n)) in my design. The running time on a tree of height 'h' is O(h). As the height of the designed quadtree is lg(n), the running time for insert for a balanced tree will be O(lg(n)).
2. Clear:
   The time complexity for clear in my design is O(n). My clear function follows a post order tree traversal to delete the nodes, and this takes O(n) time. So, it traverses each node n times and hence the clear takes O(n) running time.