Project 3: Design Document

Rupakeerthana Vemulapalli

20778222

What class(es) did you design? What are the member variables and member functions for each of these classes?

Classes:

```
public:
quadtree();
~quadtree();

void insert_city(string Name, double Longitude, double Latitude, long int Population, long int Cost_Of_Living, long int Average_Net_Salary);
struct Node* insertion_helper(struct Node* current_node, struct Node* new_city);

struct Node* search_helper(struct Node* current_node, double Longitude, double Latitude);
void search_node(double Longitude, double Latitude);

struct Node* max_population(struct Node* current_node);
struct Node* max_cost_Of_Living(struct Node* current_node);
struct Node* max_average_Net_Salary(struct Node* current_node);
long int q_max_helper(struct Node* current_node, string attr);
void find_q_max(double Longitude, double Latitude, string Direction, string attr);

struct Node* min_population(struct Node* current_node);
struct Node* min_cost_Of_Living(struct Node* current_node);
struct Node* min_average_Net_Salary(struct Node* current_node);
long int q_min_helper(struct Node* current_node, string attr);
void find_q_min(double Longitude, double Latitude, string Direction, string attr);

struct Node* total_population(struct Node* current_node);
struct Node* total_cost_Of_Living(struct Node* current_node);
struct Node* total_average_Net_Salary(struct Node* current_node);
long int q_total_helper(struct Node* current_node, string attr);
void find_q_total(double Longitude, double Latitude, string Direction, string attr);

void print();
void inorder_print_helper(struct Node* current_node);
void clear();
void postorder_clear_helper(struct Node* current_node);
void size();
};
```

*Note: the helper functions are used in each case to implement the actual functions

Member variables:

```
class quadtree{

    private:
    Node* root;
    int tree_size;
    long int total_Population, total_Cost_Of_Living, total_Average_Net_Salary, max_Population, max_Cost_Of_Living, max_Average_Net_Salary;
    long int min_Population, min_Cost_Of_Living, min_Average_Net_Salary;
    long int prev_comparison_value;
    long int prev_comparison_value_min;
```

```
struct Node{

    Key Location;
    Value Attribute;

    struct Node* NW;
    struct Node* NE;
    struct Node* SW;
    struct Node* SE;

};
```

```
struct Value{
    string Name;
    long int Population;
    long int Cost_Of_Living;
    long int Average_Net_Salary;

    Value(string n, long int p, long int COL, long int ANS){
        Name = n;
        Population = p;
        Cost_Of_Living = COL;
        Average_Net_Salary = ANS;
    }
    Value(){
        Name = "";
        Population = 0;
        Cost_Of_Living = 0;
        Average_Net_Salary = 0;
    }

};
```

```
struct Key{
    double x_Longitude;
    double y_Latitude;

    Key(double _x, double _y)
    {
        x_Longitude = _x;
        y_Latitude = _y;
    }
    Key()
    {
        x_Longitude = 0;
        y_Latitude = 0;
    }
};
```

For each class, what are your design decisions regarding constructors?

```
quadtree::quadtree(){
    root = NULL;
    tree_size = 0;

    total_Population =0;
    total_Cost_Of_Living = 0;
    total_Average_Net_Salary =0;

    max_Population =0;
    max_Cost_Of_Living = 0;
    max_Average_Net_Salary =0;

    min_Population =0;
    min_Cost_Of_Living = 0;
    min_Average_Net_Salary =0;

    prev_comparison_value = 0;
    prev_comparison_value_min = 10000000;


}
```

- initializing the root value to NULL
- Other values, the initial value is 0
- For comparing for the minimum value, the initial value is maximum, that's why it is 10000000

For each class, what are your design decisions regarding destructors?

```
quadtree::~quadtree(){
}
```

If It is expected that your implementation has asymptotic upper/tight bound, you should also describe how you have achieved this (or better) runtime in your implementation

Worst-Case Running time for each method of concern

Function	**Worst-case Running time**
i	O(1)
s	O(log(n))- balanced tree height
q_max	O(log(n))- balanced tree height
q_min	O(log(n))- balanced tree height
q_total	O(log(n))- balanced tree height
Print	O(log(n))- balanced tree height
Clear	O(n) – deleting the pointers
Size	O(n)- balanced tree