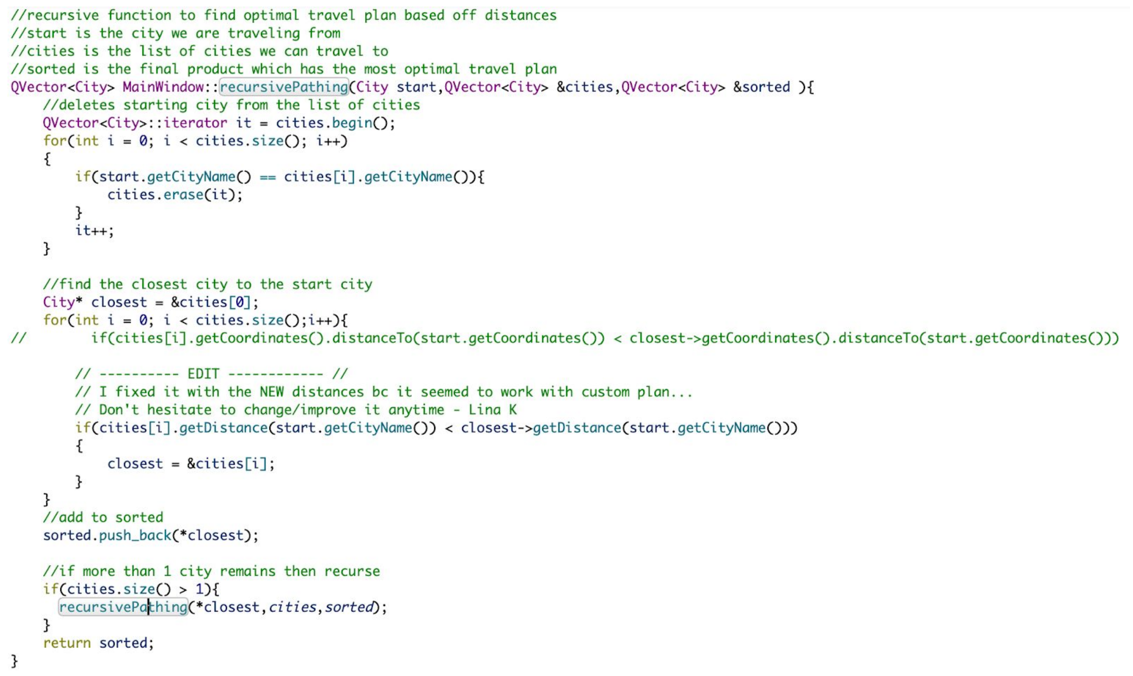
Memory Leak “Big Oh” analysis

1. Recursive Pathing Function

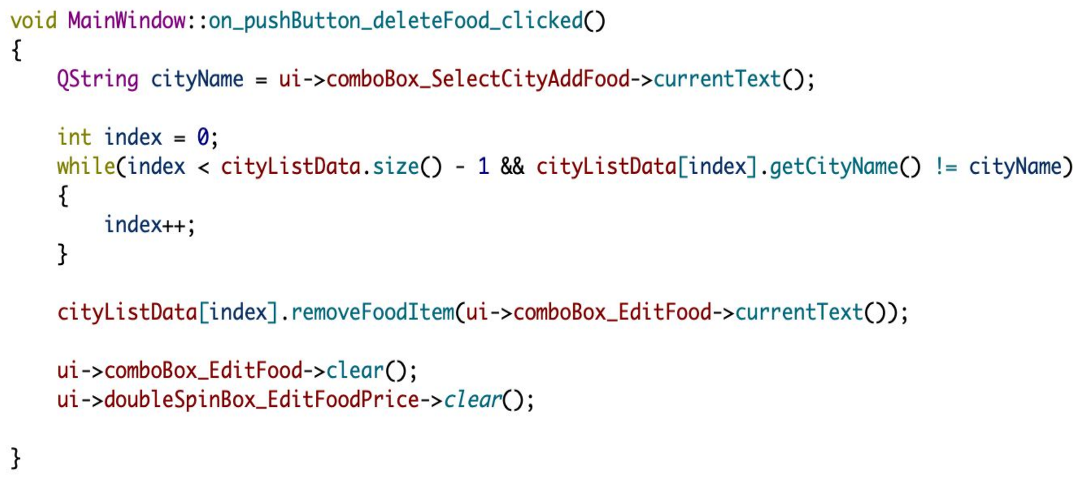


```
//recursive function to find optimal travel plan based off distances
//start is the city we are traveling from
//cities is the list of cities we can travel to
//sorted is the final product which has the most optimal travel plan
QVector<City> MainWindow::recursivePathing(City start,QVector<City> &cities,QVector<City> &sorted ){
    //deletes starting city from the list of cities
    QVector<City>::iterator it = cities.begin();
    for(int i = 0; i < cities.size(); i++)
    {
        if(start.getCityName() == cities[i].getCityName()){
            cities.erase(it);
        }
        it++;
    }

    //find the closest city to the start city
    City* closest = &cities[0];
    for(int i = 0; i < cities.size();i++){
//        if(cities[i].getCoordinates().distanceTo(start.getCoordinates()) < closest->getCoordinates().distanceTo(start.getCoordinates()))

        // ---------- EDIT ------------ //
        // I fixed it with the NEW distances bc it seemed to work with custom plan...
        // Don't hesitate to change/improve it anytime - Lina K
        if(cities[i].getDistance(start.getCityName()) < closest->getDistance(start.getCityName()))
        {
            closest = &cities[i];
        }
    }
    //add to sorted
    sorted.push_back(*closest);

    //if more than 1 city remains then recurse
    if(cities.size() > 1){
      recursivePathing(*closest,cities,sorted);
    }
    return sorted;
}
```

<u>Analysis:</u>

Worst case scenario: This function runs in O(n)

2. Delete Food Function

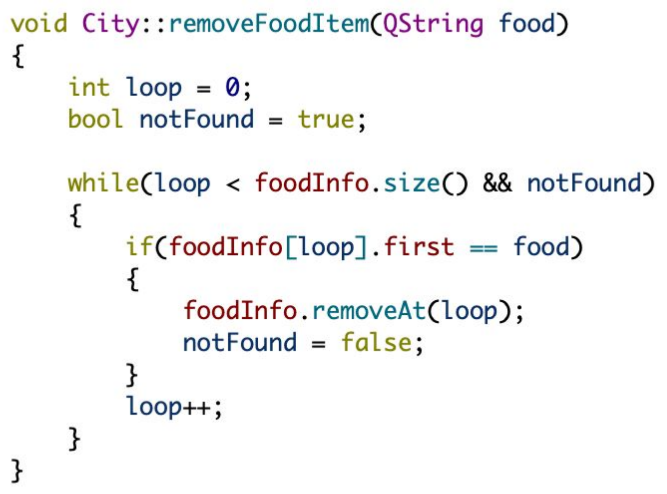


```
void MainWindow::on_pushButton_deleteFood_clicked()
{
    QString cityName = ui->comboBox_SelectCityAddFood->currentText();

    int index = 0;
    while(index < cityListData.size() - 1 && cityListData[index].getCityName() != cityName)
    {
        index++;
    }

    cityListData[index].removeFoodItem(ui->comboBox_EditFood->currentText());

    ui->comboBox_EditFood->clear();
    ui->doubleSpinBox_EditFoodPrice->clear();

}
```

<u>Analysis:</u>

Worst case scenario: This function runs in O(n)

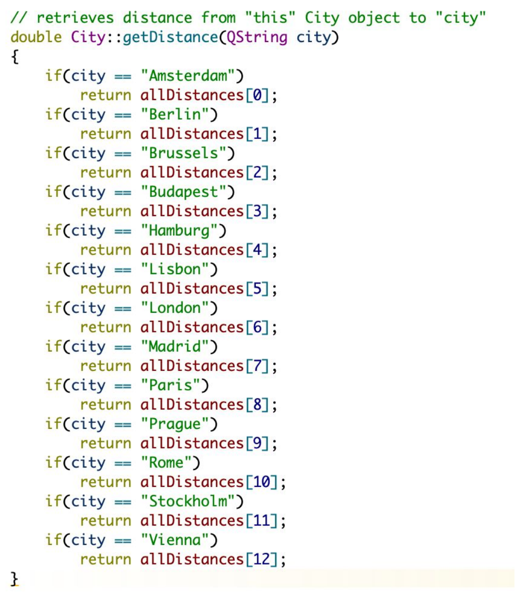
3. Remove Food Item Function (method within the city class)



```
void City::removeFoodItem(QString food)
{
    int loop = 0;
    bool notFound = true;

    while(loop < foodInfo.size() && notFound)
    {
        if(foodInfo[loop].first == food)
        {
            foodInfo.removeAt(loop);
            notFound = false;
        }
        loop++;
    }
}
```

<u>Analysis:</u>
Worst case scenario: This function runs in O(n)

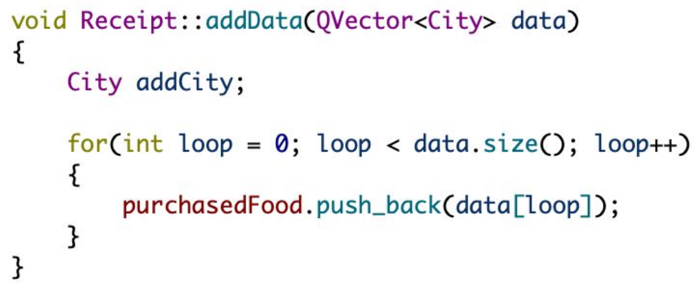
4. Get Distance Function (method within the city class)



```
// retrieves distance from "this" City object to "city"
double City::getDistance(QString city)
{
    if(city == "Amsterdam")
        return allDistances[0];
    if(city == "Berlin")
        return allDistances[1];
    if(city == "Brussels")
        return allDistances[2];
    if(city == "Budapest")
        return allDistances[3];
    if(city == "Hamburg")
        return allDistances[4];
    if(city == "Lisbon")
        return allDistances[5];
    if(city == "London")
        return allDistances[6];
    if(city == "Madrid")
        return allDistances[7];
    if(city == "Paris")
        return allDistances[8];
    if(city == "Prague")
        return allDistances[9];
    if(city == "Rome")
        return allDistances[10];
    if(city == "Stockholm")
        return allDistances[11];
    if(city == "Vienna")
        return allDistances[12];
}
```

<u>Analysis:</u>
Worst case scenario: This function runs in O(1)

5. Add Data Function (method in the Receipt class)



```
void Receipt::addData(QVector<City> data)
{
    City addCity;

    for(int loop = 0; loop < data.size(); loop++)
    {
        purchasedFood.push_back(data[loop]);
    }
}
```

<u>Analysis:</u>

Worst case scenario: This function runs in O(n)