Design and Analysis of Algorithms

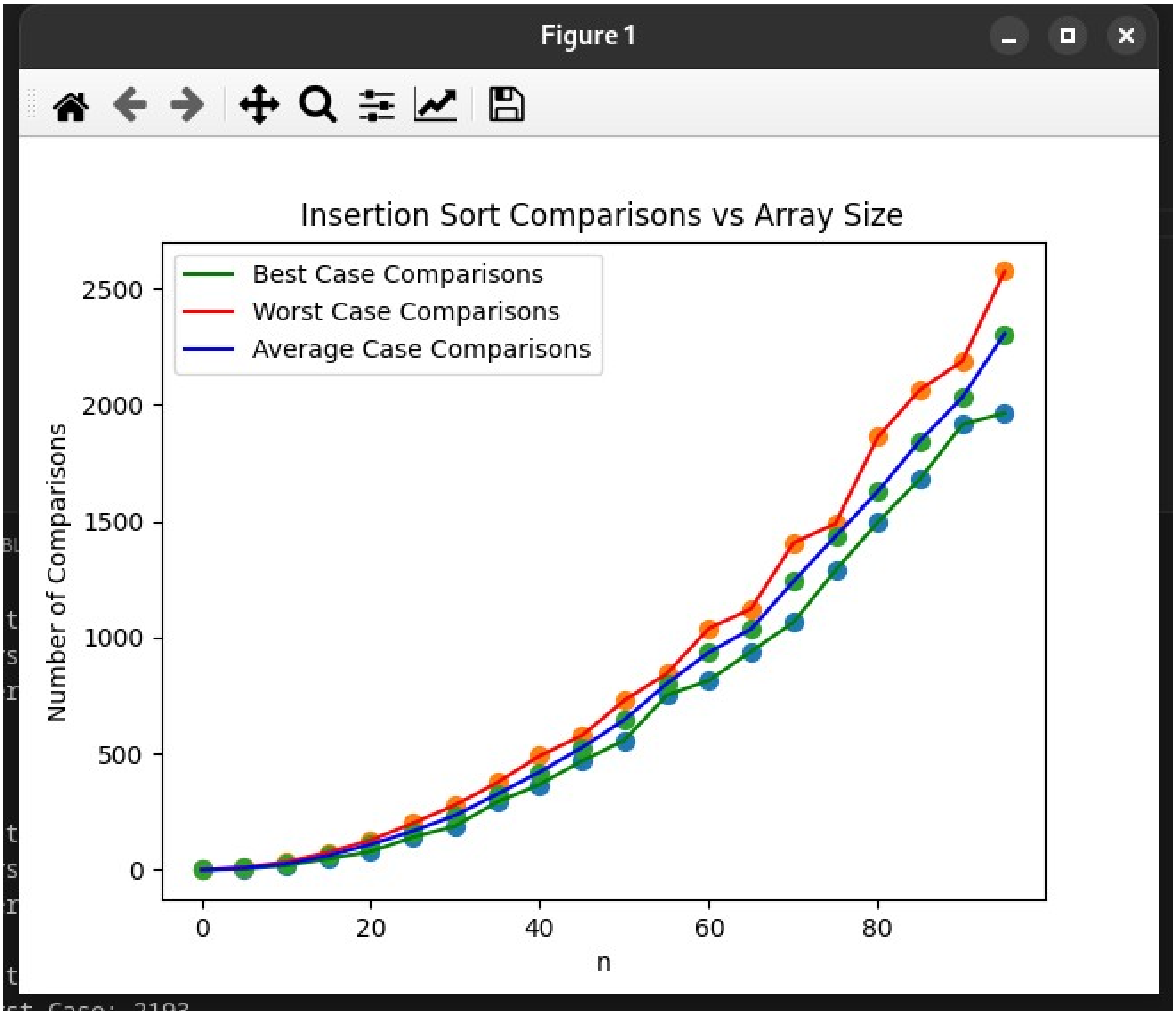
Insertion Sort Assignment
Arun Chandra (09)

1. Run on randomly generated data and plot graph.

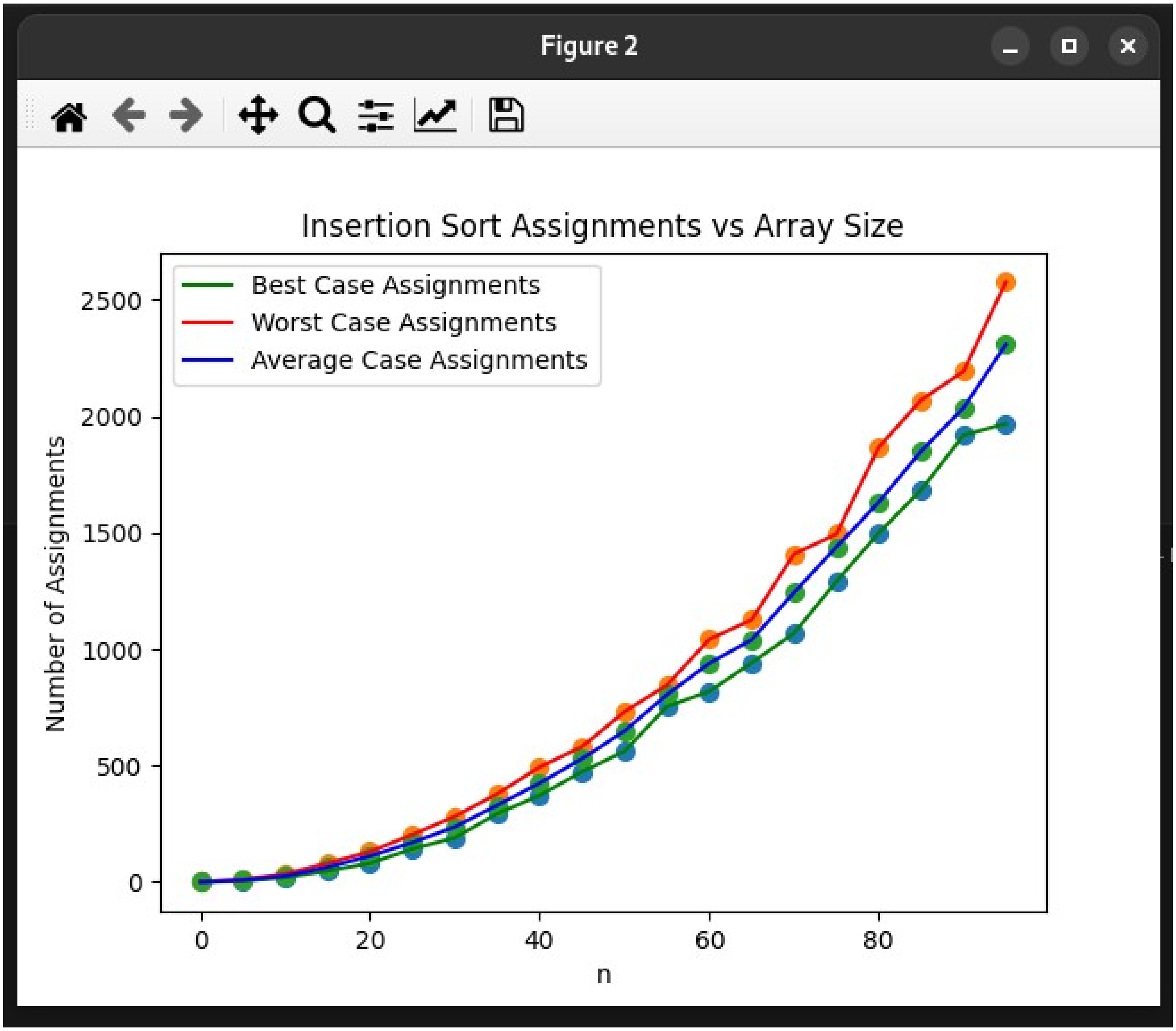
For different array_sizes:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
#include <cstdlib>
#include <ctime>
#include <matplotlibcpp.h>
using namespace std;

namespace plt = matplotlibcpp;

void insertionSort(int A[], int n, int &comparisons, int &assignments){
        /*
    Function: Sorts an array A[] of length n using Insertion Sort.
    Input:
        - A: An array to sort (modified in place).
        - comparisons: Counts comparisons made.
        - assignments: Counts assignments made.
    Output:
        - Sorted array A.
        - Updated counts for comparisons and assignments.
    */
    for( int i = 1; i<=n-1; i++){
        int key = A[i];
        int j = i-1;
        while(j>=0 && A[j]>key){
            //No. of comparisons computed here
            comparisons++;
            A[j+1] = A[j];
            //No. of assignments computed here
            assignments++;
            j = j-1;
        }

        if (j >= 0) {
            comparisons++; // final comparison when key is inserted
        }

        A[j+1] = key;
        assignments++;
    }

}

int main(){

    srand(time(0));
    vector <int> n_values;

    vector <int> best_casesC;
    vector <int> worst_casesC;
    vector <int> average_casesC;

    vector <int> best_casesA;
    vector <int> worst_casesA;
    vector <int> average_casesA;

    for(int n = 0; n<100; n+=5){

        n_values.push_back(n);

        int sumForAverage = 0;
        int averagecaseComparisons;
        int bestcaseComparisons = INT_MAX;
        int worstcaseComparisons = INT_MIN;

        int sumForAverage2 = 0;
        int averagecaseAssignments;
        int bestcaseAssignments = INT_MAX;
        int worstcaseAssignments = INT_MIN;

        for (int i = 0; i < 10; i++) {

            int arr[n];
            int comparisons=0;
            int assignments=0;

            for (int i = 0; i < n; i++) {
                arr[i] = rand() % n + 1;
            }

            insertionSort(arr, n,comparisons, assignments);

            bestcaseComparisons = min(bestcaseComparisons, comparisons);
            worstcaseComparisons = max(worstcaseComparisons, comparisons);
            sumForAverage = sumForAverage + comparisons;


            bestcaseAssignments = min(bestcaseAssignments, assignments);
            worstcaseAssignments = max(worstcaseAssignments, assignments);
            sumForAverage2 = sumForAverage2 + assignments;

        }

        averagecaseComparisons = (sumForAverage / 10);
        averagecaseAssignments = (sumForAverage2/ 10);
        best_casesC.push_back(bestcaseComparisons);
        worst_casesC.push_back(worstcaseComparisons);
        average_casesC.push_back(averagecaseComparisons);

        best_casesA.push_back(bestcaseAssignments);
        worst_casesA.push_back(worstcaseAssignments);
        average_casesA.push_back(averagecaseAssignments);

        cout << "*** No. of comparisons in (n= " << n << ") ****" << endl;
        cout << "Best Case: " << bestcaseComparisons << endl;
        cout << "Worst Case: " << worstcaseComparisons << endl;
        cout << "Average Case: " << averagecaseAssignments<< endl;
        cout << "*** No. of Assignments in (n= " << n << ") ****" << endl;
        cout << "Best Case: " << bestcaseAssignments<< endl;
        cout << "Worst Case: " << worstcaseAssignments << endl;
        cout << "Average Case: " << averagecaseAssignments<< endl;
        cout<<endl;
        cout<<endl;

    }

    // Plotting comparisons
    plt::figure();
    plt::scatter(n_values, best_casesC, 50.0); // Increase the size
    plt::scatter(n_values, worst_casesC, 50.0);
    plt::scatter(n_values, average_casesC, 50.0);
    plt::plot(n_values, best_casesC, {{"label", "Best Case Comparisons"}, {"color", "green"}});
    plt::plot(n_values, worst_casesC, {{"label", "Worst Case Comparisons"}, {"color", "red"}});
    plt::plot(n_values, average_casesC, {{"label", "Average Case Comparisons"}, {"color", "blue"}});
    plt::xlabel("n");
    plt::ylabel("Number of Comparisons");
    plt::title("Insertion Sort Comparisons vs Array Size");
    plt::legend();

    // Plotting assignments
    plt::figure();
    plt::scatter(n_values, best_casesA, 50.0);
    plt::scatter(n_values, worst_casesA, 50.0);
    plt::scatter(n_values, average_casesA, 50.0);
    plt::plot(n_values, best_casesA, {{"label", "Best Case Assignments"}, {"color", "green"}});
    plt::plot(n_values, worst_casesA, {{"label", "Worst Case Assignments"}, {"color", "red"}});
    plt::plot(n_values, average_casesA, {{"label", "Average Case Assignments"}, {"color", "blue"}});
    plt::xlabel("n");
    plt::ylabel("Number of Assignments");
    plt::title("Insertion Sort Assignments vs Array Size");
    plt::legend();

    // Show the plots
    plt::show();
    return 0;
}
```

Output:



```
rst Case: 2193
```

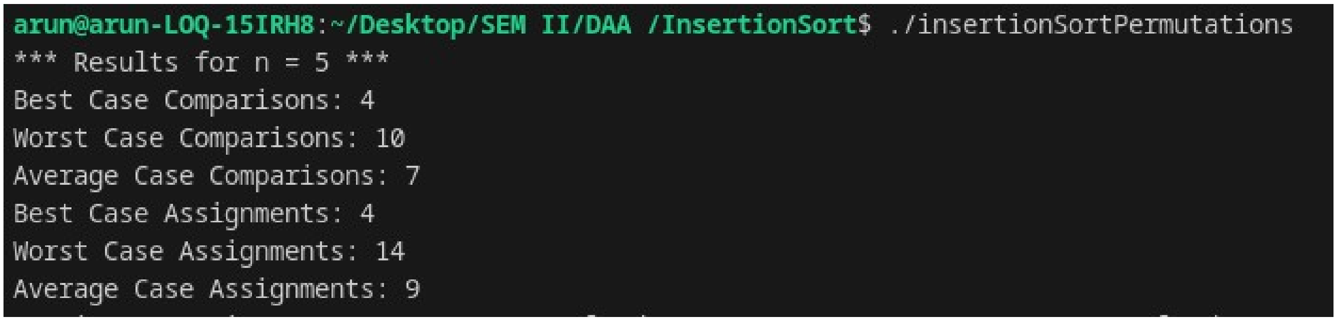


For n=5, different permutations:

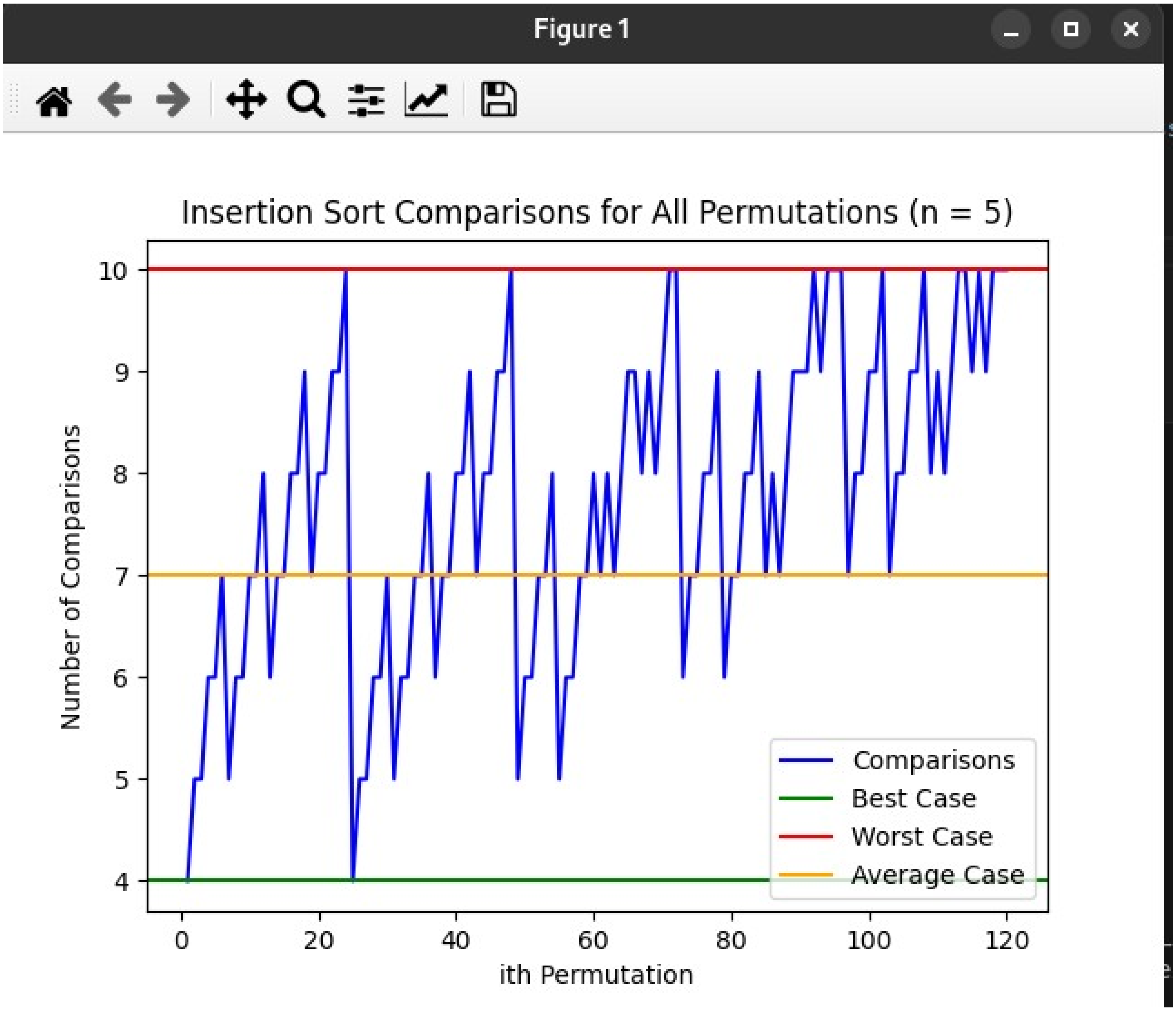
```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
#include <matplotlibcpp.h>
using namespace std;
namespace plt = matplotlibcpp;

void insertionSort(vector<int>& A, int &comparisons, int &assignments) {
        /*
    Function: Sorts a vector using Insertion Sort.
    Input:
        - A: Vector to sort (modified in place).
        - comparisons: Counts comparisons made.
        - assignments: Counts assignments made.
    Output:
        - Sorted vector A.
        - Updated counts for comparisons and assignments.
    */
    int n = A.size();
     for( int i = 1; i<=n-1; i++){
        int key = A[i];
        int j = i-1;
        while(j>=0 && A[j]>key){
            //No. of comparisons computed here
            comparisons++;
            A[j+1] = A[j];
            //No. of assignments computed here
            assignments++;
            j = j-1;
        }

        if (j >= 0) {
            comparisons++; // final comparison when key is inserted
        }

        A[j+1] = key;
        assignments++;
    }
}

int main() {
    int n = 5; // Fixed size of the array
    vector<int> array(n);
    for (int i = 0; i < n; i++) {
        array[i] = i + 1; // Initialize array with values 1, 2, 3, 4, 5
    }

    vector<int> comparisonCounts;
    vector<int> assignmentCounts;

    int sumForAverage = 0;
    int sumForAverage2 = 0;
    int bestComparisons = INT_MAX;
    int worstComparisons = INT_MIN;
    int bestAssignments = INT_MAX;
    int worstAssignments = INT_MIN;

    // Generate all permutations of the array
    do {
        int comparisons = 0;
        int assignments = 0;
        vector<int> tempArray = array; // Copy the array for sorting

        insertionSort(tempArray, comparisons, assignments);

        // Update best and worst cases
        bestComparisons = min(bestComparisons, comparisons);
        worstComparisons = max(worstComparisons, comparisons);
        bestAssignments = min(bestAssignments, assignments);
        worstAssignments = max(worstAssignments, assignments);
        // Accumulate totals for average calculation
        sumForAverage += comparisons;
        sumForAverage2 += assignments;

        // Store counts for plotting
        comparisonCounts.push_back(comparisons);
        assignmentCounts.push_back(assignments);

    } while (next_permutation(array.begin(), array.end()));

    // Calculate average cases
    int averageComparisons = sumForAverage / comparisonCounts.size();
    int averageAssignments = sumForAverage2 / assignmentCounts.size();

    // Print results
    cout << "*** Results for n = " << n << " ***" << endl;
    cout << "Best Case Comparisons: " << bestComparisons << endl;
    cout << "Worst Case Comparisons: " << worstComparisons << endl;
    cout << "Average Case Comparisons: " << averageComparisons << endl;
    cout << "Best Case Assignments: " << bestAssignments << endl;
    cout << "Worst Case Assignments: " << worstAssignments << endl;
    cout << "Average Case Assignments: " << averageAssignments << endl;

    // Plotting the results
    // As 5 factorial is 120..
    // So there are 120 different permutations

    vector<int> permutationIndices(120);
    for (int i = 0; i < permutationIndices.size(); i++) {
        permutationIndices[i] = i + 1;
    }

    // Plot comparisons
    plt::figure();
    plt::plot(permutationIndices, comparisonCounts, {{"label", "Comparisons"}, {"color", "blue"}});
    plt::axhline(bestComparisons, 0, permutationIndices.size(), {{"label", "Best Case"}, {"color", "green"}});
    plt::axhline(worstComparisons, 0, permutationIndices.size(), {{"label", "Worst Case"}, {"color", "red"}});
    plt::axhline(averageComparisons, 0, permutationIndices.size(), {{"label", "Average Case"}, {"color", "orange"}});
    plt::xlabel("ith Permutation");
    plt::ylabel("Number of Comparisons");
    plt::title("Insertion Sort Comparisons for All Permutations (n = 5)");
    plt::legend();

    // Plot assignments
    plt::figure();
    plt::plot(permutationIndices, assignmentCounts, {{"label", "Assignments"}, {"color", "blue"}});
    plt::axhline(bestAssignments, 0, permutationIndices.size(), {{"label", "Best Case"}, {"color", "green"}});
    plt::axhline(worstAssignments, 0, permutationIndices.size(), {{"label", "Worst Case"}, {"color", "red"}});
    plt::axhline(averageAssignments, 0, permutationIndices.size(), {{"label", "Average Case"}, {"color", "orange"}});
    plt::xlabel("ith Permutation");
    plt::ylabel("Number of Assignments");
    plt::title("Insertion Sort Assignments for All Permutations (n = 5)");
    plt::legend();

    // Show plots
```

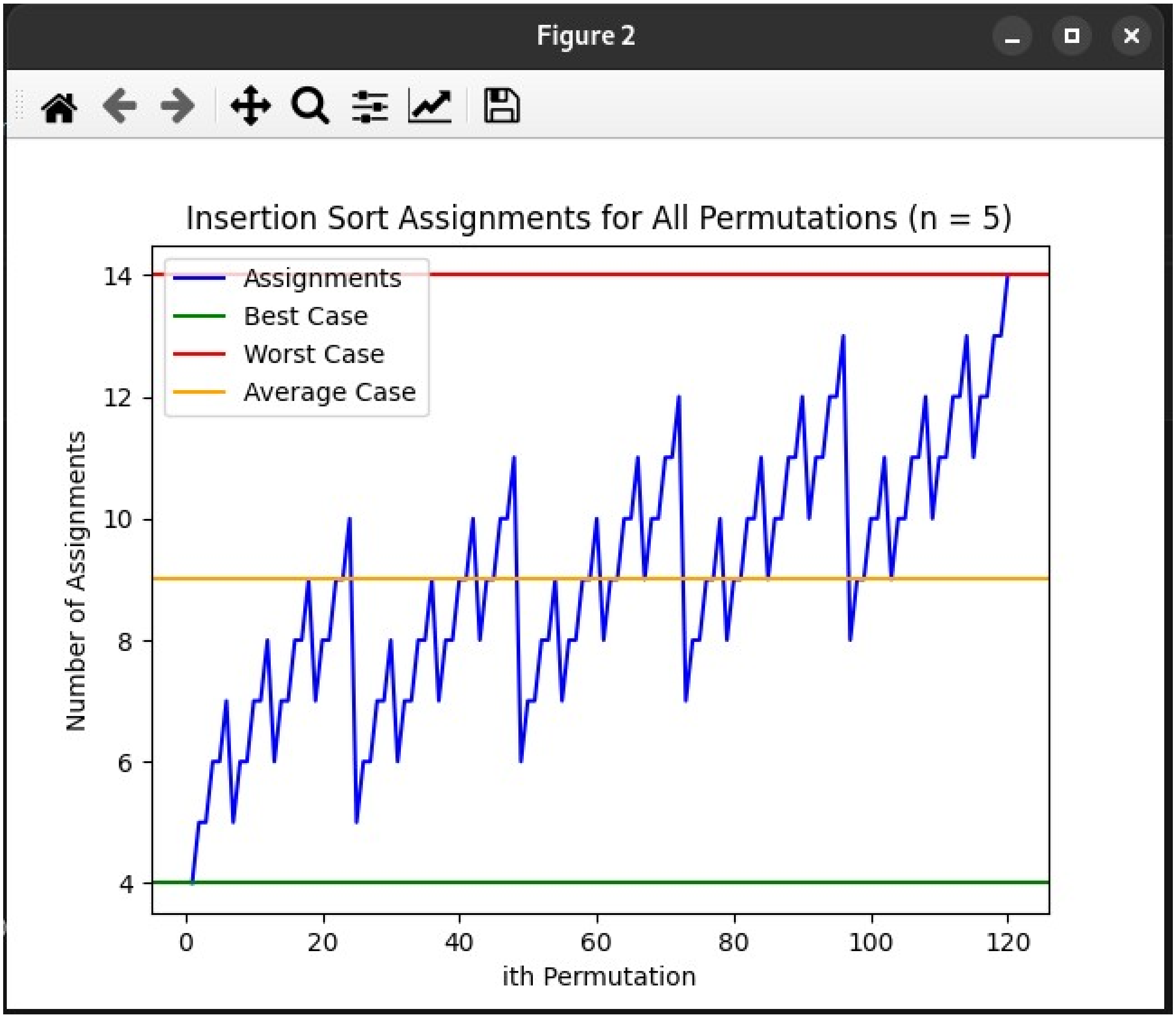
```
    plt::show();

    return 0;
}
```

Output:



```
arun@arun-LOQ-15IRH8:~/Desktop/SEM II/DAA /InsertionSort$ ./insertionSortPermutations
*** Results for n = 5 ***
Best Case Comparisons: 4
Worst Case Comparisons: 10
Average Case Comparisons: 7
Best Case Assignments: 4
Worst Case Assignments: 14
Average Case Assignments: 9
```



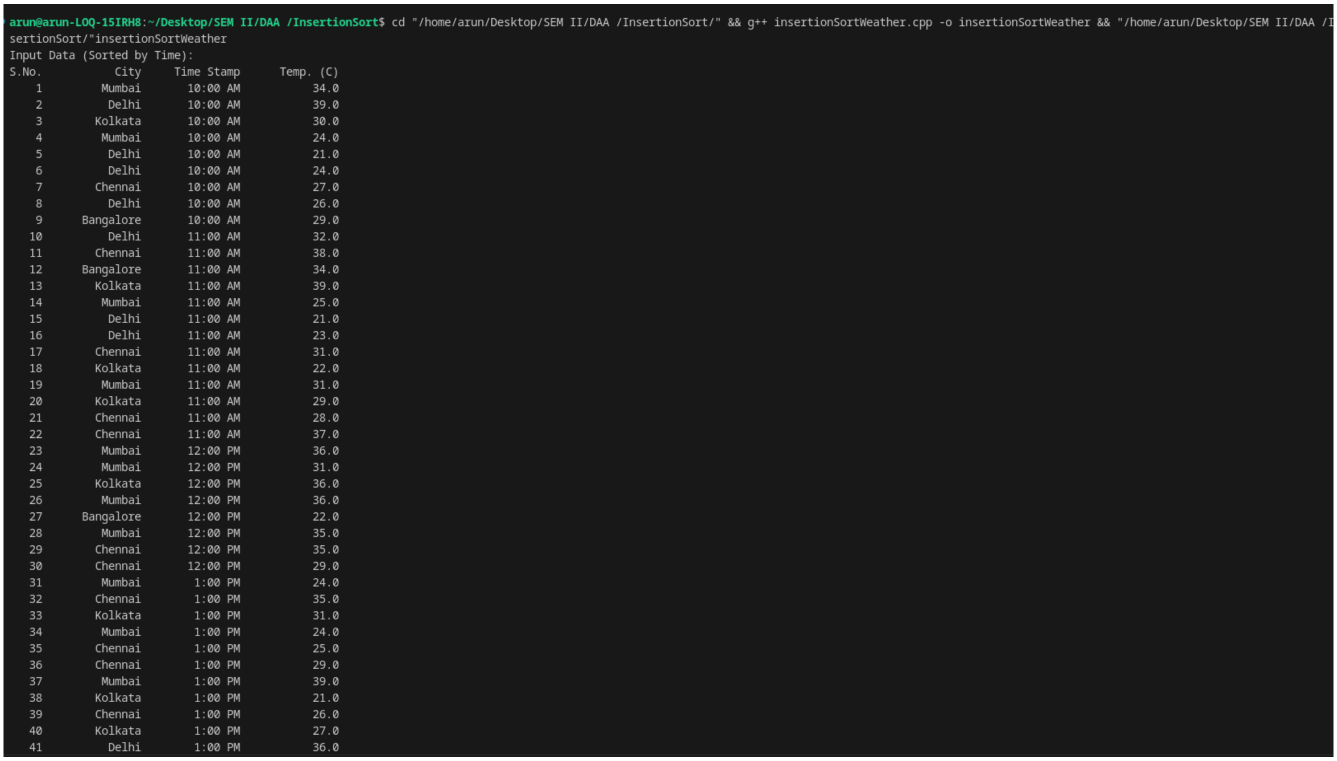


2. Run on weather data of size at least 100 and argue that IS is stable.

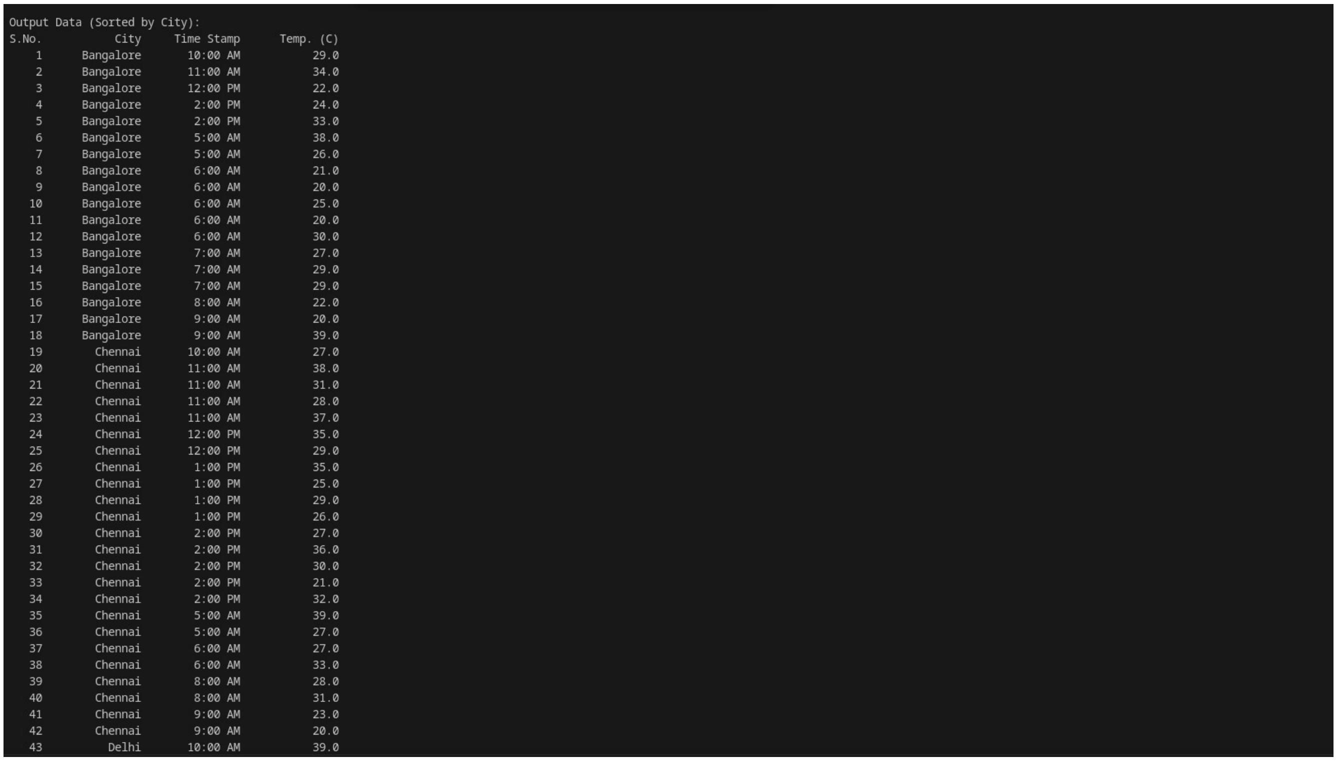
```
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
#include <algorithm>

using namespace std;

struct WeatherData {
    string city;
    string timeStamp;
    double temperature;
};
```

```
void insertionSort(vector<WeatherData>& data) {
    for (int i = 1; i <= data.size()-1; i++) {
        WeatherData key = data[i];
        int j = i - 1;

        while (j >= 0 && data[j].city > key.city) {
            data[j + 1] = data[j];
            j = j - 1;
        }
        data[j + 1] = key;
    }
}

vector<WeatherData> generateWeatherData(int numSamples) {
    vector<string> cities = {"Delhi", "Bangalore", "Mumbai", "Chennai", "Kolkata"};
    vector<string> times = {"5:00 AM", "6:00 AM", "7:00 AM", "8:00 AM", "9:00 AM",
                            "10:00 AM", "11:00 AM", "12:00 PM", "1:00 PM", "2:00 PM"};
    vector<WeatherData> data;

    for (int i = 0; i < numSamples; ++i) {
        WeatherData record;
        record.city = cities[rand() % cities.size()];
        record.timeStamp = times[rand() % times.size()];
        record.temperature = 20 + (rand() % 20);
        data.push_back(record);
    }

    sort(data.begin(), data.end(), [](const WeatherData& a, const WeatherData& b) {
        return a.timeStamp < b.timeStamp;
    });

    return data;
}

void printWeatherData(const vector<WeatherData>& data) {
    cout << setw(5) << "S.No." << setw(15) << "City" << setw(15) << "Time Stamp" << setw(15) << "Temp. (C)" << endl;
    for (int i = 0; i <= data.size()-1; i++) {
        cout << setw(5) << i + 1 << setw(15) << data[i].city << setw(15) << data[i].timeStamp << setw(15) << fixed <<
setprecision(1) << data[i].temperature << endl;
    }
}

int main() {
    // Generate weather data with at least 100 samples
    vector<WeatherData> weatherData = generateWeatherData(100);

    // Print the input data sorted by time
    cout << "Input Data (Sorted by Time):" << endl;
    printWeatherData(weatherData);

    // Sort the data by city using Insertion Sort
    insertionSort(weatherData);

    // Print the output data sorted by city
    cout << "\nOutput Data (Sorted by City):" << endl;
    printWeatherData(weatherData);

    return 0;
}
```

Output of Code to Illustrate Stability:



```
arun@arun-LOQ-15IRH8:~/Desktop/SEM II/DAA /InsertionSort$ cd "/home/arun/Desktop/SEM II/DAA /InsertionSort/" && g++ insertionSortWeather.cpp -o insertionSortWeather && "/home/arun/Desktop/SEM II/DAA /I
sertionSort/"insertionSortWeather
Input Data (Sorted by Time):
S.No.           City     Time Stamp      Temp. (C)
    1         Mumbai       10:00 AM           34.0
    2          Delhi       10:00 AM           39.0
    3        Kolkata       10:00 AM           30.0
    4         Mumbai       10:00 AM           24.0
    5          Delhi       10:00 AM           21.0
    6          Delhi       10:00 AM           24.0
    7        Chennai       10:00 AM           27.0
    8          Delhi       10:00 AM           26.0
    9      Bangalore       10:00 AM           29.0
   10          Delhi       11:00 AM           32.0
   11        Chennai       11:00 AM           38.0
   12      Bangalore       11:00 AM           34.0
   13        Kolkata       11:00 AM           39.0
   14         Mumbai       11:00 AM           25.0
   15          Delhi       11:00 AM           21.0
   16          Delhi       11:00 AM           23.0
   17        Chennai       11:00 AM           31.0
   18        Kolkata       11:00 AM           22.0
   19         Mumbai       11:00 AM           31.0
   20        Kolkata       11:00 AM           29.0
   21        Chennai       11:00 AM           28.0
   22        Chennai       11:00 AM           37.0
   23         Mumbai       12:00 PM           36.0
   24         Mumbai       12:00 PM           31.0
   25        Kolkata       12:00 PM           36.0
   26         Mumbai       12:00 PM           36.0
   27      Bangalore       12:00 PM           22.0
   28         Mumbai       12:00 PM           35.0
   29        Chennai       12:00 PM           35.0
   30        Chennai       12:00 PM           29.0
   31         Mumbai        1:00 PM           24.0
   32        Chennai        1:00 PM           35.0
   33        Kolkata        1:00 PM           31.0
   34         Mumbai        1:00 PM           24.0
   35        Chennai        1:00 PM           25.0
   36        Chennai        1:00 PM           29.0
   37         Mumbai        1:00 PM           39.0
   38        Kolkata        1:00 PM           21.0
   39        Chennai        1:00 PM           26.0
   40        Kolkata        1:00 PM           27.0
   41          Delhi        1:00 PM           36.0
```



```
Output Data (Sorted by City):
S.No.           City     Time Stamp      Temp. (C)
    1      Bangalore       10:00 AM           29.0
    2      Bangalore       11:00 AM           34.0
    3      Bangalore       12:00 PM           22.0
    4      Bangalore        2:00 PM           24.0
    5      Bangalore        2:00 PM           33.0
    6      Bangalore        5:00 AM           38.0
    7      Bangalore        5:00 AM           26.0
    8      Bangalore        6:00 AM           21.0
    9      Bangalore        6:00 AM           20.0
   10      Bangalore        6:00 AM           25.0
   11      Bangalore        6:00 AM           20.0
   12      Bangalore        6:00 AM           30.0
   13      Bangalore        7:00 AM           27.0
   14      Bangalore        7:00 AM           29.0
   15      Bangalore        7:00 AM           29.0
   16      Bangalore        8:00 AM           22.0
   17      Bangalore        9:00 AM           20.0
   18      Bangalore        9:00 AM           39.0
   19        Chennai       10:00 AM           27.0
   20        Chennai       11:00 AM           38.0
   21        Chennai       11:00 AM           31.0
   22        Chennai       11:00 AM           28.0
   23        Chennai       11:00 AM           37.0
   24        Chennai       12:00 PM           35.0
   25        Chennai       12:00 PM           29.0
   26        Chennai        1:00 PM           35.0
   27        Chennai        1:00 PM           25.0
   28        Chennai        1:00 PM           29.0
   29        Chennai        1:00 PM           26.0
   30        Chennai        2:00 PM           27.0
   31        Chennai        2:00 PM           36.0
   32        Chennai        2:00 PM           30.0
   33        Chennai        2:00 PM           21.0
   34        Chennai        2:00 PM           32.0
   35        Chennai        5:00 AM           39.0
   36        Chennai        5:00 AM           27.0
   37        Chennai        6:00 AM           27.0
   38        Chennai        6:00 AM           33.0
   39        Chennai        8:00 AM           28.0
   40        Chennai        8:00 AM           31.0
   41        Chennai        9:00 AM           23.0
   42        Chennai        9:00 AM           20.0
   43          Delhi       10:00 AM           39.0
```

The relative order of records with the same city (Bengalore or `Chennai`) is maintained. This demonstrates that Insertion Sort is stable. For example, In the sorted output, the time and temperature associated with the first instance of "Bengalore" remain in the same relative order as in the input. This behavior is a hallmark of stability in sorting algorithms.