DAA Assignment

Name – Mohit Kumar Singh

Roll No : 42

Course : MCA 2nd SEM

1.Run on randomly generated data and plot graph.

//Program 1 - It generate random arrays and after doing insertion sort in each array it compares all the comparisions to get the best,worst and average time complexities

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <algorithm> //for rnad and srand
#include <ctime>
#include <fstream>
using namespace std;


int insertionSort(int *arr, int size){
   int comparision = 0;
   int key;
   int j;
   for (int i = 1; i < size; i++)
   {
      key = arr[i];
      j = i - 1;
      while (j >= 0 && key < arr[j])
      {
         comparision++;
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
```

```
    }
    return comparision;
}

int main()
{
    srand(time(0));
    int minBound = 1, maxBound = 100;
    int *arr;
    // file opening
    ofstream csvFile("search_comparisons.csv");
    csvFile << "DataSize,BestCase,WorstCase,AverageCase\n";

    for (int i = 10; i <= 100; i = i + 5)
    {
        int bestCase = INT16_MAX;
        int worstCase = INT16_MIN;
        int averageCase = 0;
        for (int j = 0; j < 10; j++)
        {
            arr = new int[i];
            for (int k = 0; k < i; k++)
            {
                int randomNumber = minBound + rand() % (maxBound - minBound + 1);
                arr[k] = randomNumber;
            }
            int compare = insertionSort(arr, i);
            bestCase = bestCase > compare ? compare : bestCase;
            worstCase = worstCase > compare ? worstCase : compare;
            averageCase += compare;

            delete[] arr;
            // printArray(arr, i);
        }
```

```
        // inserting data in the csv file
        csvFile << i << "," << bestCase << "," << worstCase << "," << averageCase / 10 << "\n";
    }
    return 0;
}
```

//csv file after running the above program

DataSize,BestCase,WorstCase,AverageCase

10,14,29,22

15,46,61,52

20,64,132,97

25,123,185,152

30,136,251,204

35,234,341,287

40,343,459,408

45,457,543,487

50,477,701,582

55,562,812,709

60,868,974,903

65,970,1149,1042

70,1019,1320,1160

75,1230,1608,1396

80,1376,1780,1590

85,1570,1900,1782

90,1775,2134,2002

95,2062,2519,2267

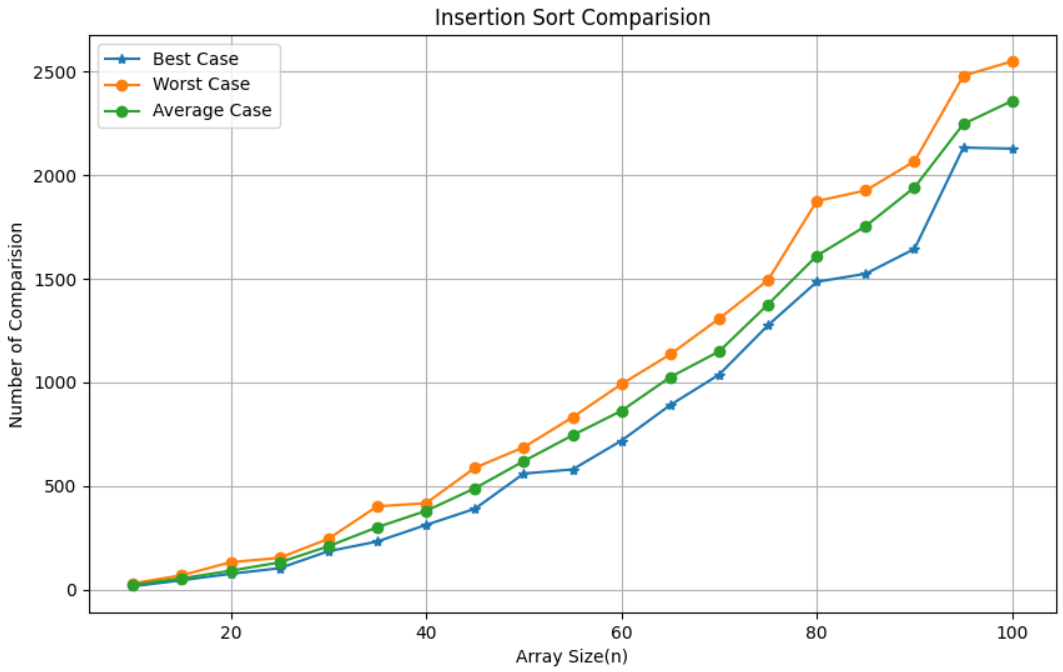
100,2221,2804,2407

Python Code to plot the graph:-

```
import pandas as pd

import matplotlib.pyplot as plt

df = pd.read_csv("search_comparisons.csv")

size = df["DataSize"]

bestCase = df["BestCase"]

worstCase = df["WorstCase"]

averageCase = df["AverageCase"]

# //plotting the graph
```

```
plt.figure(figsize=(10,6))

plt.plot(size,bestCase,label="Best Case",marker="*")

plt.plot(size, worstCase, label="Worst Case", marker='o')

plt.plot(size, averageCase, label="Average Case", marker='o')

plt.title("Insertion Sort Comparision")

plt.xlabel("Array Size(n)")

plt.ylabel("Number of Comparision")

plt.legend()

plt.grid(True)

plt.savefig("diagram.png")

plt.show()
```

OUTPUT GRAPH:



//Program 2: Insertion sort applying on all the permutation of a randomly generated array and finding it's best , worst and average time complexity

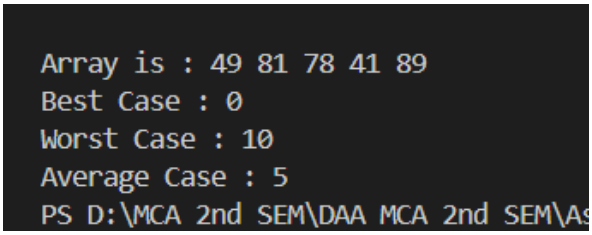
```
#include <iostream>

#include <vector>
```

```
#include <cstdlib>
#include <algorithm> //for rnad and srand
#include <ctime>
#include <fstream>
using namespace std;
// Insertion sort function
int insertionSort(vector<int> arr, int size){
   int comparision = 0;
   int key;
   int j;
   for (int i = 1; i < size; i++)
   {
      key = arr[i];
      j = i - 1;
      while (j >= 0 && key < arr[j])
      {
         comparision++;
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }
   return comparision;
}
// to print vector
void print(vector<int> vec){
   cout<<endl;
   cout<<"Array is : ";
   for(int i=0;i<vec.size();i++){
      cout<<vec[i]<<" ";
   }
   cout<<endl;
}
// function to swap two numbers in a vector
void swap(vector<int>& vec,int start,int end){
```

```
    int temp = vec[start];
    vec[start] = vec[end];
    vec[end] = temp;
}
//Function to find permutation of the given vector vec
void permute(vector<int> vec,vector<vector<int>>& ans,int index){
    if(index>= vec.size()){
        ans.push_back(vec);
        return;
    }
    for(int i=index;i<vec.size();i++){
        swap(vec,index,i);
        permute(vec,ans,index+1);
        swap(vec,index,i);
    }
}
int fact(int n){
    if(n<=1)return 1;
    return n*fact(n-1);
}
// Main function
int main(){
    srand(time(0));
    int size=5;
    vector<int> vec;

    for(int i=0;i<size;i++){
        int random_number = rand()%100;
        vec.push_back(random_number);
    }
    print(vec);

    vector<vector<int>> pVector;
    int index=0;
    permute(vec,pVector,index);
```

```
    int bestCase = INT16_MAX;
    int worstCase = INT16_MIN;
    int averageCase = 0;
    for(int i=0;i<fact(size);i++){
        //sorting the data and taking the number of comparision as output
        int compare = insertionSort(pVector[i],size);
        bestCase = bestCase > compare ? compare : bestCase;
        worstCase = worstCase > compare ? worstCase : compare;
        averageCase += compare;
    }
    // output (time complexity)
    cout<<"Best Case : "<<bestCase<<endl;
    cout<<"Worst Case : "<<worstCase<<endl;
    cout<<"Average Case : "<<averageCase/fact(size)<<endl;
    return 0;
}
```

Output :



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 <cstdlib> // For rand() and srand()
#include <ctime>   // For time()
using namespace std;

// Structure to represent weather data
struct WeatherData {
   string city;
   string timeStamp;
   double temperature;
   double humidity;
   double rainfall;
};
// Function to perform Insertion Sort
void insertionSort(vector<WeatherData>& data) {
   for (size_t i = 1; i < data.size(); ++i) {
      WeatherData key = data[i];
      int j = i - 1;

      // Move elements of data[0..i-1] that are greater than key.city to one position ahead of their current position
      while (j >= 0 && data[j].city > key.city) {
         data[j + 1] = data[j];
         j = j - 1;
      }
      data[j + 1] = key;
   }
}

// Function to generate random weather data
vector<WeatherData> generateWeatherData(int numSamples) {
   vector<string> cities = {"Delhi", "Bangalore", "Mumbai", "UP", "Goa"};
   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;
```

```
    srand(time(0)); // Seed the random number generator

    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); // Random temperature between 20 and 40
        record.humidity = 30 + (rand() % 70);    // Random humidity between 30% and 100%
        record.rainfall = (rand() % 50);         // Random rainfall between 0 and 50 mm
        data.push_back(record);
    }

    return data;
}

// Function to print the weather data
void printWeatherData(const vector<WeatherData>& data) {
    cout << setw(5) << "S.No." << setw(15) << "City" << setw(15) << "Time Stamp"<< setw(15) << "Temp. (C)" << setw(15) << "Humidity (%)" << setw(15) << "Rainfall (mm)" << endl;
    // if we write data.size() instead of 15 it will print all the data
    for (size_t i = 0; i < 15; ++i) {
        cout << setw(5) << i + 1 << setw(15) << data[i].city << setw(15) << data[i].timeStamp<< setw(15) << data[i].temperature << setw(15) << data[i].humidity << setw(15) << data[i].rainfall << endl;
    }
}

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

    // Input data sorted by time
    cout << "Input Data (Unsorted) first 15 data only:" << endl;
    printWeatherData(weatherData);
    insertionSort(weatherData);

    // Output data sorted by city
```

```
    cout << "\nOutput Data (Sorted by City) first 15 data only:" << endl;
    printWeatherData(weatherData);

    return 0;
}
```

Output :

```
Input Data (Unsorted) first 15 data only:
S.No.          City     Time Stamp      Temp. (C)   Humidity (%)  Rainfall (mm)
    1      Bangalore        7:00 AM             25             64             34
    2      Bangalore        2:00 PM             39             62             30
    3          Delhi       11:00 AM             32             54             12
    4             UP       11:00 AM             25             33              0
    5         Mumbai       11:00 AM             21             46             10
    6      Bangalore       11:00 AM             27             44             14
    7         Mumbai        2:00 PM             27             57             28
    8            Goa        8:00 AM             29             45              0
    9            Goa        7:00 AM             33             77             47
   10             UP        6:00 AM             22             69             32
   11      Bangalore       10:00 AM             22             86             39
   12      Bangalore        5:00 AM             28             47             13
   13         Mumbai        5:00 AM             24             42              8
   14         Mumbai        2:00 PM             25             46             40
   15            Goa        7:00 AM             25             36             13

Output Data (Sorted by City) first 15 data only:
S.No.          City     Time Stamp      Temp. (C)   Humidity (%)  Rainfall (mm)
    1      Bangalore        7:00 AM             25             64             34
    2      Bangalore        2:00 PM             39             62             30
    3      Bangalore       11:00 AM             27             44             14
    4      Bangalore       10:00 AM             22             86             39
    5      Bangalore        5:00 AM             28             47             13
    6      Bangalore        8:00 AM             25             42             19
    7      Bangalore        5:00 AM             34             97             41
    8      Bangalore        1:00 PM             30             58             31
    9      Bangalore       10:00 AM             36             37              5
   10      Bangalore        2:00 PM             33             48             23
   11      Bangalore       10:00 AM             28             39              9
   12      Bangalore        7:00 AM             34             38             43
   13      Bangalore       12:00 PM             26             97             27
   14      Bangalore       10:00 AM             22             48              9
   15      Bangalore       11:00 AM             34             79             47
```

Conclusion : Insertion Sort maintains the relative order of elements with equal city values because it only shifts elements when it finds a smaller value. Since it doesn't change the order of equal elements, Insertion Sort is stable.