Alperen UGUS
CWID: 10864101

Algorithms HW – 1

Q1. Implementation of Algorithms:

The first thing while implementing both of the algorithms was to read coordinates from the files and store them in a dynamic array. Then in order not to calculate every time, I have created a 2D array which holds the distances among the points.

While implementing the nearest point algorithm, I created an array just holding Booleans for the points. The points that are not visited have a "false" value at the same index in this array. It is important not to visit the same point twice except from the first one. Then for each point, I found the nearest one, changed its visited flag to "true", added the distance to the total sum.

For the exhaustive algorithm, I just needed all the possible paths in order to calculate the total distance for all those paths. I searched the internet for a permutation algorithm and found out the one that I have used in my code. (See: https://www.geeksforgeeks.org/stdnext_permutation-prev_permutation-c/) Then, for all paths, I calculated the distances, found out the least one and printed out as the absolute result.

Q2. Worst Case Complexities:

Nearest Neighbor:

1. Computing the distance between two points is just a constant.
2. Computing the distance among all is also giving (n).
3. For each point, I had to find the nearest neighbor (n).

Therefore, it is O(n^2).

Exhaustive:

1. Computing the distance between two points is just a constant.
2. Computing all permutations and distances among them: O(n!).

Q3.
Nearest Neighbor:

run / n	2	4	6	8
First	183 µs	176 µs	199 µs	297 µs
Second	123 µs	127 µs	131 µs	136 µs
Third	126 µs	128 µs	135 µs	135 µs
Average	144 µs	143 µs	155 µs	189 µs

Alperen UGUS
CWID: 10864101

Exhaustive:

run / n	2	4	6	8
First	173 µs	207 µs	219 µs	1190 µs
Second	127 µs	115 µs	151 µs	1217 µs
Third	128 µs	131 µs	152 µs	1215 µs
Average	142 µs	151 µs	174 µs	1207 µs

Q4.

It is obvious that the exhaustive algorithm takes more time with each additional point. It radically made a peak from 6 to 8. O(n!) is the fundamental reason for this because factorial numbers grow radically. And it is also obvious that nearest neighbor is proportionally takes time with respect to n. This is also the proof of O(n).

Alperen UGUS
CWID: 10864101

TSP – NEAREST NEIGHBOR

```
//
//  main.cpp
//  TSP
//
//  Created by Alperen on 9/8/19.
//  Copyright © 2019 Alperen. All rights reserved.
//

#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include <cstdlib>


using namespace std;
using namespace std::chrono;


int main(int argc, const char * argv[]) {

    auto start = high_resolution_clock::now();

    int pointNum;
    ifstream in;
    in.open("/Users/alpi/Desktop/C++/TSP/TSP/1.txt");
    in >> pointNum;
    //    cout << pointNum << endl;

    //    Dynamic array for coordinate points
    int **unvisited = new int *[pointNum];
    for (int i = 0; i < pointNum; i++) {
        unvisited[i] = new int[2];
    }

    //    Read coordinates from the file and insert into the array
    for(int i = 0; i < pointNum; i++){
        in >> unvisited[i][0];
        in >> unvisited[i][1];
    }

//        Printing the coordinates in order to check them
//        for(int i = 0; i < pointNum; i++){
//            cout << unvisited[i][0] << ' ' << unvisited[i][1] <<
endl;
//        }
```

Alperen UGUS
CWID: 10864101

```
    //    Initializing 2D dynamic distances array
    int **distances = new int *[pointNum];

    for (int i = 0; i < pointNum; i++) {
        distances[i] = new int[1];
    }

    //    Find out all distances
    for(int i = 0; i < pointNum; i++){
        for(int j = 0; j < pointNum; j++){
            distances[i][j] = abs(unvisited[i][0] - unvisited[j][0]) +
abs(unvisited[i][1] - unvisited[j][1]);
        }
    }

    //    Printing all distances
    //    for(int i = 0; i < pointNum; i++){
    //        for(int j = 0; j < pointNum; j++){
    //            cout << distances[i][j] << ' ';
    //        }
    //        cout << endl;
    //    }

    //    Boolean array to hold visited flags
    bool *gone = new bool[pointNum];

        for(int i = 0; i< pointNum ; i++){
            gone[i] = false;
//            cout << gone[i];
        }
//        cout << endl;

    int result = 0;
    int current = 0;
    int counter = 1;

    while (true) {
        gone[current] = true;
        int min = 99999;
        int min_index = 0;

        //        Search for the nearest neighbor
        for(int j = 0; j < pointNum; j++){
            if (distances[current][j] < min && gone[j] == false) {
                min = distances[current][j];
                min_index = j;
            }
        }
```

Alperen UGUS
CWID: 10864101

```
        //        Add to the result and change the current point to
the arrived one
        result += min;
        current = min_index;

        //        cout << min << endl;
        //        cout << current << endl;
        counter++;
        //        If all points are done, return to the beginning
        if(counter == pointNum){
            result += distances[current][0];
            break;
        }

    }


    cout << "Result : " << result << endl;

    auto stop = high_resolution_clock::now();

    auto duration = duration_cast<microseconds>(stop - start);

    cout << "Time taken by function: " << duration.count() << "
microseconds" << endl;


    return 0;
}
```

Alperen UGUS
CWID: 10864101

TSP - EXHAUSTIVE

```
//
//  main.cpp
//  TSP
//
//  Created by Alperen on 9/8/19.
//  Copyright © 2019 Alperen. All rights reserved.
//

#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include <cstdlib>
#include <algorithm>


using namespace std;
using namespace std::chrono;

int factorial(int n)
{
    if(n > 1)
        return n * factorial(n - 1);
    else
        return 1;
}


int main(int argc, const char * argv[]) {

    auto start = high_resolution_clock::now();

    int pointNum;
    ifstream in;
    in.open("/Users/alpi/Desktop/C++/TSP/TSP/4.txt");
    in >> pointNum;
    //    cout << pointNum << endl;

    //    Dynamic array for coordinate points
    int **unvisited = new int *[pointNum];
    for (int i = 0; i < pointNum; i++) {
        unvisited[i] = new int[2];
    }
```

Alperen UGUS
CWID: 10864101

```
    //    Read coordinates from the file and insert into the
array
    for(int i = 0; i < pointNum; i++){
        in >> unvisited[i][0];
        in >> unvisited[i][1];
    }

    //    Printing the coordinates in order to check them
//    for(int i = 0; i < pointNum; i++){
//        cout << unvisited[i][0] << ' ' << unvisited[i][1] <<
endl;
//    }

    //    Initializing 2D dynamic distances array
    int **distances = new int *[pointNum];

    for (int i = 0; i < pointNum; i++) {
        distances[i] = new int[1];
    }

    //    Find out all distances
    for(int i = 0; i < pointNum; i++){
        for(int j = 0; j < pointNum; j++){
            distances[i][j] = abs(unvisited[i][0] -
unvisited[j][0]) + abs(unvisited[i][1] - unvisited[j][1]);
        }
    }

    //    Printing all distances
//    for(int i = 0; i < pointNum; i++){
//        for(int j = 0; j < pointNum; j++){
//            cout << distances[i][j] << ' ';
//        }
//        cout << endl;
//    }

    //    Generation of natural numbers up to n - 1 that we use
for different paths
    int *numbers = new int[pointNum - 1];
    for(int i = 0; i < pointNum - 1; i ++){
        numbers[i] = i + 1;
        //        cout << numbers[i];
    }
```

Alperen UGUS
CWID: 10864101

```
    //    Number of all possible paths
    int permutations = factorial(pointNum - 1);

    int **paths = new int *[permutations];
    for (int i = 0; i < permutations; i++) {
        paths[i] = new int[pointNum + 1];
    }

    //    Start and End points would be the initial one
    for(int i = 0; i < permutations; i++){
        paths[i][0] = 0;
        paths[i][pointNum] = 0;
    }


    int counter = 0;

    //    Generation of all possible permutations
    sort (numbers, numbers + pointNum - 1);
//    cout << "The 3! possible permutations with 3 elements:\n";
    do {
        for(int i = 0; i < pointNum - 1; i++){
            paths[counter][i+1] = numbers[i];
        }
        counter++;
    } while ( std::next_permutation(numbers,numbers + pointNum -
1) );

//    for(int i = 0; i < permutations; i++){
//        for(int j = 0; j < pointNum + 1; j++){
//            cout << paths[i][j];
//        }
//        cout << endl;
//    }


    int *perm_dist = new int[permutations];

    for (int i = 0; i < permutations; i++) {
        perm_dist[i] = 0;
    }
    //    Calculate all the distances of all permutations
    for(int i = 0; i < permutations; i++){

        for(int j = 0; j < pointNum; j++){
```

Alperen UGUS
CWID: 10864101

```
            perm_dist[i] += distances[paths[i][j]][paths[i][j +
1]];
        }
    }



//    for(int j = 0; j < permutations; j++){
//        cout << perm_dist[j] << endl;
//    }

    int min = perm_dist[0];
    int min_path_index = 0;

    for(int i = 0; i < permutations; i++){
        if(perm_dist[i] < min){
            min = perm_dist[i];
            min_path_index = i;
        }
    }

//    cout << min_path_index << endl;
    cout << "Result : " << perm_dist[min_path_index] << endl;

    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);


    delete[] unvisited;
    delete[] distances;
    delete[] numbers;
    delete[] paths;
    delete[] perm_dist;

    cout << "Time taken by function: "
    << duration.count() << " microseconds" << endl;



    return 0;
}
```