Report

Name: Haoxuan Yang

1.

Nearest-neighbor heuristic approach:

I implement my function in a class object called "Solution". My function is called "TSP" having two arguments: int "n" and 2-D int array "a".

I use vector<int> to store my results (order of the indices of points) and unvisited points. Some boundary conditions such as when n = 0 and when n = 1 should also be considered. I name the input points from 0 to n. i.e., if there are 4 points as shown in the question, I will index them from 0 to 3 and store the (x,y) values of points in a 2-D int array "a". Here are the variables I have declared followed by their definitions in this method:

vector<int> result: it stores the indices of the final orders of the points visited.

vector<int> re: it stores the returned value of vector<int> result variable from the function TSP.

vector<int> unvisited: it stores the indices of the unvisited points.

float total: it stores the total travelled distance.

int cur: it stores the index of the current point.

float distance[]: it is a float array to store all the possible distance from the current point to other unvisited points.

float smallest: it stores the smallest distance from the current point to the other unvisited points.

Code explanation: I store all the unvisited points except the starting point "0" as unvisited points. There is a for loop and in each loop we have to make the choice of the next point based on the nearest-neighbor rule given. Temporary distance array to store the distance from current point to all the possible unvisited points. Thus, we can compare them and select the smallest one to be our next point. Then we need to set "cur" variable to the selected point. Then we add this smallest distance to our total length which means we have already travelled. Then we go to the next loop which is doing the same thing. When we reach the final loop, we find that there is only one value left in the unvisited array, so we need to add the distance from this final point to our starting point. All the results are printed in the console.

Exhaustive approach:

I implement my functions in a class object called "Solution". My functions are called "TSP" and "Perm". "TSP" is the first function which will use the second function "Perm". "TSP" has two arguments: int "n" and 2-D int array "a". "Perm" function has 4 arguments: start (int), end (int), unvisited array and 2-D int array "a" containing all the points.

Some boundary conditions such as when n = 0 and when n = 1 should also be considered. I name the input points from 0 to n. i.e., if there are 4 points as shown in the question, I will index them from 0 to 3 and store the (x,y) values of points in a 2-D int array "a". Here are the variables I have declared followed by their definitions in this method:

int order[]: it stores the indices of the unvisited points. We use this to generate all the possible permutations.

float distance: it is a float to store the total distance. It will be printed out and cleared after we get the end of one permutation.

int start: starting point in each permutation.

int end: end point in each permutation.

Code explanation: I store all the unvisited points except the starting point "0" as unvisited points. This problem can be converted to a permutation problem using recursive approach. So we can figure out all the possible permutations and then we can calculate their distance one by one on the way. "swap" function has been used in the process of permutation because we need to exchange the starting point with other unvisited points before we go into the next permutation. We can see all the solution of this problem in a tree structure, so we still need to exchange it back after permutation to get back to the parent solution. When we find start is equal to end in a permutation, that means we finally reach to the end of this permutation and the we can calculate the distance in this order which is stored in "order". After computing all the distances from all the possible solutions, we can get the best solution based on the results. All the results are printed in the console.

2.

For the first method, I use nested loop to iteratively find the next point until we get to the end. And the outer loop takes n-1 steps and there are two inner loops (one followed by another) both take n-1 steps. So the total worst-case time complexity is O(n^2) in this method.

For the second method, I use nested functions to solve this problem. The outer function has a for loop taking n steps but the inner function (it is outside the loop) takes n! runtime at the worst-case time complexity. So we can assume that this method takes O(n!) worst-case time complexity.

3.

	n=4	n=6	n=8	n=10
Nearest-neighbor	0.01	0.012	0.023	0.093
Exhaustive	0.002	0.005	0.263	19.112

The numbers in this table are in msec. I choose n from n = 4 to 10 because if I choose n>10, the second method will run for too long time. 11! = 39 916 800 and that just takes me forever to run this. When n = 4, the runtime is definitely 10 times larger than my clock function.

4.

When n is very small such as 4, the time complexity does not have much impact on our actual runtime because in the worst-case time, 4^2 = 16 and 4! = 24. That's just 8 times of operations difference. Also, I have more for loops and steps of operations in my nearest-neighbor method which just makes this runtime more than exhaustive method. As n increases, the impact of the time complexity will have a big difference on our actual runtime which has been shown in the table above. The experimental runtime is consistent with the theoretical runtime derived.

Code:

```
#include <iostream>

using namespace std;

#include <vector>

#include <cmath>

#include <algorithm>

#include <cstdlib>    // For srand() and rand()

#include <ctime>


class Solution{

    public:

        vector<int> TSP(int n,int a[][2]){

            vector<int> result;// store order of route

            if(n == 0){

                return result;

            }

            result.push_back(0);// starting point is always 0

            if(n == 1){

                return result;

            }

            float total = 0;// store total distance

            int cur = 0;// 0 to 3

            // unvisited points, elements decreasing

            vector<int> unvisited;// value from 0 to 3
```

```
        for(int i = 1; i < n; i++){
            unvisited.push_back(i);
        }
        for(int j = 0; j < n-1; j++) {
            // check unvisited
        //      for(int i = 0; i < unvisited.size(); i++){
        //          cout << unvisited.at(i);
        //      }
        //      cout << endl;
            // if only 1 unvisited, than add distance
            // from the last one to the starting point
            if(unvisited.size() == 1){
                total += sqrt(pow(a[unvisited[0]][0]-a[0][0],2)+
                pow(a[unvisited[0]][1]-a[0][1],2));
            }
            // compute distance, elements decresing
            float distance[unvisited.size()] = {};
            for(int i = 0; i < unvisited.size(); i++){
                distance[i] =
sqrt(pow(a[unvisited[i]][0]-a[cur][0],2)+pow(a[unvisited[i]][1]-a[cur][1],2));
            }
        //      // check distance
        //      for(int i = 0; i < unvisited.size(); i++){
        //          cout << distance[i] << endl;
        //      }
            // choose an unvisited point based on the smallest distance
            float smallest;
            for(int i = 0; i < unvisited.size(); i++){
                if(i == 0){
```

```
                        smallest = distance[i];
                        cur = unvisited[i];
                    }else if(smallest > distance[i]){
                        smallest = distance[i];
                        cur = unvisited[i];
                    }
                }
            //      cout << smallest << "haha" << cur << endl;
                // add order
                result.push_back(cur);
                // add distance
                total += smallest;
                // delete point number from the unvisited array
                vector<int>::iterator iter=find(unvisited.begin(),unvisited.end(),cur);
                if(iter!=unvisited.end()) unvisited.erase(iter);
                // check unvisited
            //      for(int i = 0; i < unvisited.size(); i++){
            //              cout << unvisited.at(i);
            //      }
            //      cout << endl;
                // cout << smallest << endl;
            }
            cout << "distance: " << total << endl;
            return result;
        }
};
int main ()
{
    // int n = 4;
```

```
// int a[n][2] = {{3,4},{6,8},{50,8},{11,7}};
// int a[3][2] = {{0,0},{5,5},{20,17}};
srand(time(0));    // Initialize random number generator
vector<int> re;
Solution b;
for(int j = 4; j <= 4; j+=2){
        int n = j;
        int a[n][2];
        for(int k = 0; k < n; k++){
                for(int t = 0; t < 2; t++){
                        a[k][t] = (rand() % 100) + 1;
                        // if(t == 1){
                        //          cout << "(" << a[k][0] << "," << a[k][1] << ")";
                        // }
                }
        }
        // cout << endl;
        double start,stop;
        start = clock();
        re = b.TSP(n,a);
        stop = clock();
        cout << "runtime is: " << (stop-start)/double(CLOCKS_PER_SEC)*1000 << endl;
        cout << "order: ";
        for(int i = 0; i < re.size(); i++){
                  cout << " " << re.at(i);
        }
        cout << endl;
}
return 0;
```

```
}

Exhaustive:
#include <iostream>
using namespace std;
#include <vector>
#include <cmath>
#include <algorithm>
#include <cstdlib>    // For srand() and rand()
#include <ctime>
#include "limits.h"

float result_dis = INT_MAX;

class Solution{
        public:
        void TSP(int n,int a[][2], int result_order[]){
                if(n == 0){
                        return;
                }
                if(n == 1){
                        cout << "order: " << "0" << endl;
                        cout << "distance: " << "0" << endl;
                        return;
                }
                int order[n];
                for(int i = 0; i < n; i++){
                        order[i] = i;
                }
```

```
        Perm(1, n, order, a, result_order);
}
void Perm(int start, int end, int order[], int a[][2], int result_order[]){
        // get to the end, output one permutation
        if(start == end){
                float distance = 0;
                int cur = 0;
                // cout << "order: ";
                for(int i = 0; i < end; i++){
                        // cout << order[i];
                        if(i > 0){
                                distance += sqrt(pow(a[order[i]][0]-a[cur][0],2)
                                +pow(a[order[i]][1]-a[cur][1],2));
                                cur = order[i];
                        }
                }
                // cout << endl;
                // add distance from the last location to the starting point(0)
                distance += sqrt(pow(a[0][0]-a[cur][0],2)
                                +pow(a[0][1]-a[cur][1],2));
                if(result_dis > distance){
                        result_dis = distance;
                        for(int i = 0; i < end; i++){
                                result_order[i] = order[i];
                        }
                }
                // cout << "distance: " << distance << endl;
                return;
        }
```

```
            // permutation
            for(int i = start; i < end; i++){
                swap(order[start],order[i]);
                Perm(start+1,end,order,a,result_order);
                swap(order[i],order[start]);
            }
        }
};

int main()
{
        // int n = 5;
        // int a[n][2] = {{0,0},{3,5},{6,11},{30,1},{8,1}};
        srand(time(0));    // Initialize random number generator
        Solution b;
        for(int j = 4; j <= 4; j+=2){
            int n = j;
            int result_order[n];
            int a[n][2];
            for(int k = 0; k < n; k++){
                for(int t = 0; t < 2; t++){
                    a[k][t] = (rand() % 100) + 1;
                    // if(t == 1){
                    //          cout << "(" << a[k][0] << "," << a[k][1] << ")";
                    // }
                }
            }
            // cout << endl;
            double start,stop;
```

```
            start = clock();
            b.TSP(n,a,result_order);
            stop = clock();
            cout << "order: ";
            for(int p = 0; p < n; p++){
                cout << result_order[p] << " ";
            }
            cout << endl;
            cout << "distance: " << result_dis << endl;
            cout << "runtime is: " << (stop-start)/double(CLOCKS_PER_SEC)*1000 << endl;
        }
        return 0;
}
```