STUDENT: **PHUC LE**

PROGRAMMING ASSIGNMENT #3

EUCLIDEAN TRAVELING SALESPERSON PROBLEM

ALGORITHM ENGINEERING **335.01**

The Exhaustive Optimization Algorithm

```
// Start the chronograph to time the execution of the algorithm

// calculate the farthest pair of vertices
Dist = farthest(n, P);
bestDist = n*Dist;
```

$(12n^2 + 12n + 1) + 1$ (calculated below) (*)

2

```
// populate the starting array for the permutation algorithm
A = new int[n];
```

1

```
// populate the array A with the values in the range 0 .. n-1
for (i = 0; i<n; i++)
    A[i] = i;
```

$\} \; n + 1$

```
// calculate the Hamiltonian cycle of minimum weight
print_perm(n, A, n, P, bestSet, bestDist);
```

$n.n! + n! + 7n + 2$ (**)

(calculated below)

```
// End the chronograph to time the loop
```

Total Running Time of Euclidean Traveling Salesperson (Exhaustive search algorithm)

$= n.n! + n! + 12n^2 + 20n + 7$

The Exhaustive Optimization Algorithm (continued)

```
void farthest_point(int n, point2D *P){
// function to calculate the furthest distance between any two 2D points
    int i, j, farthestPoint;
    float dist;
    for (i = 0; i < n - 1; i++)          n
        for (j = 0; j < n; j++) {        n + 1
            dist = sqrt((P[i].x - P[j].x)*(P[i].x - P[j].x)
                  + (P[i].y - P[j].y)*(P[i].y - P[j].y));     } 9
            if (max_dist < dist)         1
            {
                max_dist = dist;         1
                farthestPoint = i;       1
            }
        }
    return farthestPoint;                1
}
```

Total: $\underline{12n^2 + 12n + 1}$ (*)

The Exhaustive Optimization Algorithm (continued)

```
void print_perm(int n, int *A, int sizeA, point2D *P, int *bestSet, float &bestDist){
// function to generate the permutation of indices of the list of points
    int i;
    float dist = 0;        1
    if (n == 1){           1
      // we obtain a permutation so we compare it against the current shortest Hamiltonian cycle
      int j = 1;           1
      for (int i = 0; i < sizeA; ++i, ++j){   (sizeA + 1)
         if (j == sizeA)    1
            j = 0;          1
         dist += sqrt ( (P[A[i]].x - P[A[j]].x)*(P[A[i]].x - P[A[j]].x) +
                    (P[A[i]].y - P[A[j]].y)*(P[A[i]].y - P[A[j]].y));     } 9
      }
      if (dist < bestDist){      1
         for (int i = 0; i < sizeA; ++i){    (sizeA + 1)
            bestDist = dist;     1
            bestSet[i] = A[i];   1
         }
      }
    }

    else{
      for (i = 0; i< n - 1; i++){     n
         print_perm(n - 1, A, sizeA, P, bestSet, bestDist);     n!
         if (n % 2 == 0){          2
            // swap(A[i], A[n-1])
            int temp = A[i];        1
            A[i] = A[n - 1];        2
            A[n - 1] = temp;        2
         }
         else{
            // swap(A[0], A[n-1])
            int temp = A[0];        1
            A[0] = A[n - 1];        2
            A[n - 1] = temp;        2
         }
      }
      print_perm(n - 1, A, sizeA, P, bestSet, bestDist); n!
    }
}
```

Total: $nn! + n! + 7n + 2$ (**)

The Exhaustive Optimization Algorithm (continued)

$$f(n) = n.n! + n! + 12n^2 + 20n + 7$$

$$f(n) \in O(n.n!)$$

* Prove time complexity using Limit:

$$L = \lim_{n \to \infty} \frac{n.n! + n! + 12n^2 + 20n + 7}{n.n!} = 1$$

Because L is a non-negative constant, the relationship $f(n) \in O(n.n!)$ is TRUE

* Prove time complexity using definition

$$n.n! + n! + 12n^2 + 20n + 7 \leq c.n.n! \qquad c = ? \quad n_0 = ?$$

divide both sides by $n.n!$

$$1 + \frac{1}{n} + \frac{12n}{(n-1)!} + \frac{20}{n!} + \frac{7}{n.n!} \leq c$$

$$\downarrow_{n\to\infty} \quad \downarrow_{n\to\infty} \quad \downarrow_{n\to\infty} \quad \downarrow_{n\to\infty} \quad \downarrow_{n\to\infty}$$

$$1 \quad 0 \quad 0 \quad 0 \quad 0 \leq c \text{ is TRUE}$$

Improved Nearest Neighbor Algorithm

```
// Start the chronograph to time the execution of the algorithm

// allocate space for the Visited array of Boolean values
Visited = new bool[n];          1
// set it all to False
for (i = 0; i<n; i++)           n + 1
    Visited[i] = false;         1

// calculate the starting vertex A
A = farthest_point(n, P); (12n^2 + 12n + 1) + 1   (calculated below)(***)
i = 0;                          1
M[i] = A;                       1

// set it as visited
Visited[A] = true;              1

for (i = 1; i<n; i++) {                  n
    // calculate the nearest unvisited neighbor from node A
    B = nearest(n, P, A, Visited); (13n + 15) + 1   (calculated below)(****)
    // node B becomes the new node A
    A = B;                               1
    // add it to the path
    M[i] = A;                            1
    Visited[A] = true;                   1
}

// calculate the length of the Hamiltonian cycle
dist = 0;                                      1
for (i = 0; i < n - 1; i++) {                  n
    dist += sqrt((P[M[i]].x - P[M[i+1]].x)*(P[M[i]].x - P[M[i+1]].x) +
        (P[M[i]].y - P[M[i+1]].y)*(P[M[i]].y - P[M[i+1]].y));       } 9
}
dist += sqrt((P[M[0]].x - P[M[n-1]].x)*(P[M[0]].x - P[M[n-1]].x) +
    (P[M[0]].y - P[M[n-1]].y)*(P[M[0]].y - P[M[n-1]].y));           } 9

// End the chronograph to time the loop
```

Total Running Time of Improved Nearest Neighbor Algorithm

$$= 25n^2 + 41n + 17$$

Improved Nearest Neighbor Algorithm (continued)

```
int farthest_point(int n, point2D *P){
// function to calculate the furthest distance between any two 2D points
    int i, j, farthestPoint;
    float dist;

    for (i = 0; i < n - 1; i++)                                    n
        for (j = 0; j < n; j++) {                                  n + 1
            dist = sqrt((P[i].x - P[j].x)*(P[i].x - P[j].x)        } 9
                      + (P[i].y - P[j].y)*(P[i].y - P[j].y));      }
            if (max_dist < dist){                                  1
                max_dist = dist;                                   1
                farthestPoint = i;                                 1
            }
        }
    return farthestPoint;                                          1
}
```

Total: $12n^2 + 12n + 1$ (***)

```
int nearest_point(int n, point2D *P, int A, bool *Visited) {
// function to calculate the nearest unvisited neighboring point
    float nearestDist = max_dist;                                  1
    int i, nearestPoint;
    float dist;

    for (i = 0; i < n; ++i){                                       n + 1
        if (Visited[i] == false){                                  1
            dist = sqrt((P[A].x - P[i].x)*(P[A].x - P[i].x)        } 9
                      + (P[A].y - P[i].y)*(P[A].y - P[i].y));      }
            if (dist < nearestDist){                               1
                nearestDist = dist;                                1
                nearestPoint = i;                                  1
            }
        }
    }
    return nearestPoint;                                           1
}
```

Total: $13n + 15$ (****)

Improved Nearest Neighbor Algorithm (continued)

$f(n) = 25n^2 + 41n + 17$

$f(n) \in O(n^2)$

* Prove time complexity using Limit

$L = \lim_{n\to\infty} \frac{25n^2 + 41n + 17}{n^2} = 25$, a non-negative constant; thus, the relationship is TRUE

* Prove time complexity using definition:

$25n^2 + 41n + 17 \leqslant c.n^2$

divide both sides by n^2

$$25 + \frac{41}{n} + \frac{17}{n^2} \leqslant c$$

$$\downarrow_{n\to\infty} \quad \downarrow_{n\to\infty} \quad \downarrow_{n\to\infty}$$

$25 \quad 0 \quad 0 \leqslant c$ is TRUE

<u>Conclusion</u>: with the same input (10 points), both algorithm have given the same output. However, the improved nearest neighbor algorithm runs much faster than exhaustive search one for the same problem.

HOW TO RUN THE PROGRAMS:

- There are 2 executable files inside the folder "**Executable Files**"
- Or they can be run through opening project using Microsoft Visual Studio

THE OUTPUT SAMPLES:

Example 1A:

```
CPSC 335-01 - Programming Assignment #3
Euclidean traveling salesperson problem: exhaustive optimization algorithm
Enter the number of vertices (>2)
4
Enter the points; make sure that they are distinct
x=2
y=0
x=1
y=1
x=3
y=1
x=0.1
y=0
The Hamiltonian cycle of the minimum length
(2,0) (3,1) (1,1) (0.1,0) (2,0)
Minimum length is 6.65958
elapsed time: 1.4e-05 seconds
Press any key to continue . . .
```

Example 1B:

```
CPSC 335-01 - Programming Assignment #3
Euclidean traveling salesperson problem: exhaustive optimization algorithm
Enter the number of vertices (>2)
8
Enter the points; make sure that they are distinct
x=0
y=4
x=2
y=1
x=1
y=6
x=2
y=7
x=3
y=5
x=3
y=2
x=5
y=2
x=6
y=5
The Hamiltonian cycle of the minimum length
(3,2) (5,2) (6,5) (3,5) (2,7) (1,6) (0,4) (2,1) (3,2)
Minimum length is 19.0684
elapsed time: 0.021927 seconds
Press any key to continue . . .
```

Example 2A:

```
CPSC 335-01 - Programming Assignment #3
Euclidean traveling salesperson problem: improved nearest neighbor algorithm
Enter the number of vertices (>2)
4
Enter the points; make sure that they are distinct
x=2
y=0
x=1
y=1
x=3
y=1
x=0.1
y=0
The Hamiltonian cycle of a relative minimum length
(3,1) (2,0) (1,1) (0.1,0) (3,1)
The relative minimum length is 7.24136
elapsed time: 7e-06 seconds
Press any key to continue . . .
```

Example 2B:

```
CPSC 335-01 - Programming Assignment #3
Euclidean traveling salesperson problem: improved nearest neighbor algorithm
Enter the number of vertices (>2)
8
Enter the points; make sure that they are distinct
x=0
y=4
x=2
y=1
x=1
y=6
x=2
y=7
x=3
y=5
x=3
y=2
x=5
y=2
x=6
y=5
The Hamiltonian cycle of a relative minimum length
(0,4) (1,6) (2,7) (3,5) (3,2) (2,1) (5,2) (6,5) (0,4)
The relative minimum length is 22.7079
elapsed time: 1e-05 seconds
Press any key to continue . . .
```