THOMAS NGO
VAN DO

EXHAUSTIVE OPTIMIZATION ALGORITHM (EOA)

Pseudocode:
//INPUT:a positive integer n and a list P of n distinct points representing vertices of a rectilinear graph
//OUTPUT: a list of n points from P representing a Hamiltonian cycle of minimum total weight for the graph

```
    dist                          //distance of Hamiltonian path
    P                             //array P contains vertices (x,y)
    A                             /temporary array to keep indices for permutation
    sizeA                         //number of element in array A
    bestSet                       //best solution set for Hamiltonian path
    Dist                          //distance of farthest pair of vertices
starting condition
    Dist ← farthest(n, P)         //farthest function to calculate the furthest distance
                                  //between any two 2D points
    bestSet ← n * Dist
    populate the array A with the values in range 0 .. n-1

starting the algorithm
//calculate the Hamiltonian cycle of minimum weight
    print_perm(n, A, n, bestSet, bestDist)

//function to generate the permutation of indices of the list of points
void print_perm(int n, int *A, int sizeA, point2D *P, int *bestSet, float &bestDist)
    if n=1 do
        dist ← 0                  //initialization for distance of Hamiltonian path

//calculate the distance of Hamiltonian path except the last edge
        for i ← 0 to i<sizeA-1 do
            dist ← dist + (abs(P[A[i]].x – P[A[i+1]].x) + abs(P[A[i]].y – P[A[i+1]].y)

//add the last edge to distance to make a Hamiltonian cycle
        dist ← dist + (abs(P[A[0]].x – P[A[sizeA-1]].x) + abs(P[A[0]].y – P[A[sizeA-1]].y)

        if (dist < bestDist) do
            bestDist ← dist
            copy the element of array A into bestSet array
    else do
        for i ← 0 to i<n-1 do
            print_perm(n-1, A, sizeA, P, bestSet, bestDist)
            if (n%2 == 0) do
                swap A[i] and A[n-1]
            else do
                swap A[0] and A[n-1]

        print_perm(n-1, A, sizeA, bestSet, bestDist)
```

//after shuffling vertices, print out the desired output

    print_cycle(n, P, bestSet)

void print_cycle(int n, point2D *P, int *seq)

    for i ← 0 **to** i<n **do**

        print the point (P[seq[i]].x, P[seq[i]].y)

    print the last point which is also the first point of the cycle i.e (P[seq[0]].x, P[seq[0]].y)

Analysis for EOA:

We have,

$$T(n) = nT(n-1)$$

$$T(1) = n$$

Then,

$$T(n-1) = (n-1)T(n-2)$$
$$T(n-2) = (n-2)T(n-3)$$

Therefore,

$$\begin{aligned} T(n) &= nT(n-1) \\ &= n(n-1)T(n-2) \\ &= n(n-1)(n-2)T(n-3) \\ &= \ldots \\ &= n!\,T(1) \\ &= n*n! \in O(n*n!) \end{aligned}$$

IMPROVED NEAREST NEIGHBOR ALGORITHM (INNA)

Pseudocode:
//INPUT:a positive integer n and a list P of n distinct points representing vertices of a Euclidean graph
//OUTPUT: a list of n points from P representing a Hamiltonian cycle of relatively minimum total weight for the graph

```
    P                                   //array P contains vertices (x,y)
    M                                   //best solution set for Hamiltonian cycle
    Visited                             //boolean array mark visited vertices
    n                                   //number of vertices
    dist                                //distance of Hamiltonian cycle
    A                                   //starting vertex
    B                                   //nearest unvisited neighbor from node A

//calculate the starting vertex
    A ← farthest_point(n, P)

//function to calculate the furthest distance between any two 2D points
int farthest_point(int n, point2D *P)
    float max_dist ← 0                  //max_dist cannot be less than 0
    int i, j, indx;
    float dist;

    for i ← 0 to i<n do
        for j ← 0 to j<n do
            dist ← abs(P[i].x – P[j].x) + abs(P[i].y – P[j].y)
            if max_dist < dist do
                max_dist ← dist
                indx ← i
    return indx

//add A to the path
    i ← 0
    M[i] ← A

//set A as visited
    Visited[A] ← true

for i ← 1 to i<n do
    //caculate the nearest unvisited neighbor from node A
    B ← nearest(n, P, A, Visited)

    //node B becomes the new node A
    A ← B

    //add it to the path
    M[i] ← A
    Visited[A] ← true
```

```
//function to calculate the nearest unvisited neighboring point
int nearest(int n, point2D *P, int A, bool *Visited)
	float min_dist ← 99999^99			//initialize min_dist to be infinite distance
	int i, indx;

	for i ← 0 to i<n do
		if A != i do
			dist ← abs(P[i].x – P[A].x) + abs[P[i].y – P[A].y)
			if min_dist > dist and visited[i] == false do
				min_dist ← dist
				indx ← i
	return indx

//calculate the length of the Hamiltonian cycle
	dist ← 0
	for i ← 0 to i<n-1 do
		dist ← dist + abs(P[M[i]].x - P[M[i+1]].x) + abs(P[M[i]].y – P[M[i+1]].y)
	dist ← dist + abs(P[M[0]].x - P[M[n-1]].x) + abs(P[M[0]].y – P[M[n-1]].y)

//after shuffling them, print the desired output
	print_cycle(n, P, M)

//function to print a cyclic sequence of 2D points in 2D plane
void print_cycle(int n, point2D *P, int *seq)
	for i ← 0 to i<n do
		print (x,y)
	print first point
```

Analysis for INNA:

$$T(n)=\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}1+\sum_{i=0}^{n-1}1+\sum_{i=0}^{n-1}1+\sum_{i=0}^{n-1}1=\sum_{i=0}^{n-1}n+3\sum_{i=0}^{n-1}1=n\sum_{i=0}^{n-1}1+3\sum_{i=0}^{n-1}1=(n+3)\sum_{i=0}^{n-1}1$$

$$= \quad (n+3)n=n^2+3n\in O(n^2)$$

Proof:

$$n^2+3n\leq n^2+3n^2\leq 4n^2\ \forall\ n\geq 1$$

Thus, c = 4 and n_0 = 1.