Exhaustive Pseudocode

```
main():
        point p                                                         //1 step
        int bestSet, A                                                  //2 steps
        int i,n                                                         //2 steps
        bestDist, dist                                                  //2 steps

        output("Enter the number of vertices")      //1 step
        input(n)

        if n < 3:                                                       //1 + max(1,0)
                exit

        P = new points[n]                                               //1 step

        for i in 0 to n:                                    //4 * (n - 0 + 1)
                output("Enter x")
                input(P[i].x)
                output("Enter y")
                input(P[i].y)

        bestSet = new int[n]

        for i = 0 to n:                                             //1*(n-0+1)
                bestSet[i] = i

        dist = farthest(size n, points p)           //1 step
        bestDistance = n * distance                         //1 step

        A = new int[n]

        for int i = 0 to n:                                         //1*(n-0+1)
                A[i] = i

        print_perm(size n, A, size n, bestSet, bestDistance)

        output("hamiltonian cycle of min length: ")
        print_cycle(size n, points P, bestSet)

        output("best set: ")

        for i = 0 to n:                                             //1* (n-0+1)
                output("bestSet[i]")
```

```
        delete P                                                    //1 step
        delete A                                                    //1 step
        delete bestSet                                          //1 step

        return


print_cycle(size n, points P, int sequence):

        for i = 0 to n:                                         //2 *(n-0+1)
                output("P[seq[i]].x P[seq[i]].y)
                output("P[seq[0]].x P[seq[0]].y)


farthest(size n, points p):
        max_dist = 0
        int i,j
        dist

        for i = 0 to n-1:                                       //(n-1-0+1)
                for j = 0 to n -1:                              //3 * (n-1-0+1)
                        dist = (P[i].x - P[j].x)*(P[i].x - P[j].x) + (P[i].y - P[j].y)*(P[i].y - P[j].y)
                        if max_dist < dist:
                                max_dist = dist
        return sqrt(max_dist)

        //worst case runtime: O(n^2)


print_perm(size n, int a, size size_of_a, points p, bestSet, bestDistance)
        int i
        dist, totalDist = 0
        xC, yC

        if n == 1:
        //1 + max(4n+6, 0)
                for i = 0 to size_of_a -1:              //4 * (n-1-0+1)
                        yC = pow((P[A[i + 1]].y - P[A[i]].y), 2)
                        xC = pow((P[A[i + 1]].x - P[A[i]].x), 2)

                        dist = sqrt(yC + xC)

                        totalDist +=dist
```

```
                yC = pow((P[A[sizeA - 1]].y - P[A[0]].y), 2)      //1 step
                xC = pow((P[A[sizeA - 1]].x - P[A[0]].x), 2)      //1 step
                dist = sqrt(yC + xC)                                                    //1 step

                totalDist += dist                                                               //1
step


                if totalDist < bestDist:                                        //1 + max(1*n,0)
                        bestDist = totalDist

                        for int i = 0 to size_of_a -1                   //1*(n-1-0+1)
                                bestSet[i] = A[i]

        else:
                for int = 0 to n-1:                                                     //
                        print_perm(n-1, A, size_of_a, P bestSet, bestDist)
                        if n%2 == 0:                                                    //1 +
max(3,3)
                                temp = A[i]
                                A[i] = A[n-1]
                                A[n-1] = temp
                        else:
                                temp = A[0]
                                A[0] = A[n - 1]
                                A[n - 1] = temp
                print_perm(n-1, A, size_of_a, P, bestSet, BestDist)   //1 step
```

Worst case Run time = O(n^2)

Nearest Neighbor

```
main():
        pointer2d *P                                    //1 step
        int *M                                                  //1 step
        bool *visited                                   //1 step
        int i, n                                        //1 step
        float dist                                              //1 step
        int A,B                                                 //1 step

        input(number_of _elements)              //1 step

        if(n < 3)                                               //1+max(1,0) * 1
                exit

   P = new point2d[n]                           //1

        ouput("Enter distinct points")

        for i in 0 to n:                                //4 * (n - 0 + 1)
                output("Enter x")
                input(P[i].x)
                output("Enter y")
                input(P[i].y)

        M = new int[n]                                  //1 step

        for i in 0 to n:                                //1*(n-0+1)
                M[i] = i

        visited = new bool[n]

        for i in 0 to n:                                //1*(n-0+1)
                visited[i] = false

        A = farthest_point(size n, Points p)    //1 step

        i = 0                                                   //1 step
        M[i] = A                                                //1 step

        visited[A] = true                               //1 step

        for i = 1 to n:                                 //4*(n-1+1)
                B = nearest(size m, Points P, farthest A, visited)
```

```
            A = B

            M[i] = A
            visited[A] = true

        dist = 0
        float xC, yC

        for i in 0 to n-1:                                          //3 *(n-1-0+1)
            xC = pow((P[M[i]].x - P[M[i + 1]].x), 2)
            yC = pow((P[M[i]].y - P[M[i + 1]].y), 2)
            dist += sqrt(xC + yC)

        xC = pow((P[M[0]].x - P[M[n - 1]].x), 2); //1 step
        yC = pow((P[M[0]].y - P[M[n - 1]].y), 2); //1 step
        dist += sqrt(xC + yC);                                       //1 step


        output("Hamiltonian Cycle: ")
        print_cycle(size n, Points P, PAth M)                //1 step
        output("Min length")                                 //1 step

        delete M                                                    //1 step
        return                                                      //1 step


print_cycle(size n, points p, int seqeunce):
        int i                                                       //1 step
        for i in 0 to n:                                     //2 * (n-0+1)
            ouput("P[seq[i]].x P[seq[i]].y")
            output("P[seq[0]].x P[seq[0]].y")

farthest_point(size n, points p):
        max_dist = 0                                                //1 step
        i,j,A = 0                                                   //3 steps
        dist                                                        //1 step

        for i in 0 to n:                                     //3(n-1)(n-1)
            for j in 0 to n:                                 //3*(n-0+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))
                if(max_dist < dist):
                    max_dist = dist
                    A = i
```

```
        return A
        //worst case run time O(n^2)


nearest(size n, points p, int a, bool visited):
        smallest = 0, distance
        nearest_index = 0
        xC, xY

        for i in 0 to n:                                                    //8 * (n-0+1)
                if(i != A):                                                                 //1 + max(4 + max(1+
max(max(2,0), 2 ), 0) 0)
                        if visited[i] == false:
                                xC = pow((P[i].x - P[A].x), 2)
                                yC = pow((P[i].y - P[A].y), 2)
                                distance = sqrt(xC+yC)

                                if smallest > 0:
                                        if distance < smallest:
                                                smallest = distance
                                                nearest_index = i
                                else
                                        smallest = distance
                                        nearest_index = i
        return nearst_index

        worst case runtime = O(n^2)
```