// check to see if all cities are visited or taken
bool visitedAll(List<int> visited, int n)
{
    for (int j=0 ; j<n ; j++)
        if (visited[j]==0)
            return false;
    return true;
}

// check to see if all cities are visited or taken
bool visitedAll(List<bool> visited)
{
    for (int j = 0; j < visited.Count; j++)
        if (visited[j] == false)
            return false;
    return true;
}

// find index of a given gene
int FindIndexOfGene(Solution temps, int n, int gene)
{
    for (int i = 1; i < n ; i++)
    {
        if (temps.Chromosome[i] == gene)
            return i;
    }
    return 0;
}

// compare two solutions if matched
bool isMatch(Solution s1, Solution s2, int n)
{
    for (int i = 1; i < n ; i++)
    {
        if (s1.Chromosome[i] != s2.Chromosome[i])
            return false;
    }
    return true;
}
ulong CalcCost(Solution temps, int n)
{
    ulong sumcost = 0;
    // Solution temps = SingleTSPPup[s1];
    for (int i = 0; i < n - 1; i++)
    {
        sumcost = sumcost + (ulong) Cities[temps.Chromosome[i]][temps.Chromosome[i + 1]]; // distanc(i, i+1)
    }
    //adding the last city distance to the first
    sumcost = sumcost + (ulong) Cities[temps.Chromosome[n - 1]][temps.Chromosome[0]];
    return sumcost;
}

double ED(Point p1, Point p2) {
    double xx = (double) (p1.X - p2.X);
    double yy = (double) (p1.Y - p2.Y);
    xx = xx * xx;
    yy = yy * yy;
    return Math.Sqrt(xx + yy);
}

double ED(doublePoint p1, doublePoint p2) {
    double xx = (double) (p1.X - p2.X);
    double yy = (double) (p1.Y - p2.Y);
    xx = xx * xx;
    yy = yy * yy;
    return Math.Sqrt(xx + yy);
}

double ED(int x, int y) {
    Point p1 = CitiesXY[x];
Point p2 = CitiesXY[y];
double xx = (double)(p1.X - p2.X);
double yy = (double)(p1.Y - p2.Y);
xx = xx * xx;
yy = yy * yy;
xx = Math.Sqrt(xx + yy);
return xx;
}
//calculate cost for a solution if distances are taken from an array of xy points (used)
double CalcCostED(Solution temps, int n)
{
    double sumcost = 0.0;
    // Solution temps = SingleTSPPup[s1];
    Point P1, P2;
    for (int i = 0; i < n - 1; i++)
    {
        P1 = CitiesXY[temps.Chromosome[i]];
        P2 = CitiesXY[temps.Chromosome[i + 1]];
        double ed = ED(P1, P2);
        sumcost += ed; // ED distance (CITY i AND i+1)
    }
    // adding the last city distance to the first
    P1 = CitiesXY[temps.Chromosome[n - 1]];
    P2 = CitiesXY[temps.Chromosome[0]];
    sumcost += ED(P1, P2);
    return sumcost;
}

// do mutation on random solution of a given population (pup)
Solution mutation(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count); // get the random index of a solution

    Solution temp = new Solution();
    temp.Chromosome = new List<int>(CityNum); // create empty solution
for (int j = 0; j < CityNum; j++)
{
    temp.Chromosome.Add(pup[index].Chromosome[j]); // copy the selected solution to the empty one
}
int whichgene1 = rand.Next(1, CityNum); // select random location for mutation
int whichgene2 = rand.Next(1, CityNum); // select random location to swap the mutated gene
int val1 = temp.Chromosome[whichgene1]; // to be replaced
int val2 = temp.Chromosome[whichgene2]; // to be replaced
temp.Chromosome[whichgene1] = val2; // swapped
temp.Chromosome[whichgene2] = val1;
temp.Cost = (ulong)CalcCostED(temp, CityNum); // calc the cost of the new mutated solution
return temp;

// find the city of the furthest distance from its previous neighbor
int FindIndxWorstGene(Solution temp)
{
    ulong max = 0, distance; int indx = 0;
    for (int i = 0; i < temp.Chromosome.Count - 1; i++)
    {
        distance = (ulong)ED(CitiesXY[temp.Chromosome[i]], CitiesXY[temp.Chromosome[i + 1]]);
        if (distance > max)
        {
            max = distance;
            indx = i + 1;
        }
    }
    distance = (ulong)ED(CitiesXY[temp.Chromosome[temp.Chromosome.Count - 1]], CitiesXY[temp.Chromosome[0]]);
    if (distance > max)
    {
        max = distance;
        indx = temp.Chromosome.Count - 1;
    }
    return indx;
}

// find the city of the farthest distance from its previous and next neighbors
int FindIndxWorstGeneLR(Solution temp)
ulong max = 0, distance, distanceL, distanceR;
int indx = 0;
for (int i = 1; i < temp.Chromosome.Count - 1; i++)
{
    distanceL = (ulong)ED(CitiesXY[temp.Chromosome[i]], CitiesXY[temp.Chromosome[i - 1]]);
    distanceR = (ulong)ED(CitiesXY[temp.Chromosome[i]], CitiesXY[temp.Chromosome[i + 1]]);
    distance = distanceL + distanceR;
    if (distance > max)
    {
        max = distance;
        indx = i;
    }
}
distanceL = (ulong)ED(CitiesXY[temp.Chromosome[temp.Chromosome.Count - 1]],
    distance = distanceL + distanceR;
    if (distance > max)
    {
        max = distance;
        indx = temp.Chromosome.Count - 1;
    }
    return indx;

//find 2 indexes of the maximum distances from their prev cities
Point FindIndexesWorstGenes(Solution temp)
{
    ulong max = 0, distance; int indx = 0, indx2 = 0;
    ulong second_max = 0;
    for (int i = 0; i < temp.Chromosome.Count - 1; i++)
    {
        distance = (ulong)ED(CitiesXY[temp.Chromosome[i]], CitiesXY[temp.Chromosome[i + 1]]);
        if (distance > max)
        {
            second_max = max;
            indx2 = indx;
            max = distance;
            indx = i;
        }
    }
    return indx,
}
max = distance;
indx = i + 1;
}
else if (distance > second_max)
{
    second_max = distance;
    indx2 = i + 1;
}
}
distance = (ulong)ED(CitiesXY[temp.Chromosome[temp.Chromosome.Count - 1]], CitiesXY[temp.Chromosome[0]]);
if (distance > max)
{
    second_max = max;
    indx2 = indx;
    max = distance;
    indx = temp.Chromosome.Count - 1;
}
else if (distance > second_max)
{
    second_max = distance;
    indx2 = temp.Chromosome.Count - 1;
}
Point p = new Point(indx, indx2);
return p;
}

//do mutation on worst gene and swap it with rand gene
Solution mutationEnhanced(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count); //get the random index of a solution
    Solution temp = new Solution();
    temp.Chromosome = new List<int>(CityNum); //create empty solution
    for (int j = 0; j < CityNum; j++)
    {
        temp.Chromosome.Add(pup[index].Chromosome[j]); //copy the selected solution to the empty one
    }
int whichgene1 = FindIndxWorstGene(temp);//select worst gene for mutation
int whichgene2 = rand.Next(1, CityNum);//select random location to swap the mutated gene
int val1 = temp.Chromosome[whichgene1];//to be replaced
int val2 = temp.Chromosome[whichgene2];//to be replaced
temp.Chromosome[whichgene1] = val2;//swaped
temp.Chromosome[whichgene2] = val1;
temp.Cost = (ulong)CalcCostED(temp, CityNum);//calc the cost of the new mutated solution
return temp;
}

//do mutation on worst gene and swap it locally with its right neighbour
//or swap the previous gene with its previous, return the best chromosome
Solution mutationEnhancedWorstSwapLocally(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count);//get the random index of a solution

    Solution temp1 = new Solution();
temp1.Chromosome = new List<int>(CityNum);//create empty solution
for (int j = 0; j < CityNum; j++)
{
    temp1.Chromosome.Add(pup[index].Chromosome[j]);//copy the selected solution to the empty one
}

    Solution temp2 = new Solution();
temp2.Chromosome = new List<int>(CityNum);//create empty solution
for (int j = 0; j < CityNum; j++)
{
    temp2.Chromosome.Add(pup[index].Chromosome[j]);//copy the selected solution to the empty one
}

    int whichgene1 = FindIndxWorstGene(temp1);//select worst gene for mutation
int indexMinus1 = whichgene1 - 1;
int indexMinus2 = whichgene1 - 2;
int indexPlus1 = whichgene1 + 1;
if (indexMinus1 < 1)
    indexMinus1 = indexMinus1 + temp1.Chromosome.Count - 1;
if (indexMinus2 < 1)
```csharp
    indexMinus2 = indexMinus2 + temp1.Chromosome.Count - 1;
    if (indexPlus1 > temp1.Chromosome.Count - 1)
        indexPlus1 = 1;
    //    int whichgene1 = rand.Next(1, CityNum);//select random location for mutation
    //int whichgene2 = rand.Next(1, CityNum);//select random location tp swap the mutated gene
    //solution 1 : swap previous points
    int val1 = temp1.Chromosome[indexMinus1]; //to be replaced
    int val2 = temp1.Chromosome[indexMinus2]; //to be replaced
    temp1.Chromosome[indexMinus1] = val2; //swaped
    temp1.Chromosome[indexMinus2] = val1;     temp1.Cost = (ulong)CalcCostED(temp1, CityNum); //calc the cost of the new mutated solution
    //solution 2 : swap worst with its next
    val1 = temp2.Chromosome[whichgene1]; //to be replaced
    val2 = temp2.Chromosome[indexPlus1]; //to be replaced
    temp2.Chromosome[whichgene1] = val2; //swaped
    temp2.Chromosome[indexPlus1] = val1;
    temp2.Cost = (ulong)CalcCostED(temp2, CityNum); //calc the cost of the new mutated solution
    if (temp1.Cost < temp2.Cost)
        return temp1;
    else
        return temp2;
}
```

```csharp
//do mutation on worst gene from left nad right and swap with rand
Solution mutationEnhancedWorstLR(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count); //get the random index of a solution

    Solution temp = new Solution();
temp.Chromosome = new List<int>(CityNum); //create empty solution
    for (int j = 0; j < CityNum; j++)
    {
        temp.Chromosome.Add(pup[index].Chromosome[j]); //copy the selected solution to the empty one
    }

    int whichgene1 = FindIndxWorstGeneLR(temp); //select worst gene for mutation
    int whichgene2 = rand.Next(1, CityNum); //select random location tp swap the mutated gene
    int val1 = temp.Chromosome[whichgene1]; //to be replaced
    int val2 = temp.Chromosome[whichgene2]; //to be replaced
```
temp.Chromosome[whichgene1] = val2; // swapped
temp.Chromosome[whichgene2] = val1;
temp.Cost = (ulong)CalcCostED(temp, CityNum); // calc the cost of the new mutated solution
    return temp;
}

// do mutation on worst gene from left and right and swap with random gene
// in NNRange around its nearest neighbour city
Solution mutationEnhancedWorstLRNNcity(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count); // get the random index of a solution
    // NNRange = 5;
    Solution temp = new Solution();
    temp.Chromosome = new List<int>(CityNum); // create empty solution
    for (int j = 0; j < CityNum; j++)
    {
        temp.Chromosome.Add(pup[index].Chromosome[j]); // copy the selected solution to the empty one
    }

    int whichgene1 = FindIndxWorstGeneLR(temp); // select index of worst gene for mutation
    // find the worst city of index worst gen
    int WorstCity = temp.Chromosome[whichgene1];
    // find the nearest city to the worst city
    int NNcty = NearestCity[WorstCity];
    // find the index of the nearest city in the mutated solution
    int indexNNcity = temp.Chromosome.IndexOf(NNcty);
    // set the range for the random city to be swapped with the worst gen
    int r1 = indexNNcity - NNRange;
    int r2 = indexNNcity + NNRange;

    int whichgene2 = rand.Next(r1, r2); // select random location to swap the mutated gene in the best neighbourhood range
    if (whichgene2 < 0)
        whichgene2 = whichgene2 + temp.Chromosome.Count;
    if (whichgene2 > temp.Chromosome.Count - 1)
        whichgene2 = whichgene2 - temp.Chromosome.Count;
    if (whichgene2 == 0)
whichgene2++;
//swap
int val1 = temp.Chromosome[whichgene1];  // to be replaced
int val2 = temp.Chromosome[whichgene2];  // to be replaced
temp.Chromosome[whichgene1] = val2;  // swaped
temp.Chromosome[whichgene2] = val1;
temp.Cost = (ulong)CalcCostED(temp, CityNum);  // calc the cost of the new mutated solution
return temp;
}

// do mutation on worst gene from left and right, find its nearest neighbour city then
// and locate a range around nearest neighbour city NNRange
// find the furthest city from the NNcity in this Range to be swaped with the worst gene
Solution mutationEnhancedWorstLRFurthestNNcity(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count);  // get the random index of a solution
    Solution temp = new Solution();
    temp.Chromosome = new List<int>(CityNum);  // create empty solution
    for (int j = 0; j < CityNum; j++)
    {
        temp.Chromosome.Add(pup[index].Chromosome[j]);  // copy the selected solution to the empty one
    }

    int whichgene1 = FindIndxWorstGeneLR(temp);  // select index of worst gene for mutation
    // find the worst city of index worst gen
    int WorstCity = temp.Chromosome[whichgene1];
    // find the nearest city to the worst city
    int NNcty = NearestCity[WorstCity];
    // find the index of the nearest city in the mutated solution
    int indexNNcity = temp.Chromosome.IndexOf(NNcty);
    // set the range for the random city to be swapped with the worst gen
    int r1 = indexNNcity - NNRange;
    int r2 = indexNNcity + NNRange;
    int indxg = indexNNcity;  // default value
    double maxdistance = -1;
    for (int g = r1; g <= r2; g++)
    {
        int gg=g;
        ...
if (gg < 0) // to get red of negative index nd rotte to the right of array
    gg = gg + temp.Chromosome.Count;
if (gg < temp.Chromosome.Count - 1) // if index outside array rotate to the left of array
    gg = gg - temp.Chromosome.Count;
if (gg == 0)
    continue; // to prevent swapping with the start point the (0)
if (gg == whichgene1)
    continue; // to prevent swapping with the same worst gene

double D = ED(CitiesXY[temp.Chromosome[gg]], CitiesXY[temp.Chromosome[indexNNcity]]);
if (D > maxdistance) {
    maxdistance = D;
    indxg = gg;
}

if (indxg == 0) // to prevent swapping with the start point the (0)
    indxg++;

int whichgene2 = indxg; // select random location to swap the mutated gene in the best neighbourhood range
/* int whichgene2 = rand.Next(r1, r2); // select random location to swap the mutated gene in the best neighbourhood range */
if (whichgene2 < 0)
    whichgene2 = whichgene2 + temp.Chromosome.Count;
if (whichgene2 > temp.Chromosome.Count - 1)
    whichgene2 = whichgene2 - temp.Chromosome.Count;
if (whichgene2 == 0)
    whichgene2++;

// swap
int val1 = temp.Chromosome[whichgene1]; // to be replaced
int val2 = temp.Chromosome[whichgene2]; // to be replaced

temp.Chromosome[whichgene1] = val2; // swapped
temp.Chromosome[whichgene2] = val1;

temp.Cost = (ulong)CalcCostED(temp, CityNum); // calc the cost of the new mutated solution

    return temp;
}

// do mutation on worst gene from left and right, find its nearest neighbour city then
//insert worst near the nearest neighbour city NNRan ge
Solution mutationEnhancedWorstLRInsertNNcity(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count);// get the random index of a solution
    Solution temp = new Solution();
    temp.Chromosome = new List<int>(CityNum);// create empty solution
    for (int j = 0; j < CityNum; j++)
    {
        temp.Chromosome.Add(pup[index].Chromosome[j]); // copy the selected solution to the empty one
    }
    int whichgene1 = FindIndxWorstGeneLR(temp);// select index of worst gene for mutation
    // find the worst city of index worst gen
    int WorstCity = temp.Chromosome[whichgene1];
    // find the nearest city to the worst city
    int NNcty = NearestCity[WorstCity];
    // find the index of the nearest city in the mutated solution
    int indexNNcity = temp.Chromosome.IndexOf(NNcty);
    if (indexNNcity < whichgene1) {
        for (int g = whichgene1; g > indexNNcity; g--) {
            temp.Chromosome[g] = temp.Chromosome[g - 1];
        }
        temp.Chromosome[indexNNcity + 1] = WorstCity;
    }
    if (indexNNcity > whichgene1)
    {
        for (int g = whichgene1; g < indexNNcity; g++)
        {
            temp.Chromosome[g] = temp.Chromosome[g + 1];
        }
        temp.Chromosome[indexNNcity - 1] = WorstCity;
    }
    int rnd = rand.Next(0, 2);
    if (rnd == 0) {
        if (indexNNcity > 1 && indexNNcity != temp.Chromosome.Count - 1) {
            int val1 = temp.Chromosome[indexNNcity-1]; // to be replaced
            int val2 = temp.Chromosome[indexNNcity+1]; // to be replaced
            temp.Chromosome[indexNNcity-1] = WorstCity;
        }
    }
}
temp.Chromosome[indexNNcity - 1] = val2; //swaped
    temp.Chromosome[indexNNcity + 1] = val1;
}
    temp.Cost = (ulong)CalcCostED(temp, CityNum); //calc the cost of the new mutated solution

    return temp;

}

//do mutation on random gene, find its nearest neighbour city then
//insert it near its nearest neighbour city NNRange
Solution mutationEnhancedRandInsertNNcity(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count); //get the random index of a solution
    Solution temp = new Solution();
    temp.Chromosome = new List<int>(CityNum); //create empty solution
    for (int j = 0; j < CityNum; j++)
    {
        temp.Chromosome.Add(pup[index].Chromosome[j]); //copy the selected solution to the empty one
    }

    int whichgene1 = rand.Next(1, temp.Chromosome.Count); //select index randomly for mutation
    //find the city of the random index
    int RandCity = temp.Chromosome[whichgene1];
    //find the nearest city to the rnd city
    int NNcty = NearestCity[RandCity];
    //find the index of the nearest city in the mutated solution
    int indexNNcity = temp.Chromosome.IndexOf(NNcty);

    if (indexNNcity < whichgene1)
    {
        for (int g = whichgene1; g > indexNNcity; g--)
        {
            temp.Chromosome[g] = temp.Chromosome[g - 1];
        }
        temp.Chromosome[indexNNcity + 1] = RandCity;
    }
    if (indexNNcity > whichgene1)
{ for (int g = whichgene1; g < indexNNcity; g++)
    { temp.Chromosome[g] = temp.Chromosome[g + 1];
    }
    temp.Chromosome[indexNNcity - 1] = RandCity;
} int rnd = rand.Next(0, 2);
if (rnd == 0)
{ if (indexNNcity > 1 && indexNNcity < temp.Chromosome.Count - 1)
    { int val1 = temp.Chromosome[indexNNcity - 1]; //to be replaced
      int val2 = temp.Chromosome[indexNNcity + 1]; //to be replaced
      temp.Chromosome[indexNNcity - 1] = val2; //swaped
      temp.Chromosome[indexNNcity + 1] = val1;
    }
} temp.Cost = (ulong)CalcCostED(temp, CityNum); //calc the cost of the new mutated solution
    return temp;
}

//do mutation on worst gene, find the nearest city from random cities and
//insert it between worst city and its previous neighbour, the nearest is the min distance from both worst and its
previous
Solution mutationEnhancedWorstInsertRandBefore(List<Solution> pup)
{ int index = rand.Next(0, pup.Count); //get the random index of a solution
    Solution temp = new Solution();
    temp.Chromosome = new List<int>(CityNum); //create empty solution
    for (int j = 0; j < CityNum; j++)
    { temp.Chromosome.Add(pup[index].Chromosome[j]); //copy the selected solution to the empty one
    }
    int whichgene1 = FindIndxWorstGene(temp); //select worst gene for mutation
    int prevIndex = whichgene1 - 1;
//select random citites
List<int> RandomCities = new List<int>();
int NumRndCities = temp.Chromosome.Count / 10;
//int NumRndCities = temp.Chromosome.Count;// / 10;
int RndCity;
for (int r1 = 0; r1 < NumRndCities; r1++) {
    nextRand:
        RndCity = rand.Next(1, temp.Chromosome.Count);
        if (RndCity == whichgene1 || RndCity == prevIndex)
            goto nextRand;
        RandomCities.Add(RndCity);
    }
//find the nearest city to both worst and its previous to be inserted in between.
int RandCity = 1; //default value
double min = double.MaxValue;
for (int r1 = 0; r1 < NumRndCities; r1++) {
    double D = ED(CitiesXY[temp.Chromosome[whichgene1]], CitiesXY[RandomCities[r1]]) +
               ED(CitiesXY[temp.Chromosome[prevIndex]], CitiesXY[RandomCities[r1]]);
    if (D < min) {
        min = D;
        RandCity = RandomCities[r1];
    }
}
int indexRndCity = temp.Chromosome.IndexOf(RandCity);
//  int PrevCity = temp.Chromosome[prevIndex];
//  int WorstCity = temp.Chromosome[whichgene1];
if (indexRndCity < prevIndex)
    for (int g = indexRndCity; g < prevIndex; g++)
    {
        temp.Chromosome[g] = temp.Chromosome[g + 1];
    }
    temp.Chromosome[prevIndex] = RandCity;
if (indexRndCity > whichgene1)
for (int g = indexRndCity; g > whichgenel; g--)
{
    temp.Chromosome[g] = temp.Chromosome[g - 1];
}
temp.Chromosome[whichgenel] = RandCity;
}

//rotate right
int t = temp.Chromosome[temp.Chromosome.Count - 1];
for (int g = temp.Chromosome.Count - 1; g > 1; g--)
{
    temp.Chromosome[g] = temp.Chromosome[g - 1];
}
temp.Chromosome[1] = t;
temp.Cost = (ulong)CalcCostED(temp, CityNum);//calc the cost of the new mutated solution

return temp;

//do mutation on random gene, select random genes, calc the distance between the selected gene and all the selected
cities, find the nearest city and
//insert it between the selected random city and its previous neighbour, the nearest is the min distance from both
//the selected gene and its previous
Solution mutationEnhancedRandInsertRandBefore(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count);//get the random index of a solution
    Solution temp = new Solution();
temp.Chromosome = new List<int>(CityNum);//create empty solution
    for (int j = 0; j < CityNum; j++)
    {
        temp.Chromosome.Add(pup[index].Chromosome[j]);//copy the selected solution to the empty one
    }
    int whichgenel = rand.Next(1, CityNum);//select random location for mutation
    int prevIndex = whichgenel - 1;
    //select random citites
    List<int> RandomCities = new List<int>();
    int NumRndCities = temp.Chromosome.Count / 10;
    //int NumRndCities = temp.Chromosome.Count;// / 10;
int RndCity;
for (int r1 = 0; r1 < NumRndCities; r1++)
{
    nextRand:
    RndCity = rand.Next(1, temp.Chromosome.Count);
    if (RndCity == whichgene1 || RndCity == prevIndex)
        goto nextRand;
    RandomCities.Add(RndCity);
}
//find the nearest city to both worst and its previous to be inserted in between.
int RandCity = 1; //default value
double min = Double.MaxValue;

for (int r1 = 0; r1 < NumRndCities; r1++)
{
    double D = ED(CitiesXY[temp.Chromosome[whichgene1]], CitiesXY[RandomCities[r1]]) +
               ED(CitiesXY[temp.Chromosome[prevIndex]], CitiesXY[RandomCities[r1]]);
    if (D < min)
    {
        min = D;
        RandCity = RandomCities[r1];
    }
}

int indexRndCity = temp.Chromosome.IndexOf(RandCity);
// int PrevCity = temp.Chromosome[prevIndex];
// int WorstCity = temp.Chromosome[whichgene1];
if (indexRndCity < prevIndex)
{
    for (int g = indexRndCity; g < prevIndex; g++)
    {
        temp.Chromosome[g] = temp.Chromosome[g + 1];
    }
    temp.Chromosome[prevIndex] = RandCity;
}
if (indexRndCity > whichgene1)
{
    for (int g = indexRndCity; g > whichgene1; g--)
    {
        temp.Chromosome[g] = temp.Chromosome[g + 1];
    }
    temp.Chromosome[whichgene1] = RandCity;
}
temp.Chromosome[g] = temp.Chromosome[g - 1];
}
temp.Chromosome[whichgene1] = RandCity;
}/rotate right
int t = temp.Chromosome[temp.Chromosome.Count - 1];
for (int g = temp.Chromosome.Count - 1; g > 1; g--)
{
    temp.Chromosome[g] = temp.Chromosome[g - 1];
}
temp.Chromosome[1] = t;
temp.Cost = (ulong)CalcCostED(temp, CityNum);//calc the cost of the new mutated solution
return temp;
}//do mutation on worst gene best of 3
Solution mutationEnhancedBo3(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count);//get the random index of a solution
    Solution temp1 = new Solution();
    Solution temp2 = new Solution();
    Solution temp3 = new Solution();
temp1.Chromosome = new List<int>(CityNum);//create empty solution
temp2.Chromosome = new List<int>(CityNum);//create empty solution
temp3.Chromosome = new List<int>(CityNum);//create empty solution
for (int j = 0; j < CityNum; j++)
{
    temp1.Chromosome.Add(pup[index].Chromosome[j]);//copy the selected solution to the empty one
    temp2.Chromosome.Add(pup[index].Chromosome[j]);//copy the selected solution to the empty one
    temp3.Chromosome.Add(pup[index].Chromosome[j]);//copy the selected solution to the empty one
}

int whichgene1 = FindIndxWorstGene(temp1);//select worst gene for mutation
//    int whichgene1 = rand.Next(1, CityNum);//select random location for mutation
int third = temp1.Chromosome.Count / 3;
```csharp
int whichgene2 = rand.Next(1, third); // select random location to swap the mutated gene
int val1 = temp1.Chromosome[whichgene1]; // to be replaced
int val2 = temp1.Chromosome[whichgene2]; // to be replaced
temp1.Chromosome[whichgene1] = val2; // swapped
temp1.Chromosome[whichgene2] = val1;
temp1.Cost = (ulong)CalcCostED(temp1, CityNum); // calc the cost of the new mutated solution

// solution 2
whichgene2 = rand.Next(third, 2 * third); // select random location to swap the mutated gene
val1 = temp2.Chromosome[whichgene1]; // to be replaced
val2 = temp2.Chromosome[whichgene2]; // to be replaced
temp2.Chromosome[whichgene1] = val2; // swapped
temp2.Chromosome[whichgene2] = val1;
temp2.Cost = (ulong)CalcCostED(temp2, CityNum); // calc the cost of the new mutated solution

// solution 3
whichgene2 = rand.Next(2 * third, CityNum); // select random location to swap the mutated gene
val1 = temp3.Chromosome[whichgene1]; // to be replaced
val2 = temp3.Chromosome[whichgene2]; // to be replaced
temp3.Chromosome[whichgene1] = val2; // swapped
temp3.Chromosome[whichgene2] = val1;
temp3.Cost = (ulong)CalcCostED(temp3, CityNum); // calc the cost of the new mutated solution
    return temp1;
else if (temp2.Cost > temp3.Cost)
    return temp2;
else
    return temp3;
}
```

// do mutation on worst gene and swap it with the next worst gene

```csharp
Solution mutationEnhancedWoW(List<Solution> pup)
{
    int index = rand.Next(0, pup.Count); // get the random index of a solution

    Solution temp = new Solution();
temp.Chromosome = new List<int>(CityNum); // create empty solution
    for (int j = 0; j < CityNum; j++)
    {
```
temp.Chromosome.Add(pup[index].Chromosome[j]); // copy the selected solution to the empty one

Point p1 = FindIndexesWorstGenes(temp);

int whichgene1 = p1.X; // select worst gene for mutation
//    int whichgene1 = rand.Next(1, CityNum); // select random location for mutation
int whichgene2 = p1.Y; // select next worst gene for mutation
int val1 = temp.Chromosome[whichgene1]; // to be replaced
int val2 = temp.Chromosome[whichgene2]; // to be replaced

// swap the genes
temp.Chromosome[whichgene1] = val2;
temp.Chromosome[whichgene2] = val1;
temp.Cost = (ulong)CalcCostED(temp, CityNum); // calc the cost of the new mutated solution

return temp;

// do enhanced mutation for a sample of solution
void mutationEnhanced(int type)
{
    float MutRatio = (float)this.trackBar1.Value / (float)trackBar1.Maximum;

    int Mutations = (int)(MutRatio * PopSize); // number of Mutation per generation

    // dynamic mutation ratio
    if (this.checkBox4.Checked)
    {
        double MR = (double)progressBar1.Value / (double)progressBar1.Maximum;
        Mutations = (int)(MR * PopSize);
    }

    List<Solution> mut = new List<Solution>(); // create a list for the sample of mutated solutions
    List<Solution> BestMutation; // temporary solutions to get the best and add to mut

    for (int i = 0; i < Mutations; i++)
    {
        if (type == 1)
            mut.Add(mutationEnhanced(SingleTSPPup)); // add new mutated solutions
        else if (type == 2)
mut.Add(mutationEnhancedWoW(SingleTSPPup));
else if (type == 3)
    mut.Add(mutationEnhancedBo3(SingleTSPPup));
else if (type == 4)
    mut.Add(mutationEnhancedWorstLR(SingleTSPPup));
else if (type == 5)
    mut.Add(mutationEnhancedWorstLRNNcity(SingleTSPPup));
else if (type == 6)
    mut.Add(mutationEnhancedWorstLRFurthestNNcity(SingleTSPPup));
else if (type == 7)
    mut.Add(mutationEnhancedRandInsertNNcity(SingleTSPPup));
else if (type == 8)
    mut.Add(mutationEnhancedWorstSwapLocaly(SingleTSPPup));
else if (type == 9)
    mut.Add(mutationEnhancedWorstInsertRandBefore(SingleTSPPup));
else if (type == 10)
    mut.Add(mutationEnhancedRandInsertRandBefore(SingleTSPPup));
else
    { //try all mutations and get the best solution and not in population
      BestMutation = new List<Solution>();
      BestMutation.Add(mutationEnhanced(SingleTSPPup));
      BestMutation.Add(mutationEnhancedWoW(SingleTSPPup));
      BestMutation.Add(mutationEnhancedBo3(SingleTSPPup));
      BestMutation.Add(mutationEnhancedWorstLR(SingleTSPPup));
      BestMutation.Add(mutationEnhancedWorstLRNNcity(SingleTSPPup));
      BestMutation.Add(mutationEnhancedWorstLRFurthestNNcity(SingleTSPPup));
      BestMutation.Add(mutationEnhancedRandInsertNNcity(SingleTSPPup));
      BestMutation.Add(mutationEnhancedWorstSwapLocaly(SingleTSPPup));
      BestMutation.Add(mutationEnhancedWorstInsertRandBefore(SingleTSPPup));
      BestMutation.Add(mutationEnhancedRandInsertRandBefore(SingleTSPPup));
      bool found = false;
      BestMutation.Sort((s1, s2) => s1.Cost.CompareTo(s2.Cost));
      while (BestMutation.Count > 0 && !found)
      {
        }
      }

}
if (!inPopulation(BestMutation[0]))
{
    found = true;
    mut.Add(BestMutation[0]);
    break; // you can add a counter for each mutation here
}
else
{
    BestMutation.RemoveAt(0);
}
}

for (int k = 0; k < mut.Count; k++)
{
    if (!inPopulation(mut[k])) // make sure no repetition
    {
        SingleTSPPup.Add(mut[k]);
    }
}

mut.Clear(); // garbage collection

// do mutation for a sample of solution
void mutation()
{
    float MutRatio = (float) this.trackBar1.Value / (float) trackBar1.Maximum;

    int Mutations = (int) (MutRatio * PopSize); // number of Mutation per generation
    // dynamic mutation start small end 100%
    if (this.checkBox4.Checked)
    {
        double MR = (double) progressBar1.Value / (double) progressBar1.Maximum;
        Mutations = (int) (MR * PopSize);
    }
    // dynamic mutation start 100% end small
if (this.checkBox5.Checked)
{
    double MR = 1 - ((double)progressBar1.Value / (double)progressBar1.Maximum);
    Mutations = (int)(MR * PopSize);
}

List<Solution> mut = new List<Solution>(); // creat a list for the sample of mutated solutions
for (int i = 0; i < Mutations; i++)
{
    mut.Add(mutation(SingleTSPPup)); // add new mutated solutions
}
for (int k = 0; k < mut.Count; k++) // copy the mutated solutions list to the population
    if (!inPopulation(mut[k])) // make sure no repetition
    {
        SingleTSPPup.Add(mut[k]);
    }
}
mut.Clear(); // garbage collection