**The algorithm produced below calculates the shortest path for a bitonic tour. One thing that I still need to choose is a sorting algorithm for when the x coordinates are not arranged in increasing order. My suggestion for a sorting algorithm would most likely be something of maybe a mergesort or a quicksort do to there fast time complexity of o(nlogn). This algorithm uses a two-dimensional array that stores the x and y coordinates. The start of this program begins with a Euclidean distance formula between two points to help calculate for the shortest path. This method is then applied in the TSP method which finds the shortest path by going through the for loop of if i<j-1 and if i=j-1. When the case is i<j-1 then the point from arr(i+1) to arr[j] should appear in the increasing sequence. So I end up getting the qequation of arr[I,j] = arr[I,j-1] + distance(j-1,j).**

**public** **class** BitonicTSP

{

//calculates the distance between two points using the Euclidean distance formula

**static** **double** distance(**int**[][]arr, **int** i, **int** j)

{

//start by subtracting the two x values

**double** x = arr [j] [0] -arr [i] [0];

// subtracting the two y values

**double** y = arr [j] [1] -arr[i] [1];

// taking the two values up top and squaring them then adding them together

**double** distance = Math.*sqrt*((Math.*pow* ((x), 2)

+ Math.*pow* ((y), 2)));

**return** distance;

}

//calculates the shortest path for the set of values in the two-dimensional array

**static** **double** TSP(**int**[][]arr, **int** n) {

**double**[][]arr2 = **new** **double**[n][n];

//initialization

arr2[0][1] = *distance*(arr, 0, 1);

**for**(**int** j=2;j<n; j++) {

//this for loop is taking consideration that i<j-1

**for**(**int** i = 0; i <j-1;i++) {

arr2[i][j] = arr2[i][j-1]+*distance*(arr,i,j);

}

// the next point takes into consideration when the value is i=j-1

// It takes the double.MAX\_VALUE which is the max value of a double.

arr2[j-1][j] = Double.***MAX\_VALUE***;

**double** temp = 0;

**for**(**int** k = 0; k<j-1;k++) {

temp = arr2[k][j-1]+*distance*(arr, k, j);

**if**(temp<arr2[j-1][j]) {

arr2[j-1][j]= temp;

}

}

}

// producing the shortest path to go from left most to right most and back

arr2[n-1][n-1] = arr2[n-2] [n-1] + *distance*(arr, n-2, n-1);

**return** arr2[n-1][n-1];

}

//returns the shortest distance that was produced from the tsp method

**public** **static** **void** bitonic (**int**[][]arr, **int** n) {

**double** shortestDistance = *TSP*(arr,n);

System.***out***.println ("The shortest path is:" + shortestDistance);

}

**public** **static** **void** main(String args[])

{

//testing the methods

**int** n = 3;

**int**[][] x= {{1,1},{2,3},{3,1}};

*bitonic*(x ,n);

}

}