Student Declaration of Academic Integrity

This form **must** be filled in and completed by the student submitting an assignment. **Assignments submitted without the completed form will not be accepted.**

Name:	**Ciaran O'Donnell**
Student ID Number:	**15414048**
Programme:	**ECE**
Module Code:	**EE324**
Assignment Title:	**TSP Assignment**
Submission Date:	**9th November 2017**

- I understand that the University regards breaches of academic integrity and plagiarism as grave and serious.
- I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy.
- I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references.
- I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.
- I have used the DCU library referencing guidelines (available at: http://www.library.dcu.ie/LibraryGuides/Citing&ReferencingGuide/player.html) and/or the appropriate referencing system recommended in the assignment guidelines and/or programme documentation.
- By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.
- By signing this form or by submitting material for assessment online I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy (available at: http://www.dcu.ie/registry/examinations/index.shtml)

Name:_____

Date:_____9th-November-2017_____

EE 324 Project 2

Approximate solution and application of the Travelling Salesman Problem

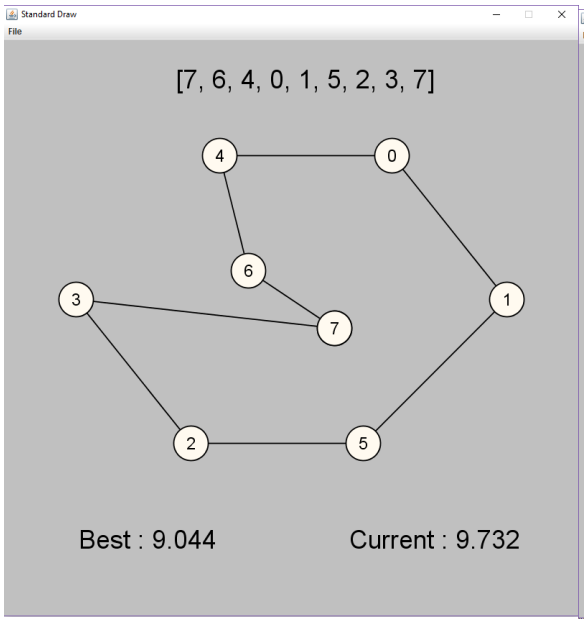
Exercise 1

The first exercise of the project was too write a java method the approximate the shortest path through a number of points. We were to do this using the nearest neighbour algorithm, i.e. from each point go to the next closest point that has not been visited until all the points have been visited.

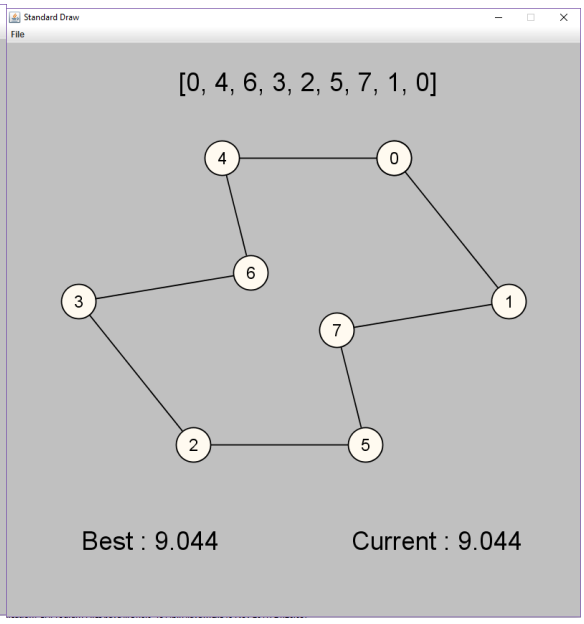
Here are the results of my program:

City	City Cords	NN Path	NN Dist	Opt Dist	%diff
0	(2.2, 1.0)	0, 4, 6, 7, 5, 2, 3, 1, 0	10.3316	9.044	14.23706
1	(3.0, 0.0)	1, 7, 6, 4, 0, 5, 2, 3, 1	11.4529	9.044	26.63534
2	(0.8, -1.0)	2, 5, 7, 6, 4, 0, 1, 3, 2	10.3316	9.044	14.23706
3	(0.0, 0.0)	3, 6, 7, 5, 2, 4, 0, 1, 3	11.4529	9.044	26.63534
4	(1.0, 1.0)	4, 6, 7, 5, 2, 3, 0, 1, 4	10.7843	9.044	19.24259
5	(2.0, -1.0)	5, 7, 6, 4, 0, 1, 2, 3, 5	10.7843	9.044	19.24259
6	(1.2, 0.2)	6, 7, 5, 2, 3, 4, 0, 1, 6	9.7323	9.044	7.610571
7	(1.8, -0.2)	7, 6, 4, 0, 1, 5, 2, 3, 7	9.7323	9.044	7.610571

My best result:



Optimal Route:



Pseudocode:

```
Array of Cities = ary
List of Cities in Correct order = Path        //starts with starting point only
Temporary List of cities = tmppath        //starts with all points in any order
City NearestNeighbour
Totaldistance                                 //variable to record the total distance
For( j = 0 to the number of cities)
        If( tmpPath is not empty ){
                For( i = 0 to tmpPath length ){    //decreases as Cities are removed from tmpPath
                        Compare distance from Path(j) to tmpPath(i)
                        If City in tmpPath(i) is closer than previous closest set it NearestNeighbour
                }End For
        Add NearestNeighbour to Path
        Remove NearestNeighbour from tmpPath
        TotalDistance += this distance
        }End if
}End For
Path add StartingCity                         //so we end where we begin
TotalDistance+= (final distance back to start)
```

We now have the total distance travelled and an array of the path in the correct order but the function is meant to return an ary of ints referring to the index of each City

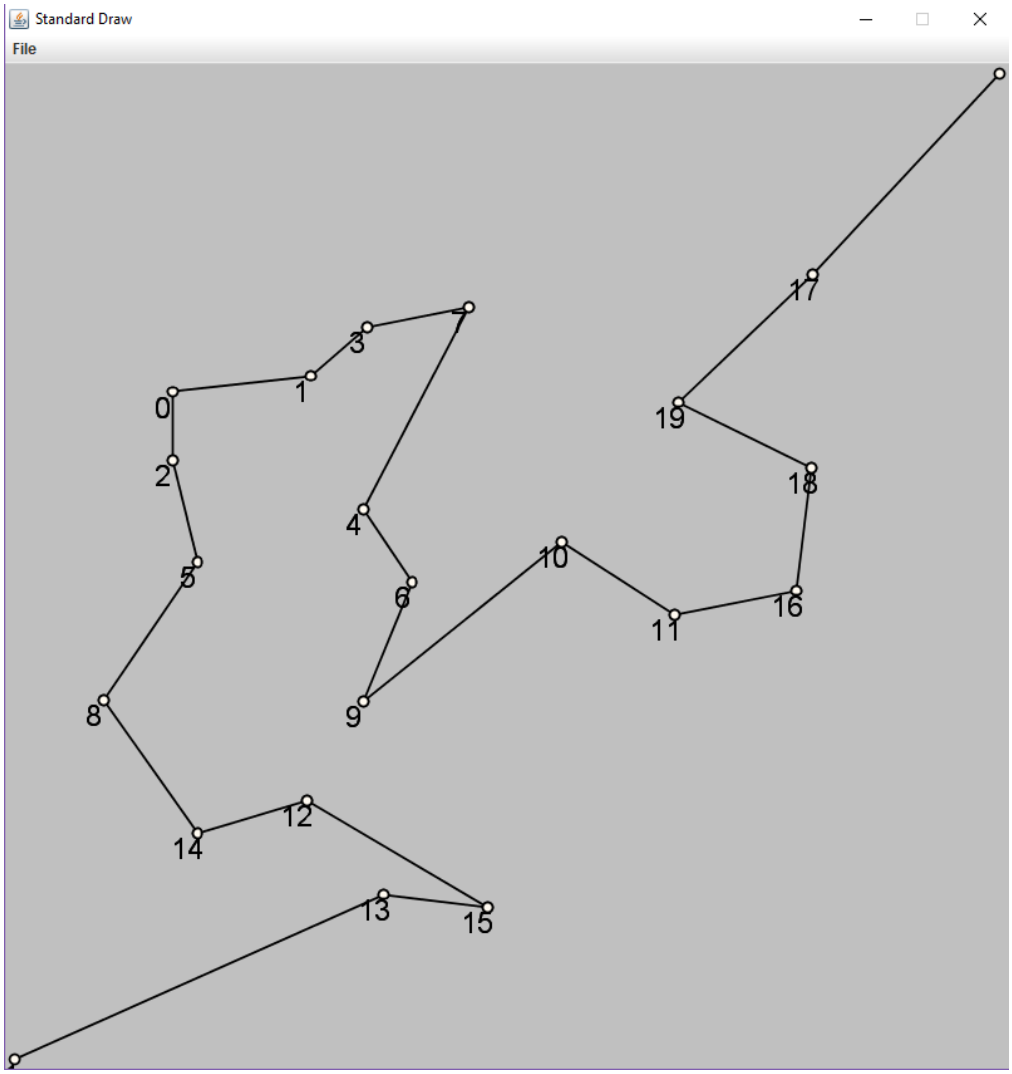
```
For (k = 0 to length of Path)
        For( i = 0 to length of cities)
                If(Path(k) = cities(i))            //gets the index of each city and sets it to ary
                    ary(k)=cities(i)
        }End For
}End For
Return ary
```

So this code simply matches the Cities in my path to ones in the cities array and fills my int array accordingly.

Exercise 2

The next exercise was for us to try an implement our Nearest Neighbour algorithm in another class and use case, Finding the shortest path for a printer head to travel to get to an array of points starting at (1,1) and finishing at point (0,0).

(a) Route code found: [Start, 17, 19, 18, 16, 11, 10, 9, 6, 4, 7, 3, 1, 0, 2, 5, 8, 14, 12, 15, 13, Fin]
(b) Path length: 3.392358
(c) Photo of path:



(d) Description of code:

```
City Start
City Finish
CityArray Path

For( b=0 to 1)
        If(b=1)  Start = (0,0)
                 Finish = (1,1)                    End if
        For( i=0 to length of points array)
                 Path = NearestNeighbourAlgo(i)          //i is the starting city
                 Path add Start and Finish
                 If (DistanceofPath < ShortestDistance)
                          =ShortestDistance= DistanceOfPath
                          BestPath = i
End for  End for End if
If (shortest path was found when b=0)
        Flip back start and finish                 End if
Path = NearestNeighbourAlgo(BestPath)
Path add Start and Finish
If (shortest path was found when b=1)
        ReversePath()                              End if
Print(CalculateDistancePath)
Draw(Path)
```

So basically, my code begins by running through all possible NN paths calculating the shortest one and saving its argument. Then it does this again with the starting points flipped. Then once it is done it overwrites the final temporary path with the NN path of the argument BestPath. Having to also do some clean-up to insure it has the right direction. I have left a lot of the nuances out of this pseudo code but it is all visible in my java file.

Exercise 3:

The nearest neighbour is of Order n^2 so

If $5^2*t=0.1$ $t=0.1/5^2=4*10^{-4}$

And therefore 20 points would take

$20^2*t = 20^2*4*10^{-4} = 1.6$ Seconds

Brute Force is of Order N! so

If $5!*t=0.1$ $t=0.1/5!=8.3333*10^{-4}$

And therefore 20 points would take

$20!*t = 20!* 8.3333*10^{-4} = 2.027*10^{15}$ Seconds

Which is 64 million years.