Main:
Sets up timer to stop program when running too long.

Gets the filename and calls readInput() which returns data (a list of tuples(city #, x cor, y cor))

Calls createGraph(data) to create an adjacency list. {city#: {neighbor1: distance, neighbor2: distance,....}....}

Calls nearestNeighbor(graph) to create a walk/tour of each city visited exactly once based on which city is nearest to the current city

Calls twoOPTTour() to improve tour if the current amount of time is less than the time limit.

Outputs time take and calls writeOutput()

Read Input:
Opens file

For line in file, each line has 3 ints: create a tuple (city number, x-cor, y-cor)

Add tuple to data

Close file

writeOutput:
Create and open a *"filename"*.txt.tour

Write the tour length
Write each element from the tour

Close file

CreateGraph: create an adjacency list using data

Make a distance table to hold distances from [source][destination] (initialize all to 0)

For each city in data

    Source = city

    For each city in data

        Destination = city

If source city and destination city are not the same

Check to see if there is a value already in the distance table, use the distance if present

If not call calculateDistance() to get the distance. Fill the destination table at [source][destination] and [destination][source] since it is the same distance both ways.

Append the (destination city: distance) to the source city neighbor list

calculate distance:
calculates the distance between 2 cities using the distance formula

Distance = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

nearest Neighbor: creates a tour by using the adjacency list (choosing the closest unvisited city)

Create order = [0] to hold the cities that have been visited in the order they are visited.

While all cities are NOT visited

Starting at city 0 , sort its neighbor list by distance (smallest to largest)

If the closest neighbor is already in order[] then look at next closest neighbor

If the neighbor has NOT been visited, append it to order and start process over (look for the closest neighbor to the city that was just added to order)

total length: adds up the distance for the entire tour

For each city, look up the distance from the adjacency list (graph) and add it to the total distance

twoOptTour: attempts to minimize distance by using a 2-OPT method

While 2-OPT is improving the tour and time has not run out

Create a new tour using 2-OPT

createNewTour
From 2nd city (m) to next to last city

From m+1 (n) to next to last city

Preform twoOptSwap (tour, m, n)
If the new distance is better than the old distance, update and return true

If all edges are checked with 2-opt swap and no improvements were made, return false. You have the best possible tour from this algorithm

twoOptSwap:

From : https://en.wikipedia.org/wiki/2-opt

```
2optSwap(route, i, k) {
    1. take route[0] to route[i-1] and add them in order to new_route
    2. take route[i] to route[k] and add them in reverse order to new_route
    3. take route[k+1] to end and add them in order to new_route
    return new_route;
}
```

Make a new tour:

Add tour from start city to m-1
Add tour from m to n in reverse order
Add tour from n+1 to end