Shae Judge
Brendan Byers

CS 325
Project 3 - TSP

1) **Nearest Neighbor**

With the Nearest Neighbor algorithm, a random city or vertex is first selected. Then you find the closest city or vertex closest to it and travel there. Then you continue to repeat the step of going to the closest city until all of the cities have been visited. Finally, you return to the initial city or vertex. This is the algorithm we ended up implementing in the final version of our program. Though it didn't return the optimal route, it figured out a cheap route in an extremely small amount of time. We traded off route length for extreme performance.

Pseudocode:

```
Initialize linked list unVisited
Initialize linked list visited
Foreach city{
        Set as struct
        Set city identifier
        Set x coordinate
        Set y coordinate
        Add to unVisited
}
startCity = random from unVisited
currentCity = random from unVisited
Remove currentCity from unVisited

while(unVisited != null){
        nearNeighbor = closest city to currentCity
        nearNeighbor.edgeLength = distance to currentCity
        Add nearNeighbor to visited
        Remove nearNeighbor from unVisited
        currentCity = nearNeighbor
}

Add startCity to visited
totalDistance = sum of edgeLengths in visited
Order of tour = path traveled in visited
```

2) **Brute Force**

Probably the worst algorithm to figure out the traveling salesman problem is the brute force algorithm. It simply tries every single combination of paths, then compares them all to find the best. This has a runtime of O(n!), where n is number of vertices.

Pseudocode

```
brute_force(route r):
if(!citiesnotinroute.isempty{
        For each city in citiesnotinroute
                New route r
                route.addcity(citiesnotinroute(city.index))
                brute_force(route)
} else{  //all cities are flagged, so route is full
        Bestroute = r
}
```

Time Results

3 Example Cases:

1. 133895 0.058455 ms
2. 3155 0.647468 ms
3. 1939181 2160.73 ms (2.16 seconds)

7 Competition Cases

1. 6382 0.0263ms
2. 8208 0.0775ms
3. 15985 0.5016ms
4. 19897 2.04914ms
5. 28399 7.19713ms
6. 40356 32.3978ms
7. 63769 703.285ms

All of our times were under a second, and though they were not the optimal solutions they were extremely quick. While some groups took up to 1048 seconds, all of our calculations were done in under a second. These times were also calculated on a virtual machine, removing it a step away from bare hardware and adding precious time to the calculation. The speed could be improved by running it directly on the processor, and possibly implementing threading or GPU utilization.