Algorithm Description

This algorithm starts the salesman at a random city in the tour and repeatedly visits the nearest city until all have been visited. We implemented this algorithm in Python 3.5. We first encountered the nearest neighbor algorithm when researching general TSP algorithms at **wikipediaTSP** . We learned more through it's specific Wikipedia article at **wikipediaNN** .

Algorithm Discussion

We chose nearest neighbor because it is essentially greedy which makes a good choice for being efficient and quick. We found that this algorithm satisfies the 1.25 requirement for most cases. Additionally, because it runs "fairly" fast for most input sizes, we were able to set it up to try each possible starting point. For larger input sizes we had the algorithm write results every time it found a newer/better route. This allowed us to find an acceptable route under 3 minutes (though there may have been better routes with unlimited time). For the three example cases 2-opt found routes between 15% and 22% over optimal. The algorithm runs in O(n^2) time.

Algorithm Pseudo-code

```
Nearest_Neighbor():
    cities = []
    // each city object has id, x and y attributes
    read data from input file into the cities array

    adjMatrix=[len(cities)][len(cities)]
    for i = 0 to len(cities) - 1:
        for j = i + 1 to len(cities):
            ij = dist(cities[i][j])
            adjMatrix[i][j] = ij
            adjMatrix[i][j] = ji

    min_tour_dist = MAX_INT
    min_tour_order = []

    for i = 0 to len(cities) - 1:
        tour_cities = copy of cities array values w/o references to orig array
        tour_order = []

        tour_order.append(tour_cities[i].id)
        tour_cities.remove(tour_cities[i])
        tour_distance = 0

        while tour_cities length > 0:
```

```
    cur_city = cities[tour_order[len(tour_order) - 1]]
    min_dist = MAX_INT
    min_city = -1

    for j = 0 to len(tour_cities) - 1:
        this_dist = adjMatrix[cur_city.id][tour_cities[j].id]

        if this_dist == -1:
            this_dist = cartesian dist between cur_city and tour_cities[j]
            adjMatrix[cur_city.id][tour_cities[j].id] = this_dist
            adjMatrix[tour_cities[j].id][cur_city.id] = this_dist

        if this_dist < min_dist:
            min_dist = this_dist
            min_city = tour_cities[j]

    tour_order.append(min_city.id)
    tour_cities.remove(min_city)
    tour_distance = tour_distance + min_dist
u = cities[tour_order[0].id]
v = cities[tour_order[len(tour_order) - 1]].id
this_dist = adjMatrix[u][v]
if this_dist == -1:
    this_dist = distance between cities[tour_order[0]]
                and cities[tour_order[len(tour_order) - 1]]
    adjMatrix[u][v] = this_dist
    adjMatrix[u][v] = this_dist

tour_distance = tour_distance + this_dist

if tour_distance < min_tour_dist:
    min_tour_dist = tour_distance
    min_tour_order = copy of tour_order's array by value

    write the output file to the path specified
```