Brandon Cohen

Input Size	Western Sahara (29)	Djibouti (38)	Zimbabwe (929)	Oman (1979)
NearestNeighbor	0 ms 36388.1 length	0 ms 9748.95 length	134 ms 117734 length	445 ms 120542 length
GreedyTSP	6 ms 56420.4 length	1039 ms 1.1912e+06 length	815 ms 1.1912e+06 length	4597 ms 4.18069e+06 length

Log of Computations:

- Western Sahara has length 27603 in 0.09 secs
- Djibouti has length 6656 in 0.23 secs
- Zimbabwe has length 95345 in 544.65 secs
- Oman has length 86891 in 9809.20 secs

In the nearest neighbor implementation, I used a Node struct with three data members: ID, x-coordinate, and y-coordinate To solve the problem, I made an adjacency matrix, a vector of integers representing the tour with their ID numbers, and a vector of bools that represent if a node was visited. Since I used an adjacency matrix, the algorithm iterates over all pairs of nodes which means the time complexity is $O(|V|^2)$, where $|V|$ is the number of vertices. One problem to keep in mind is that as the graph grows, the algorithm's performance will decrease significantly which means it is not a good algorithm for representing many cities, but it is a very good algorithm for simplicity.

In greedy TSP, I used the same Node struct as in nearest neighbors but I added a few extra data members: is_in_matrix (representing if a node is in the node matrix), matrix_index (representing where a node is in the matrix index), and num_edges (representing the number of edges a node has). I also used an Edge structure consisting of three data members: from (pointer to a Node), to (pointer to a Node), and weight (distance between from and to). I used a < operator to make comparisons between the edges. The goal of the algorithm was to create all unique Edges for all nodes where each Node* will be added to a node matrix as long as it wasn't there previously. Once the node matrix is made, the distances between the ends are calculated to determine where to merge each vector to end up with one tour in the node matrix. The time complexity of the algorithm was $O(|E|\log|E| + V^2)$, where $|E|$ is the number of edges and $|V|$ is the number of vertices. The $|E|\log|E|$ comes from sorting the vector of edges and the V^2 comes from creating the edges which requires iterating over the vertices through a double for loop. This algorithm is useful when using a sparse graph because this algorithm becomes $O(|V|^2)$.

When comparing both algorithms to each other, I believe there must have been an error during the merging of small vectors in the node matrix because the length of the greedy TSP is

consistently greater than the nearest neighbor and this should not be the case if greedy TSP is implemented correctly. Based on the log of computations, the nearest neighbor and greedy TSP were consistently much faster than the most optimal solutions which are expected since the most optimal solutions will find a better result with the cost of a longer running time. In comparison to the Western Sahara, the nearest neighbor and greedy TSP had a larger length which is expected since these algorithms could be more optimized. The same was also found in Djibouti, Zimbabwe, and Oman. I learned from this project that a lot of focus has to be put into the implementation. The algorithms are not conceptually difficult but if you do not use the best data structures to represent the data, then the graph algorithm will not be effective or efficient. I also learned that many optimization methods can be used to improve a graphing algorithm since my results were significantly worse than the most optimal solutions.