My approach to the Traveling Salesman Problem was first choose the starting node, lets call it v, from a complete graph. Then add v to a set of known visited nodes, find the shortest path between v and its non-visited neighbors, and select the node on the receiving end of the shortest path. This node will now be added to the set of visited nodes and then its neighbors will be checked. This approach addresses greedy strategies. By starting at the first node, we have a chance to limit the amount calls to find the shortest path, but it is not always guaranteed we find the optimal path. Our next node will always be the shortest edge among the the unvisited neighbors due to local choices being made on which node to go to next.

My graph is stored within an adjacency hash in the form of a dictionary within a dictionary. By utilizing a python dictionary, we can get, set, and lookup values in O(1) time. The complex part it looping over the size of the graph, and checking all neighbors of the current node. In the worst case, a complete graph, the size of the neighbors will be equal to that of the graph, excluding the current node. This causes a worst -case time complexity of $O(n^2)$ where n is the number of nodes.