Strategy for the TSP Approximation Solution:

- The strategy first employed randomness to select a starting node in the initial tour
- Then, a greedy approached was used by sorting the neighbors of the current vertex and selecting the nearest unvisited one.
- Lastly, the strategy returned to a random approach by using simulated annealing to make small, random improvements to the tour generated by the greedy approach.

Walk Through and Time Complexity Analysis of the TSP Approximation:

- The main method begins by parsing input from files
    - Information is parsed into variables vertices (the number of vertices), edges(the number of edges), u (current node), v (next node), and weight (the edge weight between u and v)
- Three dictionaries are instantiated and are called graph (key is the index of u in vertex map, the value is another dictionary where that key is the index of v in vertex map and the value is the weight), vertex_map (key is the vertex, value is the index of that vertex), and rev_map (a reverse of vertex_map)
- There is a for loop which runs for every line of input to fill in the three dictionaries. Its runtime can be simplified to be **O(n^2)** because of the formula for number of edges in a complete graph.
- **There is a while loop which runs for 45 seconds using a timer.**
- Within this while loop, it first makes a random, greedy tour. This brings is to the nearest_neighbor_tour() function
- The nearest_neighbor_tour() function picks a random starting vertex and then uses a greedy approach of sorting the unvisited neighbors of the current vertex by edge weight and going to the nearest one first. This function has a runtime of **O(n^2)** because it has a for loop which loops through all of the vertex's neighbors (n-1) and it sorts those neighbors within that loop (n).
- After that tour has been made, it is passed as a parameter to the adjust function.
- The adjust function begins by copying the tour that is created by the greedy_tour() function into a list called best_tour. Then there is a for loop which run n-3 times. We will round this to **running n times**, though. It keeps track of a variable i.
    - Within the for loop, there is a while loop which runs while a boolean variable called improved is true. This can be **up to n times, but most likely much less**. For worst-case runtime, we will assume n.
        - Within the while loop, a variable j is set to a random number within the range of the tour
        - A new tour is created which is a copy. The new tour swap I and j and then calculates the tour weight after this swap. **Calculating the tour weight takes n time**.
        - Then the new weight and old weights are compared and the best tour is saved based on this comparison.

- Next, in the main, the tour weight is calculated using a function that has a for loop that runs n-1 times. So, O(n).
- Based on this, the time complexity class is **O(n^3)**, but it is within a while loop based on a timer.