Design Architecture:

I designed my program to have a factory pattern as well as a strategy pattern. I chose to use a factory pattern since it allows me to add another type of algorithm, like a sorting algorithm, to my program without having to change a lot of code. I implemented the factory pattern with the virtual Algo.h class that is implemented by the TSPAlgo class. The TSPAlgo class is where the file is loaded in, output stats are printed to the console, the desired algorithm is selected, and the selected algorithm is executed. I chose to create a strategy pattern for the TSP algorithms since both algorithms had similar functionality that could be abstracted behind a class. I chose to create TSP.h that is implemented like a strategy pattern. Both my dynamicProgramming and Naïve class implement the TSP class. This will also allow me to add more TSP algorithms to the program without having to change a lot of code.

Results:

The graph above shows the execution time of the dynamic programming algorithm and the execution of the naïve algorithm versus the number of nodes in the graph. One thing I found to be interesting is the naïve approach is actually faster with 4 nodes than the dynamic programming approach. The naïve approach starts to slow down rapidly after 5 nodes. I believe the reason that the naïve approach is faster for 4 nodes is due to the fact that the dynamic programming approach has the time complexity of 2^n \* n^2 which is greater than n! at smaller numbers.