**Project 1**

Danny Foley

Computer Science and Engineering

Speed School of Engineering

University of Louisville, USA

[danny.foley@louisville.edu](mailto:danny.foley@louisville.edu)

**Introduction**

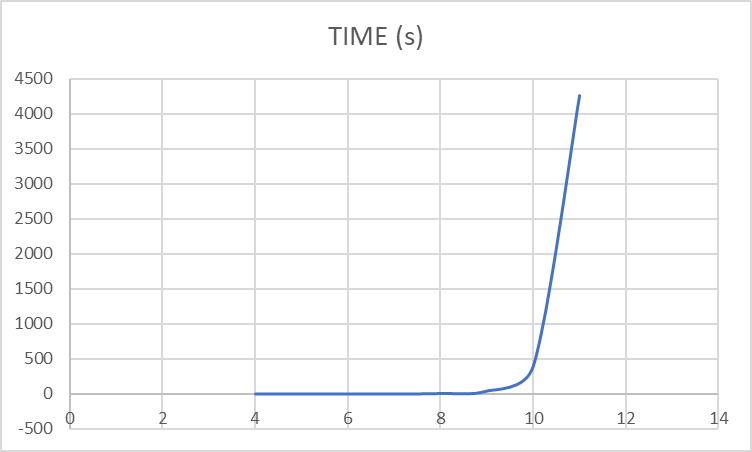
In Project 1, the assignment was to compute the shortest possible route from the Traveling Salesperson Problem. The traveling salesperson problem is a common problem where it is representing an actual real-life issue where a salesperson would need to figure out the best way to travel to each city in the area so that they can reduce time and costs.

**Approach**

For this project, a brute force method is to be taken. In my project I decided to use a method to go through all possible routes. Using a built-in permutation function to create all possible permutations of a vector, I was then able to create all possible routes. If the function can determine the next higher permutation, it rearranges the elements as such and returns true. If that was not possible (because it is already at the largest possible permutation), it rearranges the elements according to the first permutation (sorted in ascending order) and returns false.1  
 Once the route distance was calculated it was then compared to the current shortest route, if the new route was shorter then the new route would replace the old route. Given that the route itself is a circuit, I only made the program begin the route at city 1, because no matter where you start the ending best route would always be the same. This means that instead of having n! possible routes (n = number of cities), there would then be (n-1)! possible routes.

**Results**

The algorithm performed as expected with a constantly increasing performance time with each additional added city. The data used was from provided .tsp files that contained information on the number of cities and their Euclidean coordinates.



**Discussion**

These results were expected. Based on the number of permutations for each number of cities it would make sense that each additional city would multiply the previous by how many cities there were before because of the (n-1)!. With 7 cities, the execution time was 0.843 seconds and then with 8 cities the execution time 6.482 seconds. That is a 7.689x time increase, which is very close to 7 times. When there are 7 cities the number of routes checked is 720 (6!), then when there are 8 cities there are then 5040 routes checked. The trend continues where the difference between 9 and 10 cities is a 9.5x increase in time and then the difference between the 10 and 11 is 10.7x increase. With these numbers that would mean that 12 cities would take around 49,070 seconds or around 13 to 14 hours. Due to time constraints, I was unable to check to see if this was correct in practice.

**References**

[1] <http://www.cplusplus.com/reference/algorithm/next_permutation/>

<https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/>