Brian Barbu (brb9da)

Post-Lab 9

Complexity Analysis:

The structure of my topological program in my pre-lab and of my traveling program in my in-lab determines the runtime complexity of their execution. The type of variables being processed and the size they hold in memory all influence runtime. In the topological code, 2 strings are read at a time and then pushed onto a vector. The vector is iterated through. If n is the number of strings, these tasks take Theta(n) as everything needs to be iterated through once. Elements from the file are read and pushed onto a list taking Theta(n). Then sort is called arranging the list. Every element only needs to be accessed once in the loop and this process also take Theta(n). Lastly, the list is iterated through and printed which take Theta(n). This program has a list of n strings. Each string is 4 bytes. It also has a map object which is a list of strings matched to strings of a vector. This results in 8 bytes for n elements in a map object.

The structure of my traveling program is also the determinant of its complexity. The program instantiates one middle earth object which has a constant vector of cities. The middle earth object does not change and can be directly accessed so uses a Theta(1) run time. Another vector is created to hold the correct cities, also in constant runtime. The dests vector is looped through to compute the distance of the trip. This method runtime is Theta(n) for n cities in the vector. Next_permutation() analyzes every combination of city routes which must take Theta(n!) runtime for all possibilities. With the trip sorted, its printed once per element take Theta(n) in runtime. In terms of memory, a middle earth object has a vector containing n cities strings of 4 bytes each, a vector of 4 byte elements called xsize, a vector of 4 byte element called ysize, and 4

ints which are 4 bytes each. I also create a vector holding 4 bytes for each element of the cities to consider. There is another vector also holding 4 bytes for each element of the path of cities. The string holding the starting city is 4 bytes. The float for the minimum distance is 8 bytes.

Acceleration Techniques:

There are a few interesting methods I researched to improve implementation of the traveling salesman problem. The Branch and Bound algorithm is an acceleration technique designed to break down solutions and replace them with better ones. It splits the range of data into smaller ranges and compares them to the bounds of the best solution. The algorithm slowly tweaks and improves the best solution through all of these comparisons. This allows the algorithm to work on the whole spectrum of data and optimize the solution by better estimating at the smaller intervals.

A good example of an approximation technique is the nearest neighbor algorithm. This algorithm decides the path by always moving to the next closest location. For data that is very linear, this method of approximation can be very beneficial for being a simple method that can always pick the most optimized choice from consistent reliable data. Implementation should only take linear time in the worst case. However, in a large collection of data, this method may be incomplete. It may be quicker to take a long path to reach another node closer to the final node, rather than always moving to the first closest node. This technique is great for its simplicity and effective in the right scope, but has obvious limitations.

Another technique I found interesting to solve this problem is the Ant Colony Optimization technique. It was inspired by the structure of real life ant colonies. The main idea is to build a collection of structures to optimize the full program. Structures will be designed to collect

relevant data and search based on types of structures containing this data. There is trail which the algorithm will detect and process more efficiently depending on what has already been done. This techniques all approach the traveling salesman problem differently, but can all be the best at optimizing with the right circumstances.