Lab on the TSP

Names of group members: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Purpose: To get some practical data about how large of a traveling salesperson problem can be solved by exact methods. Also, to see how quickly the running time increases as we increase the number of cities.

1. Work in groups of 2 or 3. You can divide up the work, compare the results within your group, etc.

2. Copy the supplied folder of TSP materials to a location where you have write access. Among the files contained within this folder should be one named TSPLab.docx. That is the file containing these directions.

3. Open your copy of TSPLab.docx.

4. Go to your folder of TSP materials. Extract the data files from tspdatafiles.zip. Next, if the .cpp files have not been compiled, you will need to do so. They were designed to be compiled with Visual Studio 2015 on 64 bit Windows machines, but you may have luck in compiling the code in other settings. Then run the program tspcombined.exe by double clicking the file. (If you are not using Windows, tspcombined.cpp will probably not work, but you could instead use the other three programs, though you will have to time their execution manually in order to enforce the 5 minute time limit discussed below.)

 The tspcombined.exe program will ask you for the filename prefix for the data files. Use tsp.d for this as the data files are named tsp.d3, tsp.d4, etc. These files each contain two things: first, the number of cities, and then the matrix of costs between pairs of cities. (You can open one of the data files to see what it looks like.) You will next be asked for the number of data files to do. Try 16 as that is the number of data files supplied in this folder. Then it asks for a time limit. This is in seconds. Use 300 seconds (which is 5 minutes). Then let the program run until it quits.

 If it appears that you can handle more than 16 data files, you can copy the last data file tsp.d18 into tsp.d19 and edit it so that it says that there are 19 cities. Then you must adjust the matrix so that it is 19 rows by 19 columns. Note that this matrix of costs must be symmetric. Try again with 17 as the number of data files and the same 300 second time limit.

5. The program will produce output on the screen as well as in a report file named report.txt. All of the group members should then work on analyzing the data. This data should show that the software ran 3 different algorithms on each TSP problem, where each new problem has 1 more city than the previous one. The algorithms are named tsp1, tsp2, and tsp3. Tsp1 is the usual brute force method of enumerating all the tours of the cities and adding up their costs. Tsp2 is the same but adds a slight improvement by not recalculating partial sums of costs when some tours start out with the same sequence of cities. Tsp3 adds a "branch and bound" technique that quits adding up the cost of any tour when the partial sum of the costs is already the same as or greater than the lowest cost already found for a complete tour. We mostly analyze the data for tsp1 and tsp2 in this lab, but look at the tsp3 results to see how they compare with those of the other two methods.

6. How many cities was tsp1 able to handle (within the given time limit)? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 How many cities was tsp2 able to handle (within the given time limit)? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 How many cities was tsp3 able to handle (within the given time limit)? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. Next, we graph the **number of additions** used by tsp1, tsp2, and tsp3 to get an idea of how fast the running time grows, for each of the three algorithms, as the number of cities increases. We already know that the number of additions for tsp1 is given by the factorial function, so we expect a rapidly increasing function.

 Due to this rapid growth, we actually want to plot on the y axis the log (base 10) of the number of additions. One way to do this is by using semilog graph paper. Start with 3 cities. The number of cities goes on the x axis (normal scale). The number of additions goes on the y axis (logarithmic scale). The **lowest** line crossing the y axis should be labeled 1, the next 2, and so on. When you reach 10 you should again get to lines that are further apart. Successive lines after 10 are 20, 30, etc. up to 100. We then hit a large gap after the 100, etc. Thus, the largest gap is between the 1 and 2, while the smallest gap is between the 9 and 10. Then in the next grouping, the largest gap is between the 10 and 20, while the smallest gap is between 90 and 100.

As an example, suppose that you want to plot a y value of 21. You might think that you put this point 1/10-th of the way between 20 and 30, but it should go somewhat higher than that, since the logarithmic scale starts with large gaps followed by smaller and smaller gaps. By plotting numbers this way you are really getting the approximate log base 10 of the number on the y axis.

In order to fit all of your data, you may need to tape together a few sheets of semilog graph paper. The top horizontal line of one sheet and the bottom horizontal line of the sheet above it should coincide exactly, one on top of the other.

Another way to produce the graphs is to use a calculator to find the log base 10 of each number of additions. Plot this data on normal graph paper with the number of cities on the x axis and the log of the number of additions on the y axis. Label the y axis 0, 1, 2, 3, etc. with successive integers being the same distance apart. The x axis is again for the number of cities. Start at x = 3 cities and go as far as you can until you either run out of data or you cannot fit more on the sheet of graph paper (or a few sheets carefully taped together.)

A third way to produce the graphs is to use spreadsheet software to imitate what was just described in the other two methods.

8. For the graphs for tsp1 and tsp2, decide if they give a straight line or if the graph curves slightly upward or downward (concave up or concave down). (Suggestions: You can put a ruler along the points of the graph to see if they line up in a straight line. You can also place the graph on a tabletop and with you head at about tabletop height sight along the points to see how well they line up.) This will tell you if the log of the number of additions grows faster than a straight line, exactly as a straight line, or slower than a straight line. (Hint: It will be hard to tell among these 3 possible answers due to how the logarithm compacts the data on the y axis. The graph is either a straight line or slightly curved.) Write your conclusions here for the two graphs.

 tsp1:

 tsp2:

9. When plotting the log base 10 of y versus x (as above), what does it mean mathematically if the graph is in fact a straight line? (Hint: solve for y in the equation log(y) = mx + b.) Write your answer here, showing your calculations. (If need be consult a math or physics major!)

10. What do your answers in parts 8 and 9 imply about solving the TSP with the **tsp1** and **tsp2** program?

11. Calculate how long it would take on a personal computer to solve a 30-city TSP using **tsp1**. Assume that the time is roughly proportional to the number of additions. We already know that the number of additions for tsp1 is given by n!. Let's assume that our software gets to use 3.2 billion processor cycles per second. However, it may take several cycles to do each operation, such as addition. Let's assume that each addition can be done in 8 cycles, as a rough guess. That means that our computer can do 0.4 billion additions per second. Use this to calculate the amount of time needed to solve a 30 city TSP. Put your answer into minutes, hours, days, or whatever units are most convenient to understand the result. Write your calculations and answer here:

What are the implications of your answer for solving a 30-city TSP using **tsp1** on a computer with this speed?

How would your answer change if we got a supercomputer or computer cluster that is 10,000 times faster than the computer just described? Give the resulting amount of time here (in appropriate units):