Laboratory

17

Limits of Computing

Objectives

- Investigate the growth of mathematical functions.
- Watch the Traveling Salesperson find the best route.

References

Software needed:

1) A web browser (Internet Explorer or Netscape)
2) Applets from the CD-ROM:
   a) Comparison of several functions
   b) Plotter
   c) Traveling Salesperson Problem

Textbook reference: Chapter 17, pp. 528–534, 557–558

Background

Everything you need to learn is explained in Chapter 17, "Limitations of Computing."

Activity

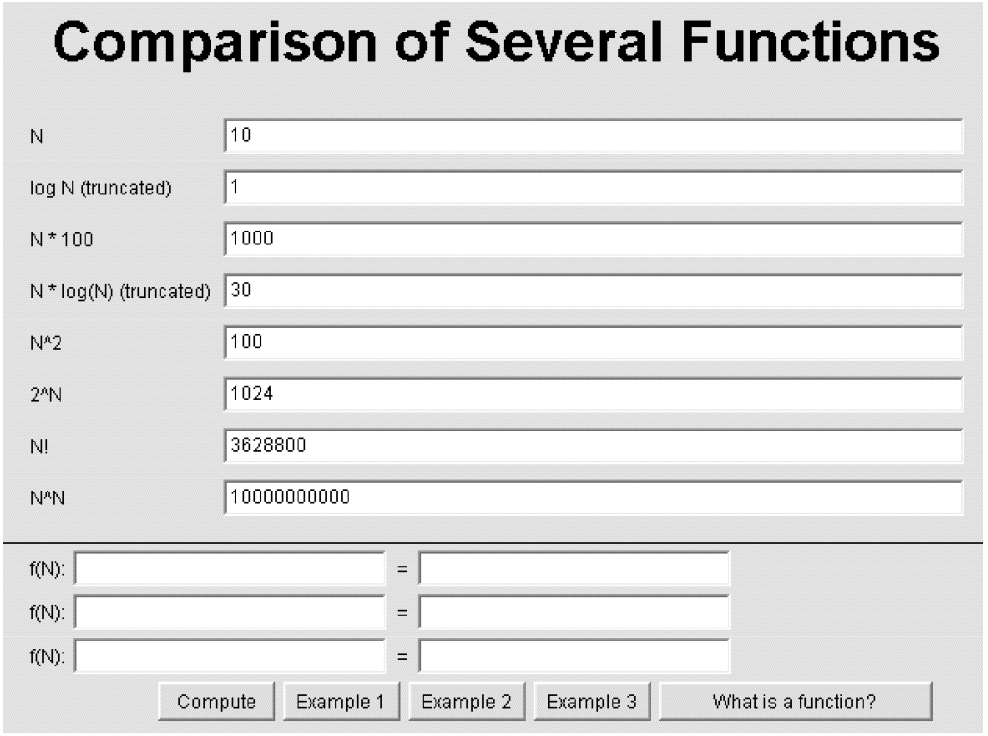
Part 1

Chapter 17 discusses a fascinating variety of limitations on the power of computers. Many people seem to believe that computers are all-powerful, but this chapter will give you the intellectual tools to discard that notion. (Of course, we would not deny for a moment that computer *scientists* are all-knowing and all-powerful!)

For our first investigation, we'll use the "Comparison of several functions" applet to watch how large the values of functions can get. Start that applet and type `10` into the top field, next to N. The applet will calculate several functions of N and display their values. (See the screenshot below.)

Review the values in the white text areas and try to make sense of them. N*100 should be easy; just multiply the value in the N box by 100. The *caret symbol*, as in N^2, is used in some computer programs to signify exponentiation. Thus, N^2 means "raise N to the power of 2" or "multiply N by itself." We would normally see this in math textbooks as N^2. 2^N is quite a different matter! You can see that 2^N is usually much larger than N^2, so the exponentiation operator is by no means commutative. In this case, 2^N means "raise 2 to the power of N."

The logarithm function, notated log N, unnecessarily scares some people. All it really does is count the number of 0s after the 1, if the number is base 10. Thus, log 100,000 is 5. Another way of thinking about logarithms is to imagine they are exponents standing on their head. 10^5 is 100,000. Not so scary, huh?



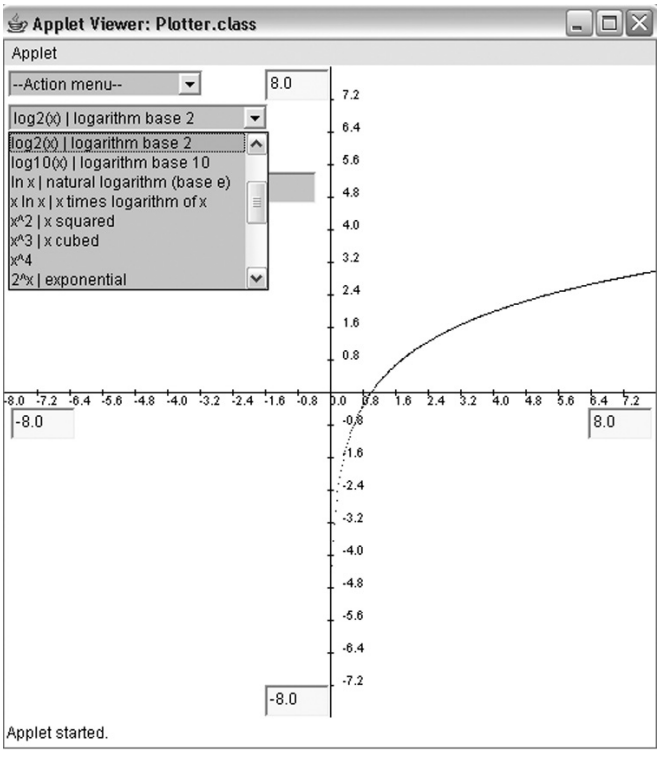
Part 2

Functions are interesting creatures. Some are tame and predictable while others are wild and exotic. Mathematicians capture and study them as if they were rare reptiles. Computer scientists, on the other hand, are interested in functions because they summarize the behavior of algorithms.

Every algorithm uses both time and space. It takes a certain number of instructions to find a solution to a problem, and the program needs a certain number of memory cells to find that solution. Functions describe these numbers in terms of how big the problem is.

In Part 1, we looked at functions numerically, which gave us insight as to how fast they grow. In this part, we use an applet that lets us draw plots of functions on an X-Y coordinate graph. The resulting lines and curves are visualizations of function behavior.

Start the "Plotter" applet and notice that there are the usual X and Y axes. The second pull-down menu lets you choose a function that has already been programmed in. Below shows the function log2(x), which is the logarithm base 2 of x, where x is a value on the X axis.



You can plot more than one function at a time. Doing so allows you to compare functions. For example, plot these functions together:

2x

xˆ2

2ˆx

The notation xˆ2 means "x raised to the second power" which is x times itself. You are more likely to see x^2 in a mathematics textbook. The caret symbol is sometimes used in place of superscripts, which are tricky to type on traditional keyboards.

The "Action" menu has a number of helpful options, including Help and Clear. Clear removes all currently plotted functions. You can also hide the axes, the yellow bounds boxes, and even the other boxes.

The yellow bounds boxes at the edges of the X and Y axes allow you to type in a new value. The top box holds the maximum Y, while the bottom box holds minimum Y. The left one holds minimum X and the right one maximum X. You can zero in on a particular part of the plot by appropriately setting those values.

Sometimes functions change so quickly that the line or curve is shown as a sequence of dots. These are the individual plot points representing the y value that the applet calculated for the X value underneath. To make the lines or curves smoother, increase the number of plot points in the box. Don't be afraid to make it exorbitantly large, like 30,000. Computers are so splendidly fast that you won't see a slowdown.

While the functions that this applet includes are interesting, it would be nice to enter your own, which you can do in the textfield labeled "Your formula:" There are two kinds of formulas you can type:

polynomials	$5x^3 - 3.2x^2 + x - 2.1$
Fourier series	$3\cos x - \sin 0.5x + 4.7\cos 13x$

After you have typed in a formula press *return* in the box. You can only plot one user formula at a time. There are four example formulas that you can select from the Choose Function pull-down.

Polynomials are powers of x, multiplied by real numbers. The powers can be integers, fractions, or negatives. However, if they are negatives, they must be enclosed in parentheses. Since you can't type superscripts in a textfield, use the caret symbol. Here are some valid polynomials that this applet accepts:

5x^3-3.2x^2+x-2.1	
x^0.5	(the same as the square root of x)
x^(-4)	(negative powers require parentheses)

Fourier series are sums of sines and cosines, which are periodic trigonometric functions. If all this terminology scares you, then just enjoy the beautiful pictures that mathematics paints.

First, clear all functions. Then select sin x from the pull-down function menu. No, that is not an exhortation to do something bad! Shame on you for thinking it! In fact, it is pronounced "sign-X" just like street sign. In full, it is spelled sine. cos is short for cosine, which is what you do to help a friend get a loan…no, wrong verb! Sorry. tan is short for tangent, which is what this book often goes off on!

All three of these trigonometric functions are *periodic*, which means they repeat themselves over and over forever. Sine and cosine are nice undulating waves, like the tide on the beach. In fact, sine and cosine have exactly the same shape. The only difference is that they are displaced differently on the X axis. If you could slide cosine so that it overlaps sine, you would see only one wave.

Tangent is a strange duck because, though it is periodic like its two prettier cousins, there are sharp breaks. It is not *continuous*. There is not one unbroken curve. Instead, every so often the tangent line veers off toward infinity or negative infinity and there isn't anything you can do about it.

Jean Baptiste Joseph Fourier, who was a child during the American Revolution, discovered that every periodic function, even those with warty or boxy shapes, could be broken down into a sum of sine and cosine functions. With the appropriate values, and enough sines and cosines, any period function can be displayed.

The applet's Example 4 is a sum of two cosine functions that gives a mildly interesting curve. To see it better, type 32 in the maximum X box and −32 in the minimum X box. Also change the number of plot points to 3000. Can you see the periodicity of the function?

Here are some Fourier series that this applet accepts:

sin x

cos x

5sin x + 2.6cos x

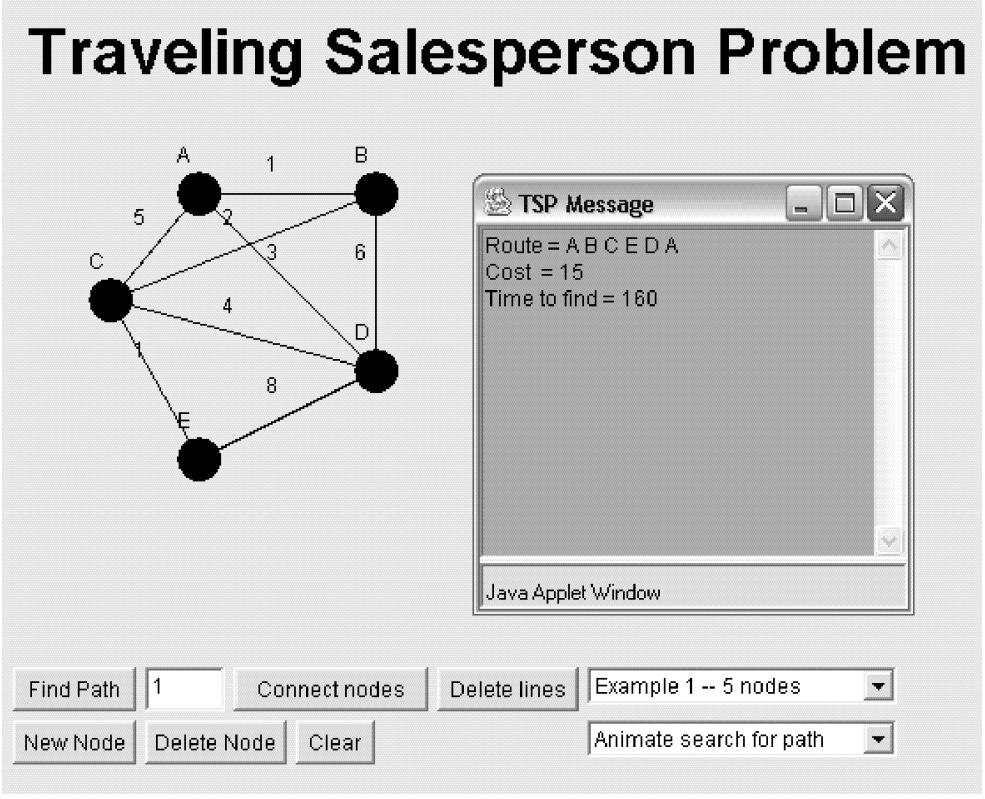
sin 0.5x − 3.8 sin 17.4x

In summary, this plotter applet allows you to visualize some functions and to compare their behaviors graphically.

Part 3

The Traveling Salesperson Problem (TSP for short) is one of those NP-complete problems that seem to take an inordinate amount of time to solve. It is explained on p. 538 of your textbook, but we'll summarize it briefly here.

First, start the TSP applet and click on *Load Example*. This brings up a five-node graph. Click on *Find Path* and a salmon-colored window shows the most efficient (shortest) path for the salesperson to follow when visiting all the cities.



Think of the graph as a map that shows cities A, B, C, D, and E. Some cities have a two-way road between them, with a distance number shown on the road. The distances shown above are not necessarily in scale with the length of the line, but that's okay—graphs are abstract creatures and live by their own rules.

The salesperson must visit all the cities, and wants to save gasoline as well. By starting at city A and going to each city exactly once, ending up back at A, the salesperson can accomplish the first goal, and to some degree the second, since at least he or she will not be backtracking and going through a city more than once. A path that visits all cities exactly once is called a *tour* or a *circuit*.

But wait, the boss has just added a third restriction! There may be several possible tours, but the boss wants the salesperson to use the shortest one. Such a tour is called a *Hamiltonian circuit*, and this is what the TSP applet finds, if there is any possible circuit at all.

When you run the applet, notice the cost measure it calculates. This is the sum of all the numbers (called *weights*) on the lines between the cities. A Hamiltonian circuit has the smallest cost measure of all the circuits (if there are any). Also notice the "Time to find" value, which the applet reports underneath the cost. This is a measure of how many comparisons the program made in order to find the shortest circuit. Notice how it goes up dramatically as you add nodes and try to find a Hamiltonian circuit in a big graph.

While it is easy for us to find all the circuits in the previous graph, and therefore not too taxing to find the Hamiltonian circuit, larger graphs make it harder and harder, both for us and for the computer. While the computer (thankfully) doesn't complain of eye strain or overwork, it *does* take a long time to find the Hamiltonian circuit. By the time we get up to the relatively small number of, say, 40 cities (and, of course, there are many more than 40 cities in the United States, or even in most states), the amount of time the computer needs to find the Hamiltonian circuit is so large that all the protons of the universe will disintegrate first! (Physicists believe that protons will last only 10^{34} years, by the way.)

The fact that it takes so long to solve this problem is really unfortunate, because it would be great to find the best possible solution to such a problem. There are other problems similar to the Traveling Salesperson Problem that we would like to solve, but no one knows of a faster way to solve them. Even worse, no one has been able to prove that no faster solution is possible! This leaves open the tantalizing possibility that an algorithm may someday be found, which would revolutionize a lot of things. But computer scientists have been trying to find a faster algorithm for over 30 years, and they're no closer now than they've ever been.

The TSP applet also allows you to add new nodes to the existing graph. Simply double-click on the empty area and a new node with a one-letter name will appear. To connect it to other nodes, set the weight for it in the small text area (it is currently 1). Then click on *Connect nodes*. Click on one of the nodes and a line will follow your mouse pointer. Click on the destination node to complete the line. Remember the lines represent two-way paths, so you don't need to be concerned about which node you click on first.

You can repeat this process for any number of nodes. Click on the *Stop connecting* button (which is the same as the *Connect nodes* button, with its label changed) to end the process. You can also remove all lines from a node by clicking on *Delete lines*. To get rid of all the nodes, click *Clear*.

Tip

TSP can be used as a standalone Java application. If you use TSP as an application (not as an applet), you can load and save your graphs. To run the Java application, navigate to the folder containing the TSP class files and double-click on the **run_application.bat** file.

Exercise 1

Name _____ Date _____

Section _____

1) Start the "Comparison of several functions" applet and click on the *Example 1* button. The applet runs through a bunch of N values and puts the corresponding f(N) in the text areas. Write down what value N has when the N^N text area first says "too big!" Do the same for the N! and 2^N text areas. You may have to click on *Example 1* several times, watching carefully for when the fields change to "too big!"

2) Type the following function into the first f(N) text space:

   log(log(N))

   (There's no mystical significance to this formula; we're just using it as an example.) Then type in several values of N, one at a time, and press the *Compute* button after each one. Now see if you can get the value 1.0 to appear in the result area by finding a value of N such that log(log(N)) approximately equals 1.0. (Hint: Think BIG! You may not hit 1.0 exactly but you can come close.)

   Take a screenshot when you've found it.

3) Let's experiment with three functions. Click on *Example 3* and three functions will be inserted into the three f(N) text areas.

4) Take a screenshot, print it, and write the order of magnitude in big-O notation next to each of three functions. (See p. 547 of your textbook.)

5) Type `1` into the N text area and press *Compute*. Write down the values of the three functions. Which function gives the biggest answer?

6) Type 3000 into the N text area and press *Compute*. Write down the values of the three functions. Which function now gives the biggest answer?

7) Finally, find a value for N that causes the third function to give the biggest answer.

8) Write down your observations as to the values that these three functions produce for given ranges of N. Is one function always larger than the others? Why or why not?

Exercise 2

Name _____ Date _____

Section _____

1) Start the "Traveling Salesperson Problem" applet.
2) Pull down the examples list and select "Example 1 — 5 nodes."
3) Before the applet finds a path, try to find one yourself. Either list the nodes in order from A back to A, or take a screenshot and highlight the path with a pen.

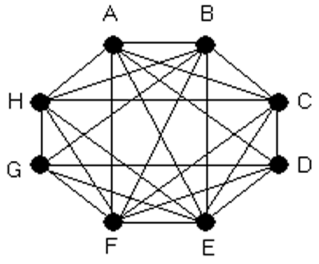
4) Now click on the "Find Path" button and watch the applet search. Write down the path it found and the cost.

5) Your path might be different from what the applet found. Make a guess as to why the applet found the one it did. (Hint: think of alphabetical order….)

6) Are there only two circuits in this 5-node network?

7) Is the network shown in this 5-node example a directly (fully) connected network? (See Laboratory 15.) To refresh your memory, a directly connected network is one where there is a single wire between any two nodes.

8) Following is a picture of an 8-node directly connected network.



Why is it very easy to find a circuit in a directly connected network?

9) Are there many circuits in a directly connected network? Can you list several others in the 8-node network above?

10) What other network topologies have easy-to-find circuits?

Exercise 3

Name _____ Date _____

Section _____

1) Start the "Traveling Salesperson Problem" applet.
2) Pull down the examples list and select "Example 2 – 12 nodes."
3) Before the applet finds a path, try to find one yourself. Either list the nodes in order from A back to A, or take a screenshot and highlight the path with a pen.

4) Now click on the *Find Path* button and watch the applet search. The applet automatically animates its search by coloring tentative path segments in yellow. Watch the applet run for a few minutes. Did it find a path in that time?

5) Now stop and restart the applet (which you can do by pressing the BACK button on your browser or by closing your browser and beginning over). Select "Example 2 – 12 nodes" and select "Don't animate search." Click on *Find Path*. Write down the path it found and the cost, as well as the time to find it.

6) Did it find the same path that you did? Why did the applet choose the path it did rather than yours?

7) Add one new node and connect it to 3 other nearby nodes. Then click on *Find Path* again and write down the time it took. Was the time about the same as Step 5 above or was it a lot more?

8) Make a guess as to how the time to find increases as you add nodes. (Don't worry about being mathematically precise or figuring out the big-O function.)

Exercise 4

Name _____ Date _____

Section _____

1) Start the "Plotter" applet. Select these functions from the "Choose Function" pull-down menu:

   x^2

   x ln x

   (x^2 is "x squared" or "x multiplied by itself." x ln x is "x multiplied by the logarithm of x.")

2) Which one grows faster? That is, as x gets larger and larger, which function's y value gets larger than the other's? The faster-grower's curve will be above the other one's curve.

3) Take a screenshot and label the two curves, since the Plotter applet doesn't do that for you.

4) Suppose that you heard that Algorithm A's running time function was x^2, meaning that if the problem size is k, then the number of time steps it would take to find the solution is k times k. Suppose that Algorithm B's running time function was x ln x. If both algorithms solve the same problems, such as "sort a list of numbers," which one should you use?

5) Find a function that can be called *linear*, that is, whose plot is a straight line.

6) Why should we try to find algorithms that have linear functions for their running times?

Exercise 5

Name _____ Date _____

Section _____

1) Start the “Plotter” applet. Select these functions from the “Choose Function” pull-down menu:

   x^x

   x^4

2) Take a screenshot and label the two curves.

3) Which function seems to grow faster, based on the picture?

4) Now change the maximum Y axis to be 1000, instead of 8. Take another screenshot.

5) Do you want to change your answer to Question 3 above? Why?

Exercise 6

Name _____ Date _____

Section _____

1) Start the "Plotter" applet. Select these functions from the "Choose Function" pull-down menu:

   1/x

2) What power of x is the same function as 1/x?

3) Suppose that somebody told you that Algorithm C's running time function for positive values of x was 1/x. What would this mean as x increased? That is, as your problem size increased, what happens to the running time of the algorithm?

4) Is this realistic? If not, why not?

Deliverables

Turn in your hand-written answers on a sheet of paper, along with the screenshots required for the two exercises.

Deeper Investigation

In this lab, we have investigated only one aspect of the limits of computing. The textbook describes this aspect and many others. The term given broadly to this part of computer science is *theoretical computer science*. Some of its findings are quite old, such as Turing's work, dating back to the 1930s, and some are at the cutting edge of the field. In the meantime, several questions remain unanswered.

Let's think about how we might measure software quality and complexity. Think back to some of the algorithms presented in Chapter 9 of the textbook. What kinds of measurements can you come up with that would begin to measure the quality or complexity of a chunk of code? Obviously, the number of lines in the code is a very crude measure. What more clever, insightful methods can you devise?

Turning our attention to the Traveling Salesperson Problem, we mentioned that finding the shortest path through a 40-city grid would probably take more time than the universe has to offer. Computer scientists have attempted to short-circuit this impossible situation by finding paths that may not be the best, but are pretty close to the best. Think of a way that we could change the Traveling Salesperson algorithm to do this. Currently, the algorithm looks at *all* paths and finds *all* circuits. Then it compares them and chooses the circuit with the smallest cost. (Hint: Think of an egg timer!)

Though the Traveling Salesperson Problem may strike you as silly and not terribly important, there certainly are real-world applications that are similar. Can you think of one or two? (Hint: Think about the Internet, think about security, and think about packet routing.)