by Jon L. Bentley

programming pearls

THE ENVELOPE IS BACK

The February 1984 column was about "back-of-the-envelope" calculations. When you're deciding whether to add a new command to a database system, for instance, you might want to estimate

How much programmer time is required to develop the code?

How many disks have to be added to store additional data?

Is the current CPU fast enough to provide reasonable response time?

Quick calculations are also useful in everyday life: will the reduced fuel bills of a 30-mile-per-gallon car balance a purchase price $1000 greater than a 20-mpg car?

Since that column appeared, many readers have contributed further ideas on the topic. This column presents several of those: rules of thumb useful for programmers, techniques for quick but accurate calculations, and Little's Law (a simple rule of amazing utility). But before we get to the technical details, the next section provides some mental stretching exercises.

A Warm-Up for Cool Brains

The student told me that the run time of his binary search program was 1.83 $\log_2 N$. When I asked, "1.83 what?" he thought for a minute and responded, "Either microseconds or milliseconds — I'm not really sure."

The student was blissfully ignorant of a factor of one thousand, three orders of magnitude. He couldn't even offer the excuse that performance wasn't a concern — he cared enough to calculate three significant digits. Like too many programmers, he suffered from what Douglas Hofstadter calls "number numbness": milliseconds and microseconds are both unimaginably small time units, so why bother distinguishing between the two? This section provides some resensitization exercises to increase your appreciation of orders of magnitude.

Is a factor of a thousand really such a big deal? A microyear is about 32 seconds while a milliyear is 8.8 hours — I wish I'd let the student choose between those two for how long he had to stay after school.

Electricity travels about a foot in a nanosecond — that's a bottleneck in supercomputer design. In a microsecond it can go across a large building, and in a millisecond from New York to Washington, D.C. And speaking of Washington, it seems that people there are always forgetting the difference between a million and a billion.

A fast sprinter can run a hundred meters in ten seconds, for an average velocity of 10 meters per second. One thousand times that speed is faster than the space shuttle, while one-thousandth the rate is slower than an ant. A factor of a thousand *is* a big deal, but there are bigger deals yet. Table I shows some additional order-of-magnitude checkpoints of velocity.[1]

If I describe a moving object, you can probably estimate its numerical velocity pretty accurately. Whether the object is a rocket flying through the air or a beaver gnawing through a log, you can most likely guess its speed to within a notch or two of its true position in Table I. In the next section we'll work on intuition about computational velocity.

Performance Rules of Thumb

I don't know how much salt costs, and I don't really care. It's so cheap that I use it without regard to cost, and when I run out I buy some more. Most programmers feel the same way about CPU cycles, with good reason — they cost next to nothing.

I expect that executives at salt companies have a different attitude toward the lowly substance. If each American consumes a dollar worth of salt each year, that creates a market worth a quarter of a billion dollars — a ten-percent decrease in production costs could be worth a fortune. And every now and then,

[1] Table I was inspired by *Powers of Ten*, by Morrison et al. (published in 1982 by Scientific American Books). Its subtitle is "A book about the relative size of things in the universe and the effect of adding another zero." It zooms in 42 factors of ten from a view 10^{25} meters across (ten thousand times the diameter of our galaxy), to a human being in a view one meter across, to a view 10^{-16} meters across deep within a carbon atom within the human's hand. The book's jacket accurately says, "An atlas of the universe and a guide to its exploration, *Powers of Ten* takes the reader on an extraordinary adventure in magnitudes. From the splendid photographs to the poetic pages of commentary, *Powers of Ten* is a book to savor, a visual odyssey in which we better comprehend not only the known world but our own place in it."

programmers must worry about the cost of CPU cycles for a similar reason: some programs spend them by the billions.

The price of salt is usually marked on the container, but how can you determine the cost of a line of code? Benchmarking the performance of a computer system is a difficult and important task; multimillion-dollar systems are purchased on the basis of ten and twenty-percent differences. Fortunately, rough estimates are easier to come by. They may be off from their "true" values by a factor of two, but they're still useful.

I'll now describe how I spent half an hour to get ballpark cost estimates on the system I usually use, a VAX-11/750® running the C language and the UNIX™ operating system. (Even if you don't care about CPU time, you may be interested in the design of these simple experiments.) I started with a five-line C program whose guts can be written in pseudocode as

```
N := 1000000
for I := 1 to N do ;
```

The UNIX system's `time` command reported that it took 6.1 seconds; each iteration of the (null) loop therefore cost 6.1 microseconds. My next experiment used the integer variables $I1$, $I2$ and $I3$:

```
N := 1000000
for I := 1 to N do I1 := I2 + I3
```

This code took 9.4 seconds, so an integer addition costs about 3.3 microseconds. To test the cost of procedure calls, I defined the integer function

```
function Sum2(A, B: integer)
    return A+B;
```

and assigned `I1 := Sum2(I2, I3)` in the loop. That took 39.4 seconds, so a function call with two integer parameters takes about 30 microseconds.

VAX and PDP are trademarks of Digital Equipment Corporation. UNIX is a trademark of AT&T Bell Laboratories.

TABLE I. Orders of Magnitude in Speed

Meters Per Sec	English Equivalent	Example
10^{-11}	1.2 in/century	Stalactites growing
10^{-10}	1.2 in/decade	Slow continent drifting
10^{-9}	1.2 in/year	Fingernails growing
10^{-8}	1 ft/year	Hair growing
10^{-7}	1 ft/month	Weeds growing
10^{-6}	3.4 in/day	Glacier
10^{-5}	1.4 in/hr	Minute hand of a watch
10^{-4}	1.2 ft/hour	Gastro-intestinal tract
10^{-3}	2 in/min	Snail
10^{-2}	2 ft/min	Ant
10^{-1}	20 ft/min	Giant tortoise
1	2.2 mi/hr	Human walk
10^{1}	22 mi/hour	Human sprint
10^{2}	220 mi/hour	Propeller airplane
10^{3}	37 mi/min	Fastest jet airplane
10^{4}	370 mi/min	Space shuttle
10^{5}	3700 mi/min	Meteor impacting earth
10^{6}	620 mi/sec	Earth in galactic orbit
10^{7}	6200 mi/sec	LA to satellite to NY
10^{8}	62,000 mi/sec	One-third speed of light

TABLE II. Cost of Mathematical Operations in C

Operation	μsecs
Integer operands	
Addition	3.3
Subtraction	3.7
Multiplication	10.6
Division	11.0
Floating operands	
Addition	10.6
Subtraction	10.2
Multiplication	15.7
Division	15.7
Conversions	
Integer to float	8.2
Float to integer	11.2
Functions	
Sine	790
Logarithm	860
Square root	940

Unfortunately, even experiments as simple as these are fraught with potential problems. Is C addition really that fast (3.3 microseconds), or did the compiler notice that the same addition was done repeatedly and therefore perform it just once, before the loop? But if that happened, what would account for the 3.3 microsecond delay? Maybe the new code was aligned differently in the instruction cache, or maybe the system was just busier. And so on and so on.

Testing those hypotheses expanded the performance experiments from a few minutes to half an hour, but I'm pretty sure that the resulting estimates are accurate to within, say, a factor of two. With that caveat, Table II presents ballpark estimates of the cost of several mathematical operations in this implementation of C.

It's easy to summarize Table II: most arithmetic operations in C cost about 10 microseconds. Integer addition/subtraction is faster (3.5 microseconds) and floating multiplication/division is slower (16 microseconds). But beware the evil functions! They're only a few characters to type, but they're two orders of magnitude more expensive than the other operators.

On those rare occasions when performance matters, I use Table II in two distinct ways. The general trend helps me to make accurate estimates. If a routine performs N^2 steps of a few arithmetic operations each, a C implementation will take roughly half a minute when N is 1000. If the routine is called once a day, I won't worry about its efficiency from now on. If I had planned to call the routine several times a minute in a real-time system, though, I won't even bother coding it and instead search for a better solution.

Table II also highlights the expensive operations. Budget-minded chefs can safely ignore the price of salt if caviar is on the menu; C programmers on this system can ignore the primitive arithmetic operations surrounding a square root. But beware that relative costs change from system to system. On a PDP-10™ Pascal compiler I once used, floating point operations cost 2 microseconds, while square roots and integer-to-float conversions both cost 40 microseconds (a conversion in C costs 1 floating point operation while a square root costs 60; in Pascal the costs are both 20). Problem 2 encourages you to estimate the costs on your system.

Confidence in Quick Calculations

Once you have the background data, these tricks can increase your trust in your back-of-the-envelope calculations.

Two Answers Are Better Than One. The February 1984 column presented a problem I first heard from Bob Martin: "How much water flows out of the Mississippi River in a day?" Near its mouth the river is, say, a mile wide, roughly twenty feet deep (1/250 mile), and flows at about five miles an hour (120 miles per day). Multiplying

$$1 \text{ mile} \times 1/250 \text{ mile} \times 120 \text{ miles/day}$$
$$\approx 1/2 \text{ mile}^3/\text{day}$$

shows that the river discharges about half a cubic mile of water per day. When I asked Peter Weinberger how much water flows out, he responded, "As much as flows in." He then estimated that the Mississippi basin is about 1000 by 1000 miles, and that the annual runoff from rainfall there is about one foot (or 1/5000 mile). That gives

$$1000 \text{ miles} \times 1000 \text{ miles} \times 1/5000 \text{ mile/year}$$
$$\approx 200 \text{ miles}^3/\text{year}$$

Since there are (roughly) 400 days per year, this is a little more than half a cubic mile per day.

Each derivation supports the other: I'd be real surprised if the true answer were a factor of ten away from these estimates. If the two had been off by a factor of a hundred, though, I'd check both derivations very carefully, think hard about the underlying assumptions, and look for a third derivation. When I looked up the answer in an almanac, I was only a little surprised that it reported the flow as 0.4 cubic miles per day — a twenty percent error was even less than I had hoped for.

Quick Checks. Polya devotes three pages of his *How To Solve It* to "Test by Dimension"; he describes it as a "well-known, quick and efficient means to check geometrical or physical formulas." The dimensions in a sum must be the same, which is in turn the dimension of the sum — you can add feet together to get feet, but you can't add seconds to pounds. The second rule is that the dimension of a product is the product of the dimensions. The examples above obey both rules; multiplying

$$(\text{miles}+\text{miles}) \times \text{miles} \times \text{miles/day} = \text{miles}^3/\text{day}$$

has the right form, apart from any constants.

Dimension tests check the form of equations. Check your multiplications and divisions with an old trick from slide rule days: independently compute the leading digit and the exponent. There are several quick checks for addition.

```
 3142      3142      3142
 2718      2718      2718
+1123     +1123     +1123
-----     -----     -----
  983      6982      6973
```

The first sum has too few digits and the second errs in the least significant digit. "Casting out nines" reveals the error in the third example: the digits in the summands sum to 8 modulo 9, while those in the answer sum to 7 modulo 9 (in a correct addition, the sums of the digits are equal after "casting out" groups of digits that sum to nine).

Above all, don't forget common sense: be suspicious of any calculations that show that your machine can sort an English dictionary in 6 nanoseconds.

Tabular Computations. A simple table can help keep track of messy computations. To perform Weinberger's calculation (how much water flows into the Mississippi), we first write down the three original factors.

$$\begin{array}{c|c|c} 1000\ \text{mi} & 1000\ \text{mi} & 1\ \ \text{mi} \\ \hline & & 5000\ \ \text{yr} \end{array}$$

Canceling terms shows the annual outflow is 200 miles^3/year.

$$\begin{array}{c|c|c|c} \cancel{1000}\ \cancel{\text{mi}} & \cancel{1000}\ \cancel{\text{mi}} & \cancel{1}\ \cancel{\text{mi}} & 200\ \text{mi}^3 \\ \hline & & \cancel{5000}\ \text{yr} & \end{array}$$

Now we multiply by the identity (well, close) that there are 400 days per year.

$$\begin{array}{c|c|c|c|c} \cancel{1000}\ \cancel{\text{mi}} & \cancel{1000}\ \cancel{\text{mi}} & \cancel{1}\ \cancel{\text{mi}} & 200\ \text{mi}^3 & \text{yr} \\ \hline & & \cancel{5000}\ \text{yr} & & 400\ \text{days} \end{array}$$

Cancellation yields the (by now familiar) answer of half a cubic mile per day.

$$\begin{array}{c|c|c|c|c|c} \cancel{1000}\ \cancel{\text{mi}} & \cancel{1000}\ \cancel{\text{mi}} & \cancel{1}\ \cancel{\text{mi}} & \cancel{200}\ \text{mi}^3 & \cancel{\text{yr}} & 1 \\ \hline & & \cancel{5000}\ \cancel{\text{yr}} & & \cancel{400}\ \text{days} & 2 \end{array}$$

Little's Law

Most back-of-the-envelope calculations use obvious rules: total cost is unit cost times number of units. Sometimes, though, one needs a more subtle insight. Bruce Weide of Ohio State University wrote the following note about a rule that is surprisingly versatile.

"The 'operational analysis' introduced by Denning and Buzen (see *Computing Surveys 10*, 3, November 1978, 225–261) is much more general than queueing network models of computer systems. Their exposition is excellent, but because of the article's limited focus, they didn't explore the generality of Little's Law. The proof methods have nothing to do with queues or with computer systems. Imagine *any* system in which things enter and leave. Little's Law states that 'The average number of things in the system is the product of the average rate at which things leave the system and the average time each one spends in the system.' (And if there is a gross 'flow balance' of things entering and leaving, the exit rate is also the entry rate.)

"I teach this technique of performance analysis in my computer architecture classes. But I try to emphasize that the result is a general law of systems theory, and can be applied to many other kinds of systems. For instance, if you're in line waiting to get into a popular nightspot, you might figure out how long you'll have to wait by standing there for a while and trying to estimate the rate at which people are entering. With Little's Law, though, you could reason, 'This place holds about 60 people, and the average Joe will be in there about 3 hours, so we're entering at the rate of about 20 people an hour. The line has 20 people in it, so that means we'll wait about an hour. Let's go home and read *Communications* instead.' You get the picture."

Peter Denning succinctly phrases this rule as "The average number of objects in a queue is the product of the entry rate and the average holding time." He applies it to his wine cellar: "I have 150 cases of wine in my basement and I consume (and purchase) 25 cases per year. How long do I hold each case? Little's Law tells me to divide 150 cases by 25 cases/year, which gives 6 years per case."

He then turns to more serious applications. "The response-time formula for a time-shared system can be proved using Little's Law and flow balance. Assume N terminals of average think time Z are connected to an arbitrary system with response time R. Each user cycles between thinking and waiting-for-response, so the total number of jobs in the meta-system (consisting of terminals and the computer system) is fixed at N. If you cut the path from the system's output to the terminals, you see a meta-system with average load N, average response time $Z+R$, and throughput X (measured in jobs per time unit). Little's Law says $N=X\times(Z+R)$, and solving for R gives $R = N/X - Z$."

Denning goes on to say that "Little's Law is more useful when augmented with the 'forced flow law' and the 'utilization law'. You can then calculate answers to questions like this: A humongous computer system contains a bazillion disks, a quadrillion CPUs, a classified operating system, and 20 terminals of average think time 20 seconds. Its disk unit is observed to serve 100 requests per job and runs at the rate of 25 requests per second. What is the system's throughput and response time? (I get 0.25 jobs/second and 60 seconds.) These answers are *exact* if the system is in flow balance, which is normally very close to true. *Any* system of arbitrary configuration containing a disk with those measured values and terminals of those measured values will have the same throughput and response time. Amazing? Only to the extent that one does not understand the power of the basic laws of system flow and congestion."

Principles

The four sections in this column highlight four assets that are often useful for programmers.

Familiarity with numbers.

Willingness to experiment.

Discipline in checking answers.

Mathematics, when you need it.

Problems

1. Make tables like Table I to illustrate factors of ten in measures such as time, weight, distance, area and volume.
2. Conduct experiments to measure the performance of your computer system. Table III is a starting point from which you may build your own list of

TABLE III. How Much Do They Cost?

- CPU Time
  - Control flow
    - Statement overhead: for, while, if
    - Subroutine call
  - Arithmetic operations
    - Integers
      - Add, subtract, multiply, divide
    - Floating point
      - Add, subtract, multiply, divide
      - Square root, logarithm, sine
    - Type conversions between integer and float
  - String operations
    - Comparison and copy
- I/O Time
  - Read/write one character/integer
  - Disk access
  - Disk accesses per database operation
- Utilities
  - Sort 10,000 integers
  - Sort 10,000 20-byte strings
  - Search a text file for a string

useful quantities. Other handy facts include the speed of your compiler (in lines of source code per second) and the disk space required to store a one-byte file.

3. Tables II and III assume a "performance model" in which variables are accessed in a constant amount of time and a given instruction always requires the same amount of time to execute. Give examples of systems on which these and other "reasonable" assumptions are violated.
4. Explain the mathematics underlying the rule of "casting out nines" (the sum of the digits in the summands equals the sum of the digits in the sum, modulo nine).
5. Two answers are better than one, and more are better yet. Describe several different ways to estimate each of the following quantities:
   a. The daily outflow of the Mississippi River.
   b. The death rate in your city (measured in percent of population per year).
   c. The average number of users on your system at various times of the day and the week.
6. An article on page 652 of the July 1984 *Communications* states that "the system handles an average of 7,328,764 transactions a day"; comments?
7. [P. J. Denning] Sketch a proof of Little's Law.
8. [P. J. Denning] Use Little's Law to characterize the flow of a job through a network of servers.
9. [B. W. Weide] Imagine a queue of customers waiting for service. In its usual interpretation, Little's Law relates the average total number of customers in the queue *and* in the server to the average time a customer spends waiting in the queue *and* in service. How are the average waiting time in the queue alone and the average number of customers in the queue alone related to these quantities?
10. [B. W. Weide] Many computer centers still have big mainframes that handle large numbers of batch jobs concurrently. Some even have a monitor showing the jobs awaiting execution, so you can see where your job stands. Jobs must await execution, of course, because there is always a backlog of work (by Murphy's Law, not Little's). Suppose the average job spends 20 seconds "in execution" on a machine that can execute 10 jobs concurrently, and that your job is the last of 100 "awaiting execution" (to be executed in first-in-first-out order). About how long can you expect to wait until your job is finished?
11. Determine various administrative costs in your organization: How much does it cost to buy a book (beyond the cover price)? To have a secretary type a letter? What is the cost of floor space (measured in dollars per square foot per year)? What is the cost of telephone and computing systems?

Quick Calculations in Everyday Life

Back-of-the-envelope calculations about everyday events are always good practice and good fun, and are sometimes even useful.[2] For instance, how much money have you spent in the past year eating in restaurants? I was once horrified to hear a New Yorker compute that he and his wife spend more money each month on taxicabs than they spend on rent. And for California readers (who may not know what a taxicab is), how long does it take to fill a swimming pool with a garden hose?

Here's one that stumped me for a while: what is the volume of a typical 6-foot-tall male? (Volume doesn't mean cubic feet in a crowded elevator, but rather cubic centimeters of meat.) A common response figures that the typical male is 6 feet high by 2 feet wide by half a foot thick, for 6 cubic feet. A more accurate estimate exploits the fact that humans are roughly the same density as water, approximately 60 pounds per cubic foot (most swimmers float when they inhale and sink when they exhale). A person who weighs 180 pounds is therefore about 3 cubic feet. If you know a person's weight, this relation can give you their volume to within a few percent, a feat impossible by multiplying length by width by height. (The February column described how Thomas Edison used a similar trick to compute the volume of a light bulb.)

Here are a few canned questions, but keep in mind that spontaneous questions are usually the most interesting.

1. If every person in your city threw a ping pong ball into your living room, how deep would the balls be?
2. What is the cost of a one-hour lecture at your organization? Include both preparation and audience time.
3. How much money will Americans spend this year on soft drinks? On cigarettes? On atomic bombs? On video games? On the space program?
4. How many words are in a typical book? How many words a minute do you read?
5. How many dollars per year is the difference between a 20-mile-per-gallon car and a 40 mpg car? Over the lifetime of a car? What if every driver in the United States chose one or the other?
6. How much does it cost to drive your car a mile? Don't forget insurance.

[2] A reader of a draft of this column described a trip to the supermarket he had taken three days earlier. He kept a running total as he walked through the aisles by rounding each item to $1.00, $2.00, or $3.00. His final tally was roughly $70.00, and he had enough confidence in that estimate to look at the register tape when the clerk announced the total price of $92.00. The clerk had mistakenly entered the product code of six oranges (number 429) as their price ($4.29); that raised a $2.00 purchase to $25.00.

7. How much would it cost to buy extension cords to reach from the earth to the moon?
8. An old rule of thumb says that a human sitting in a room radiates about 100 watts. How many calories per day must supply that radiator?

I'd like to end with a plea to teachers. In his paper referenced under Further Reading, Hofstadter tells how he asked students in a New York physics class the height of the Empire State Building, which they could see out the window. The true height is 1250 feet, but answers ranged from 50 feet to one mile. I recently had a similar experience *after* a lecture on "back-of-the-envelope" calculations. An examination question asked for the cost of a one-semester, fifteen-student class section at that college. Most students gave an answer within thirty percent of my estimate of $30,000, but the extremes ranged from a high of $100,000,000 to a low of $38.05.

If you're a teacher, spare ten minutes of lecture for this topic, then reinforce it with little examples throughout the class. Test your success by an examination question; I bet you'll find the answers interesting.

Solutions

3. On many microcomputer BASIC interpreters, the cost of accessing a variable is proportional to its position in the symbol table (variables used near the front of the program are cheaper to access than those first used late in execution). On machines with instruction caches, a minor change can slide an inner loop out of the cache and increase total time by twenty percent. Last week, a colleague squeezed a factor of ten from a program I had written in a pattern-scanning language by changing the quotation marks surrounding a pattern (I didn't appreciate a subtle semantic distinction).
4. Hint: use modular arithmetic. Observe that $(10\times x+y) \bmod 9 = (x+y) \bmod 9$.
5. One might estimate the local death rate by counting death notices in a newspaper and estimating the population of the area they represent. An easier approach uses Little's Law and an estimate of life expectancy.
6. Some people are afraid of supplying numbers ("I have no idea how deep the Mississippi River is"). At the other extreme are people who gladly supply accuracy that isn't there ("the river is 31.415926535 feet deep").
7. Peter Denning's argument has two parts: "First, define $\lambda = A/T$, the arrival rate, where A is the number of arrivals during an observation period of length T. Define $X = C/T$, the output rate, where C is the number of completions during T. Let $n(t)$ denote the number in the system at time t in $[0,T]$. Let W be the area under $n(t)$, in units of 'item-seconds', representing the total aggregated waiting time over all items in the system during the observation period. The mean response time per item completed is defined as $R = W/C$, in units of (item-seconds)/(item). The mean number in the system is the average height of $n(t)$ and is $L = W/T$, in units of (item-seconds)/(second). It is now obvious that $L = RX$. This formulation is in terms of the output rate only. There is no requirement for 'flow balance', i.e., that flow in equal flow out (in symbols, $\lambda = X$). If you add that assumption, the formula becomes $L = \lambda \times R$, which is the form encountered in queueing and system theory."
8. Peter Denning writes: "Suppose you have a network of servers. Let V_i denote the mean number of times each job uses (visits) server i. Then $V_1 + \cdots + V_N$ denotes the total number of job-steps in an average job. The overall system throughput, X_0, is related to the local throughput at server i by the 'forced flow' law: $X_i = V_i \times X_0$. Let R_0 denote the response time experienced by a job and L_0 denote the average number of jobs in the system. Little's formula says that the system's response time is $R_0 = L_0/X_0$. But $L_0 = L_1 + \cdots + L_N$, where L_i is the mean number of jobs at server i; $L_i = R_i \times X_i$, where R_i is the mean response time per visit to server i. Using $X_i/X_0 = V_i$ from the fixed flow law, you get $R_0 = R_1 \times V_1 + \cdots + R_N \times V_N$. This is intuitively true, but easily and rigorously proved using Little's law twice."
9. Bruce Weide writes: "In the original case, the 'system' is the queue plus the server. Using the notation of Solution 7, R is the average time a customer spends in the queue and in service, and L is the average number of customers in the queue and in service. So by Little's Law, we know $L = RX$, where X is the output rate of the server. But X is also the output rate of the queue, since a customer goes directly from the queue to the server whenever another leaves the server. Considering the queue by itself to be the 'system' and defining L_Q as the average number in the queue and R_Q as the average time spent in the queue alone, we see that $L_Q = R_Q X$. The desired relationship, then, is that the ratios L/R and L_Q/R_Q are equal."
10. Bruce Weide offers this solution. "One way to solve this problem considers two queueing systems. The first is the queue of jobs awaiting execution, and the second is the computer system itself. By Little's Law, the second system has the output rate of jobs, $X = L/R$. Here, $L = 10$ jobs (because there is always a backlog of work, the system will always have the maximum 10 jobs in it, so 10 is also the average number of jobs in the system). The average time $R = 20$ seconds, so X must be 1/2 job per second. This is also the arrival rate of jobs to the second system — flow

balance is satisfied because L is constant, which means every job completing execution is immediately replaced by the next job. Now the output rate of the first system must also be 1/2 job/second. We should therefore expect the 99 jobs ahead of ours to be out of the way after about 50 seconds (actually, 99/2). Then our job completes 20 seconds later, for a total wait of 70 seconds."

Further Reading

The February and July 1984 columns listed several excellent references for quick calculations. Edward Purcell edits the monthly "Back of the Envelope" column in the *American Journal of Physics*. Douglas Hofstadter's "Metamagical Themas" column in the May 1982 *Scientific American* is subtitled "Number numbness, or why innumeracy may be just as dangerous as illiteracy"; it is reprinted with a postscript in his book *Metamagical Themas*, published by Basic Books in 1985.

A First Course in Mathematical Modeling by Frank Giordano and Maurice Weir was published by Wadsworth in 1985. It describes many facets of mathematical "common sense" (which isn't common enough). Chapter 7 covers "Dimensional Analysis"; it is especially relevant to quick calculations.

For Correspondence: Jon Bentley, AT&T Bell Laboratories, Room 2C-317, 600 Mountain Ave., Murray Hill, NJ 07974.