programming pearls

by Jon Bentley

BUMPER-STICKER COMPUTER SCIENCE

Every now and then, programmers have to convert units of time. If a program processes 100 records per second, for instance, how long will it take to process one million records? Dividing shows that the task takes 10,000 seconds, and there are 3600 seconds per hour, so the answer is about three hours.

But how many seconds are there in a year? If I tell you there are 3.155×10^7, you won't even try to remember it. On the other hand, who could forget that, to within half a percent,

π seconds is a nanocentury.
Tom Duff
Bell Labs

So if your program takes 10^7 seconds, be prepared to wait four months.

February's column solicited bumper-sticker-sized advice on computing. Some of the contributions aren't debatable: Duff's rule is a memorable statement of a handy constant. This rule about a program testing method (regression tests save old inputs and outputs to make sure the new outputs are the same) contains a number that isn't as ironclad.

Regression testing cuts test intervals in half.
Larry Bernstein
Bell Communications Research

Bernstein's point remains whether the constant is 30 or 70 percent: these tests save development time.

There's a problem with advice that is even less quantitative. Everyone agrees that

Absence makes the heart grow fonder.
Anon.

and

Out of sight, out of mind.
Anon.

Everyone, that is, except the sayings themselves—they are contradictory. There are similar contradictions in the slogans in this column. Although there is some truth in each saying in this column, all should be taken with a grain of salt.

A word about credit. The name associated with a rule is usually the person who sent me the rule, even if they in fact attributed it to their Cousin Ralph (sorry, Ralph). In a few cases I have listed an earlier reference, together with the author's current affiliation (to the best of my knowledge). I'm sure that I have slighted many people by denying them proper attribution, and to them I offer the condolence that

Plagiarism is the sincerest form of flattery.
Anon.

Without further ado, here's the advice, grouped into a few major categories.

Coding

When in doubt, use brute force.
Ken Thompson
Bell Labs

Avoid arc-sine and arc-cosine functions—you can usually do better by applying a trig identity or computing a vector dot-product.
Jim Conyngham
Arvin/Calspan Advanced Technology Center

Allocate four digits for the year part of a date: a new millenium is coming.
David Martin
Norristown, Pennsylvania

Avoid asymmetry.
Andy Huber
Data General Corporation

The sooner you start to code, the longer the program will take.
Roy Carlson
University of Wisconsin

If you can't write it down in English, you can't code it
Peter Halpern
Brooklyn, New York

Details count.
Peter Weinberger
Bell Labs

If the code and the comments disagree, then both are probably wrong.
Norm Schryer
Bell Labs

A procedure should fit on a page.
David Tribble
Arlington, Texas

If you have too many special cases, you are doing it wrong.
Craig Zerouni
Computer FX Ltd.
London, England

Get your data structures correct first, and the rest of the program will write itself.
David Jones
Assen, The Netherlands

User Interfaces

[The Principle of Least Astonishment] Make a user interface as consistent and as predictable as possible.
Contributed by several readers

A program designed for inputs from people is usually stressed beyond the breaking point by computer-generated inputs.
Dennis Ritchie
Bell Labs

Twenty percent of all input forms filled out by people contain bad data.
Vic Vyssotsky
Bell Labs

Eighty percent of all input forms ask questions they have no business asking.
Mike Garey
Bell Labs

Don't make the user provide information that the system already knows.
Rick Lemons
Cardinal Data Systems

For 80 percent of all data sets, 95 percent of the information can be seen in a good graph.
William S. Cleveland
Bell Labs

Debugging

Of all my programming bugs, 80 percent are syntax errors. Of the remaining 20 percent, 80 percent are trivial logical errors. Of the remaining 4 percent, 80 percent are pointer errors. And the remaining 0.8 percent are hard.
Marc Donner
IBM T. J. Watson Research Center

It takes three times the effort to find and fix bugs in system test than when done by the developer. It takes ten times the effort to find and fix bugs in the field than when done in system test. Therefore, insist on unit tests by the developer.
Larry Bernstein
Bell Communications Research

Don't debug standing up. It cuts your patience in half, and you need all you can muster.
Dave Storer
Cedar Rapids, Iowa

Don't get suckered in by the comments—they can be terribly misleading. Debug only the code.
Dave Storer
Cedar Rapids, Iowa

Testing can show the presence of bugs, but not their absence.
Edsger W. Dijkstra
University of Texas

Each new user of a new system uncovers a new class of bugs.
Brian Kernighan
Bell Labs

If it ain't broke, don't fix it.
Ronald Reagan
Santa Barbara, California

[The Maintainer's Motto] If we can't fix it, it ain't broke.
Lieutenant Colonel Walt Weir
United States Army

The first step in fixing a broken program is getting it to fail repeatably.
Tom Duff
Bell Labs

Performance

[The First Rule of Program Optimization] Don't do it.

[The Second Rule of Program Optimization—For experts only] Don't do it yet.
Michael Jackson
Michael Jackson Systems Ltd.

The fastest algorithm can frequently be replaced by one that is almost as fast and much easier to understand.
Douglas W. Jones
University of Iowa

On some machines indirection is slower with displacement, so the most-used member of a structure or a record should be first.
Mike Morton
Boston, Massachusetts

In non–I/O-bound programs, a few percent of the source code typically accounts for over half the run time.

Don Knuth
Stanford University

Before optimizing, use a profiler to locate the "hot spots" of the program.

Mike Morton
Boston, Massachusetts

[Conservation of Code Size] When you turn an ordinary page of code into just a handful of instructions for speed, expand the comments to keep the number of source lines constant.

Mike Morton
Boston, Massachusetts

If the programmer can simulate a construct faster than the compiler can implement the construct itself, then the compiler writer has blown it badly.

Guy L. Steele, Jr.
Tartan Laboratories

To speed up an I/O-bound program, begin by accounting for all I/O. Eliminate that which is unnecessary or redundant, and make the remaining as fast as possible.

David Martin
Norristown, Pennsylvania

The fastest I/O is no I/O.

Nils-Peter Nelson
Bell Labs

The cheapest, fastest, and most reliable components of a computer system are those that aren't there.

Gordon Bell
Encore Computer Corporation

[Compiler Writer's Motto—Optimization Pass] Making a wrong program worse is no sin.

Bill McKeeman
Wang Institute

Electricity travels a foot in a nanosecond.

Commodore Grace Murray Hopper
United States Navy

LISP programmers know the value of everything but the cost of nothing.

Alan Perlis
Yale University

[Little's Formula] The average number of objects in a queue is the product of the entry rate and the average holding time.

Peter Denning
RIACS

Documentation

[The Test of Negation] Don't include a sentence in documentation if its negation is obviously false.

Bob Martin
AT&T Technologies

When explaining a command, or language feature, or hardware widget, first describe the problem it is designed to solve.

David Martin
Norristown, Pennsylvania

[One Page Principle] A (specification, design, procedure, test plan) that will not fit on one page of 8.5-by-11 inch paper cannot be understood.

Mark Ardis
Wang Institute

The job's not over until the paperwork's done.

Anon.

Managing Software

The structure of a system reflects the structure of the organization that built it.

Richard E. Fairley
Wang Institute

Don't keep doing what doesn't work.

Anon.

[Rule of Credibility] The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.

Tom Cargill
Bell Labs

Less than 10 percent of the code has to do with the ostensible purpose of the system; the rest deals with input-output, data validation, data structure maintenance, and other housekeeping.

Mary Shaw
Carnegie-Mellon University

Good judgment comes from experience, and experience comes from bad judgment.

Fred Brooks
University of North Carolina

Don't write a new program if one already does more or less what you want. And if you must write a program, use existing code to do as much of the work as possible.

Richard Hill
Hewlett-Packard S.A.
Geneva, Switzerland

Whenever possible, steal code.

Tom Duff
Bell Labs

Good customer relations double productivity.
Larry Bernstein
Bell Communications Research

Translating a working program to a new language or system takes 10 percent of the original development time or manpower or cost.
Douglas W. Jones
University of Iowa

Don't use the computer to do things that can be done efficiently by hand.
Richard Hill
Hewlett-Packard S.A.
Geneva, Switzerland

Don't use hands to do things that can be done efficiently by the computer.
Tom Duff
Bell Labs

I'd rather write programs to write programs than write programs.
Dick Sites
Digital Equipment Corporation

[Brooks's Law of Prototypes] Plan to throw one away, you will anyhow.
Fred Brooks
University of North Carolina

If you plan to throw one away, you will throw away two.
Craig Zerouni
Computer FX Ltd.
London, England

Prototyping cuts the work to produce a system by 40 percent.
Larry Bernstein
Bell Communications Research

[Thompson's rule for first-time telescope makers] It is faster to make a four-inch mirror then a six-inch mirror than to make a six-inch mirror.
Bill McKeeman
Wang Institute

Furious activity is no substitute for understanding.
H. H. Williams
Oakland, California

Always do the hard part first. If the hard part is impossible, why waste time on the easy part? Once the hard part is done, you're home free.

Always do the easy part first. What you think at first is the easy part often turns out to be the hard part. Once the easy part is done, you can concentrate all your efforts on the hard part.
Al Schapira
Bell Labs

If you lie to the computer, it will get you.
Perry Farrar
Germantown, Maryland

If a system doesn't have to be reliable, it can do anything else.
H. H. Williams
Oakland, California

One person's constant is another person's variable.
Susan Gerhart
Microelectronics and Computer Technology Corp.

One person's data is another person's program.
Guy L. Steele, Jr.
Tartan Laboratories

If you've made it this far, you'll certainly appreciate this excellent advice.

Eschew clever rules.
Joe Condon
Bell Labs

Although this column has allocated just a few words to each rule, most could be greatly expanded (say, into an undergraduate paper or into a bull session over a few beers). These problems show how one might expand the following rule.

Make it work first before you make it work fast.
Bruce Whiteside
Woodridge, Illinois

Your "assignment" is to expand other rules in a similar fashion.

1. Restate the rule to be more precise. The example rule might actually be intended as

   Ignore efficiency concerns until a program is known to be correct.

   or as

   If a program doesn't work, it doesn't matter how fast it runs; after all, the null program gives a wrong answer in no time at all.

2. Present small, concrete examples to support your rule. In Chapter 7 of their *Elements of Programming Style*, Kernighan and Plauger present 10 tangled lines of code from a programming text; the convoluted code saves a single comparison (and incidentally introduced a minor bug). By "wasting" an extra comparison, they replace the code with two crystal-clear lines. With that object lesson fresh on the page, they present the rule

   Make it right before you make it faster.

3. Find "war stories" of how the rule has been used in larger programs.
   a. It is pleasant to see the rule save the day. Carnegie-Mellon Computer Science Report CMU-CS-83-108 describes how I made a system right before I made it faster: I built several programs in 2,000 lines of clean code, most of which worked like a charm. Unfortunately, one 600-line program to be run several times a day required 14.5 hours. Profiling showed that 66 lines of code accounted for 13.6 hours of the run time, and just 3 lines accounted for 11 hours (10 percent of the code took 94 percent of the time, and 0.5 percent of the code took 75 percent of the time). I therefore concentrated my efficiency efforts on those "hot spots," and properly ignored efficiency elsewhere.
   b. It can be even more impressive to hear how ignoring the rule leads to a disaster. When Vic Vyssotsky modified a Fortran compiler in the early 1960s he spent a week making a correct routine very fast, and thereby introduced a bug. The bug surfaced two years later, because the routine *had never once been called* in over 100,000 compilations. Vyssotsky's week of premature optimization was worse than wasted: it made a good program bad. (This story, though, served as fine training for Vyssotsky and generations of Bell Labs programmers.)
4. Critique the rules: which are always "capital-T Truth" and which are sometimes misleading? Bill Wulf of Tartan Laboratories took only a brief conversation to convince me that "if a program doesn't work, it doesn't matter how fast it runs" isn't quite as true as I once thought. He used the case of a document production system that we both used. Although it was faster than its predecessor, at times it seemed excruciatingly slow: it took several hours to compile a book. Wulf's clincher went like this: "That program, like any other large system, today has 10 known, but minor, bugs. Next month, it will have 10 different known bugs. If you could choose between removing the 10 current bugs or making the program run 10 times faster, which would you pick?"

```
function maxheap(l,u,  i) { # 1 if a heap
    for (i = 2*l; i <= u; i++)
        if (x[int(i/2)] < x[i]) return 0
    return 1
}

function assert(cond, errmsg) {
    if (!cond) {
        print ">>> Assertion failed <<<"
        print "    Error message: ", errmsg
    }
}

function siftdown(l,u,  i,c,t) {
        # pre  maxheap(l+1,u)
        # post maxheap(l,u)
assert(maxheap(l+1,u), "siftdown pre")
    i = l
    while (1) {
        # maxheap(l,u) except between
        #  i and its children
        c = 2*i
        if (c > u) break
        if (c+1 <= u && x[c+1] > x[c]) c++
        if (x[i] >= x[c]) break
        t=x[i]; x[i]=x[c]; x[c]=t  # swap i, c
        i = c
    }
assert(maxheap(l,u), "siftdown post")
}

function draw(i,s) {
    if (i <= n) {
        print i ":", s, x[i]
        draw(2*i,   s "    ")
        draw(2*i+1, s "    ")
    }
}

$1=="draw"   { draw(1, "") }
$1=="down"   { siftdown($2, $3) }
$1=="assert" { assert(maxheap($2, $3), "cmd") }
$1=="x"      { x[$2]=$3 }
$1=="n"      { n=$2 }
```

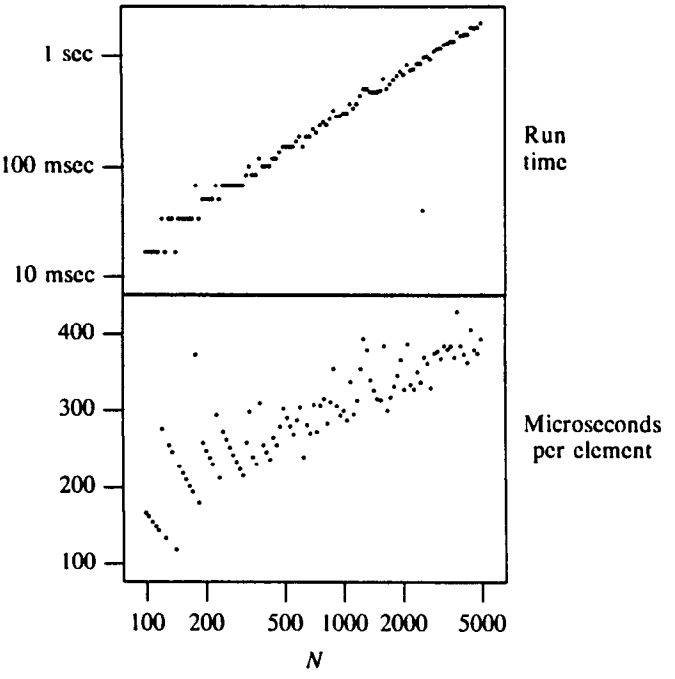
PROGRAM 1. An AWK Testbed for Heaps

Solutions for July's Problems

Several readers reported a horrible typographical error on page 672 of the July column. In the fourth line of the second paragraph in the right-hand column, the assignment `l=m+7` should have been `l=m+1`; the number "one" was mistakenly typeset as a "seven." Solutions 1, 2, and 3 refer to Program 1.

1. Program 1 is a testbed for the `siftdown` routine. The recursive `draw` routine uses indentation to print the implicit tree structure of the heap (the second parameter in the recursive call appends four spaces to the indentation string `s`).
2. The modified `assert` routine in Program 1 includes a string variable that provides information about the assertion that failed. Many systems provide an assertion facility that automatically gives the source file and the line number of the invalid assertion.
3. The `siftdown` routine in Program 3 uses the `assert` and `maxheap` routines to test the pre- and post-conditions on entry and exit. The `maxheap` routine requires $O(U\text{-}L)$ time, so the `assert` calls should be removed from the production version of the code.
4. The tests in the last column missed a bug in my first `siftup` procedure. I mistakenly initialized `i` with the incorrect assignment `i=n` rather than with the correct assignment `i=u`. In all my tests, though, `u` and `n` were equal, so they did not identify the bug. (The published `siftup` was correct, however, as far as I know.)
5. I commonly use scaffolding to time algorithms.
6. For this solution, I gathered data on the run of

the "Quicksort 2" program described in the April 1984 column (with the *CutOff* parameter set to 15). My C program was 92 lines long: 41 lines of scaffolding supported 51 lines of "real" code. (The "real" code took just 26 lines of AWK, and similar scaffolding took 10 lines.) The top graph plots the run time of Quicksort on a VAX-11/750® as a function of the size of the input array, N. The one hundred N values are uniformly spaced along the logarithmic scale. The small times are discrete because the system measures time in sixtieths of a second. That graph shows that run time is strongly correlated to input size, but provides little insight beyond that.



Further Reading

If you like heavy doses of unadorned rules, try Tom Parker's *Rules of Thumb* (Houghton Mifflin, 1983). The following rules appear on its cover

798. One ostrich egg will serve 24 people for brunch.

886. A submarine will move through the water most efficiently if it is 10 to 13 times as long as it is wide.

The book contains 896 similar rules.

Strunk and White's classic *Elements of Style* (Macmillan, third edition 1979) is built around 43 rules such as

Omit needless words.

That rule is elaborated in one and a half pages of vigorous writing, much of which is before-and-after examples. The book is just 85 pages long. On a per page basis, it is arguably the best style book ever written for English.

Kernighan and Plauger's *Elements of Programming Style* (McGraw-Hill, second edition 1978) is similar to Strunk and White both in title and in execution. They illustrate the rule

Keep it simple to make it faster.

on a 29-line sort from a programming text: a straightforward 9-line program is 25 percent faster. John Shore compares Kernighan and Plauger to Strunk and White in his *Sachertorte Algorithm, And Other Antidotes To Computer Anxiety* (Viking, 1985). His side-by-side presentation of 10 rules from each shows how good programming is similar to good writing. He ends with the rule

Do not take shortcuts at the cost of clarity.

and the riddle of whether it is from Strunk and White or from Kernighan and Plauger.

Fred Brooks's *Mythical Man Month* (Addison-Wesley, 1975) contains dozens of rules about software, including his classic

[Brooks's Law] Adding manpower to a late software project makes it later.

Butler Lampson's "Hints for Computer System Design" (*IEEE Software*, January 1984) summarize his experience in building dozens of state-of-the-art systems.

The y-scale in the bottom graph is the run time per array element in microseconds (that is, the total time divided by N). This graph displays the wide variation in run time due to the randomizing *Swap* operation. The straightness of the data indicates that the run time per element grows logarithmically, which implies that the overall run time of Quicksort is $O(N \log N)$. Most algorithms texts give a mathematical proof of this fact.

8. To test that a sort routine permutes its input, we could copy the input into a separate array, sort that by a trusted method, and compare the two arrays after the new routine has finished. An alternative method uses only a few bytes of storage, but sometimes makes a mistake: it uses the sum of the elements in the array as a *signature* of those elements. Changing a subset of the elements will change the sum with high probability. (Summing involves problems related to word size and nonassociativity of floating-point addition; other signatures, such as exclusive or, avoid these problems.)

VAX is a trademark of Digital Equipment Corporation.

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