Donald E. Knuth received a Ph.D. in mathematics from the California Institute of Technology in 1963, writing a thesis in algebra under the supervision of Marshall Hall. After serving on the faculty at Cal Tech, he joined the Department of Computer Science at Stanford University in 1968. He has worked in the analysis of algorithms, combinatorics, programming languages, and the history of computer science. He also designed the $\TeX$ typesetting system. His monumental work The Art of Computer Programming *demonstrates the significant interaction between mathematics and computer science.*

Algorithmic Themes

DONALD E. KNUTH

I like to think of mathematics as a vast musical instrument on which one can play a great variety of beautiful melodies. Many generations of mathematicians have provided us with rich tonal resources that offer limitless possibilities for harmonious combination.

A great performance of mathematics can be as exciting to the audience as it is to the person controlling the instrument. Whether we are replaying a classic theme, or improvising a new one, or just fooling around, we experience deep pleasure when we encounter patterns that fit together just right, or when we can pull out all the stops in order to unify independent voices and timbres.

This analogy isn't perfect, because mathematics is the music as well as the organ for its creation. But a view of mathematics as a multivoiced mechanism helps me understand the relationship between mathematics and its infant step-child called computer science. I believe computer science has made and will continue to make important contributions to mathematics primarily because it provides an inspiration for new themes and rhythms by which the delicious modulations of mathematics can be enjoyed and enriched.

Computer science is not the same as mathematics, nor is either field a subset of the other. I believe that there is roughly as much difference between a computer scientist and a mathematician as there is between a mathematician and a physicist (although the distance from computer science to physics is greater than the other two distances). People like myself look at mathematics as a device for articulating computer science, but there is of course a converse relation: Many mathematicians see computer science as an instrument for developing mathematics. Both viewpoints are valid, yet I wish to stress the former, which I believe is more significant for mathematicians. Computer science is now enriching mathematics — as physics did in previous generations — by asking new sorts of questions, whose answers shed new light on mathematical structures. In this way computer science makes fundamental improvements to the mathematical ensemble. When good music is played, it influences the builders of musical instruments; my claim is that the cadences of computer science are having a profound and beneficial influence on the inner structure of mathematics. (In a similar way, applications of computers to physics, medicine, psychology, mathematics, art — and, yes, music — are improving the core of computer science. But that's another story.)

I must admit that my intuitive impressions about the distinction between mathematics and computer science are not universally shared. Such opinions cannot be demonstrated like theorems. But I know that I experience a conscious "culture shock" when I switch from a mathematician's way of thinking to that of a computer scientist and back again.

For example, I recall that when I was studying the properties of Dedekind sums [**10**], I began that work with the mentality I had when I was a graduate student of mathematics, but then I got stuck. The next day I looked at the remaining problems with computer science eyes, and I saw how to write an algorithm and to ask new questions; this led me to another plateau. Once again I was stuck, since my computer science ideas had now been exhausted. So I put a mathematical cap on again and was able to move further. Such alternation continued over a period of weeks, and I could really feel the transitions.

Another example, perhaps more convincing to someone besides myself, is based on my experiences with a mathematical novelette called *Surreal Numbers* [**8**]. When I wrote that little book I was definitely relishing the perspective of a pure mathematician, with no illusions that the book would be of the slightest interest to a computer scientist. Subsequent book reviews bore out this hypothesis: The work was praised in the *Bulletin* as "an exciting and stimulating book which 'turns on' the reader" [**2**], and Gian-Carlo Rota recently wrote (while reviewing another book) that "Surreal numbers are an invention of the great J. Conway. They may well go down in history as one of the great inventions of the century" [**14**]. But the consensus in computer-science circles is that "The book is a failed experiment" [**16**].

I would like to think that those book reviews prove my point about the difference between computer scientists and mathematicians. But the argument is not conclusive, because there are different kinds of mathematicians too. For example, I showed the manuscript of *Surreal Numbers* to George Pólya before it was published; he replied as follows [**12**]:

> I must confess, I am prejudiced against the case you have chosen for a case study. I simply cannot imagine that mathematically unsophisticated young people can be interested in this kind of "abstract" topic and even develop creativity on it. I cannot get rid of my prejudice — to be honest, I cannot even really wish to get rid of it, it is in my constitution: I can develop interest only in starting from concrete, or "relatively concrete" situations (difficulties, questions, observations, ...).

Perhaps Pólya was constitutionally a computer scientist?

If I had to put my finger on the greatest difference between mathematicians and computer scientists, I would say that mathematicians have a strong preference for uniform rules, coupled with a strong dislike for case-by-case analysis; computer scientists, by contrast, are comfortable and fluent with highly non-uniform structures (like the different operations performed by real computers, or like the various steps in long and complex algorithms). This tolerance of nonuniformity is the computer scientists' strength as well as their weakness; it's a strength because they can bring order into situations where no clean mathematical models exist, but it's a weakness because they don't look hard enough for uniformity when a uniform law is actually present. The distinction between uniform laws — which are a mathematician's staple food — and non-uniform algorithms and data structures — which are bread and butter to a computer scientist — has been described beautifully by G. S. Tseytin [**15**], who tells about an evolution in his own thinking.

There are other differences between our fields and our mentalities; for example, a computer scientist is less concerned with infinite and continuous objects, and more concerned with finite (indeed small) and discrete ones. A computer scientist is concerned about efficient constructions, etc. But such things are more or less corollaries of the main uniform/nonuniform dichotomy.

My purpose in this essay is, however, not to dwell on perceived differences between mathematics and computer science, but rather to say *vive la différence*, and to emphasize mathematics. Indeed, much of my own work tries to have a foot planted firmly in each camp.

What is it that I do? I like to call it "analysis of algorithms" [**5**, **6**]. The general idea is very simple: Given an algorithm, I try to understand its quantitative behavior. I ask how much time the algorithm will take to perform its task, given a probability distribution of its inputs.

I remember vividly how I first became interested in this topic. The year was 1962, and I was a graduate student in mathematics; however, I was spending the summer earning some money by writing a computer program (a FORTRAN compiler). As I worked on that program I came to the part where an interesting algorithm called "hashing" was appropriate, and I had recently heard a rumor that two of Feller's students at Princeton had tried unsuccessfully to analyze the speed of hashing. Programming was hard work, so I took break one weekend and tried to solve this reportedly unsolvable problem. With a stroke of luck, I found the answer (see [7, pp. 529–530] and [9]); somehow my experience in programming the method had helped in the analysis. The nice thing was that the answer involved an interesting type of mathematical function I hadn't seen before:

$$1 + \frac{n}{m} + \frac{n(n-1)}{m^2} + \frac{n(n-1)(n-2)}{m^3} + \cdots .$$

(Later I would find this and similar functions arising in connection with many other algorithms.)

Well, it was fun to analyze the performance of hashing, and I soon realized that a lot more algorithms were out there waiting to be studied. I had heard about a comparatively new subject called "queuing theory"; gosh, I thought, if an entire subdiscipline can be devoted to the study of one small class of algorithms, surely there is much interesting work to be done in the study of *all* classes of algorithms. There was clearly more than a lifetime's worth of things to be done, and I decided that I wanted to spend a major part of my own life doing them. Not only was the mathematics good, the results were appreciated by programmers, so there was a double payoff.

Analysis of algorithms has been the central focus of my work ever since. After more than 25 years, I still find no shortage of interesting problems to work on. And the main point is that these problems almost invariably have a clean mathematical structure, appealing in its own right. Some applications of mathematics are no doubt boring, but the problems suggested by important algorithms have consistently turned out to be exciting. Indeed, overstimulation has been the real drawback; I need to find ways to *stop* thinking about analysis of algorithms, in order to do various other things that human beings ought to do.

Time and again I experience "the incredible effectiveness of mathematics": Looking at a new computer method (such as an algorithm for information retrieval called Patricia), I'll find that its running time depends on quantities that mathematicians have been studying for hundreds of years (such as the gamma function, hyperbolic cosine, and zeta function in the case of Patricia [7, exercise 6.3–34]).

One of the most venerable algorithms of all is Euclid's procedure for calculating greatest common divisors. I tried unsuccessfully to analyze it in 1963, so I asked several of my teachers for help. The problem is this: Let τ_n be the number of steps taken by Euclid's algorithm to determine that m and n are relatively prime, averaged over the $\varphi(n)$ nonnegative integers m that are less than n and prime to n. If we assume that the fraction m/n behaves like a random real number, Lévy's theory of continued fractions suggests that

τ_n will be asymptotically $12 \ln 2/\pi^2$ times $\ln n$. My empirical calculations in 1963 confirmed this and showed, in fact, that

$$\tau_n \approx \frac{12 \ln 2}{\pi^2} \ln n + 1.47.$$

In the first (1969) edition of [4] I discussed this conjecture and wrote:

> We have only given plausible grounds for believing that the related quantity τ_n is asymptotically $(12 \ln 2/\pi^2) \ln n$, and the theory does not suggest any formula for the empirically determined constant 1.47. The heuristic reasoning, and the overwhelming empirical evidence..., mean that for all practical purposes the analysis of Euclid's algorithm is complete. From an aesthetic standpoint, however, there is still a gaping hole left.

Research by Heilbronn [3] and Dixon [1] soon established the constant $12 \ln 2/\pi^2$, and Porter [13] proved that

$$\tau_n = \frac{12 \ln 2}{\pi^2} \ln n + C + O(n^{-1/6+\varepsilon}).$$

John Wrench and I subsequently determined that Porter's constant $C = 1.4670780794\ldots$ has the closed form

$$C = \frac{6 \ln 2}{\pi^2}(3 \ln 2 + 4\gamma - 24\pi^2 \zeta'(2) - 2) - \frac{1}{2}.$$

Therefore I could happily say in the second edition of [4] that "conjecture (48) is fully proved."

A more surprising development occurred when A. C. Yao and I decided to analyze the primitive version of Euclid's algorithm that is based on subtraction instead of division. Consider the average sum σ_n of all partial quotients of the regular continued fractions for m/n, where $1 \leq m \leq n$; this is the average running time of the subtractive algorithm for gcd. If we assume that rational fractions behave like almost all real numbers, a theorem of Khintchine states that the sum of the first k partial quotients will be approximately $k \log_2 k$. And since $k = O(\log n)$, we expect $\sigma_n = O(\log n \log \log n)$. However, Yao and I proved that

$$\sigma_n = \frac{6}{\pi^2}(\ln n)^2 + O(\log n (\log \log n)^2).$$

Therefore rational numbers tend to have larger partial quotients than their real counterparts—even though Heilbronn showed that the kth quotient of a rational number m/n approaches the corresponding distribution of a real number, for all fixed k as $n \to \infty$. This is the most striking case I know where the analogy between discrete and continuous values leads to an incorrect estimate.

Different kinds of algorithms lead to different corners of mathematics. In fact, I think that by now my colleagues and I have used results from every branch of mathematics (judging by the *MR* categories), except one. The lone exception is the topic on which I wrote my Ph.D. dissertation: finite projective planes. But I still have hopes of applying even that to computer science some day.

Here's a curious identity that illustrates some of the diversity that can arise when algorithms are analyzed: Let $\|x\|$ denote the distance from x to the nearest integer. Then

$$\cdots + \tfrac{1}{8}\|8x\|^2 + \tfrac{1}{4}\|4x\|^2 + \tfrac{1}{2}\|2x\|^2 + \|x\|^2 + 2\|\tfrac{x}{2}\|^2 + 4\|\tfrac{x}{4}\|^2 + 8\|\tfrac{x}{8}\|^2 + \cdots = |x|.$$

The sum is doubly infinite, converging at the left because $\|x\| \le \frac{1}{2}$, and converging at the right because $\|x/2^k\|$ is ultimately equal to $|x/2^k|$. The identity holds for all real x; I stumbled across it when working on an algorithm based on Brownian motion **[11]**.

Analysis of algorithms is only one small aspect of the interaction between mathematics and computer science. I have chosen to mention a few autobiographical examples only because I understand them better than I can understand some of the deeper things. I could have touched instead on some of the recent advances in algebra and number theory that have occurred as new algorithms for algebraic operations and factorization have been found. Or I could have highlighted the exciting field of discrete and computational geometry that is now opening up. And so on.

My point is rather that a great deal of interesting work remains to be done, even after a person has invented an algorithm to solve some mathematical problem. We can ask, "How good is the algorithm?" and this question will often lead to a host of relevant issues. Indeed, there will be enough good stuff to keep subsequent generations of mathematicians happy for another century at least.

References

1. John D. Dixon, "The number of steps in the Euclidean algorithm," *J. Number Theory* **2** (1970), 414–422.

2. Aviezri S. Fraenkel, Review of *On Numbers and Games* by J. H. Conway and *Surreal Numbers* by D. E. Knuth, *Bull. Amer. Math. Soc.* **84** (1978), 1328–1336.

3. Hans A. Heilbronn, "On the average length of a class of finite continued fractions," *Abhandlungen aus Zahlentheorie und Analysis = Number Theory and Analysis*, ed. by Paul Turán, Plenum, 1968/1969, 87–96.

4. Donald E. Knuth, *The Art of Computer Programming*, Vol. 2: *Seminumerical Algorithms* (Addison-Wesley, Reading, Mass., 1969).

5. Donald E. Knuth, "The analysis of algorithms," *Actes du Congrès International des Mathématiciens* 1970, **3** (Gauthier-Villars, Paris, 1971), 269–274.

6. Donald E. Knuth, "Mathematical analysis of algorithms," *Proceedings of IFIP Congress 1971*, **1** (Amsterdam: North-Holland, 1972), 19–27.

7. Donald E. Knuth, *The Art of Computer Programming*, Vol. 3: *Sorting and Searching* (Addison-Wesley, Reading, Mass., 1973).

8. Donald E. Knuth, *Surreal Numbers*, Addison-Wesley, Reading, Mass., 1974.

9. Donald E. Knuth, "Computer Science and its Relation to Mathematics," *American Mathematical Monthly* **81** (April 1974), 323–343.

10. Donald E. Knuth, "Notes on generalized Dedekind sums," *Acta Arithmetica* **33** (1977), 297–325.

11. Donald E. Knuth, "An algorithm for Brownian zeroes," *Computing* **33** (1984), 89–94.

12. George Pólya, letter to the author dated July 8, 1973.

13. J. W. Porter, "On a theorem of Heilbronn," *Mathematika* **22** (1975), 20–28.

14. Gian-Carlo Rota, review of *An Introduction to the Theory of Surreal Numbers* by H. Gonshor, *Advances in Math.* **66** (1987), 318.

15. G. S. Tseytin, "From logicism to proceduralism (an autobiographical account)," in *Algorithms in Modern Mathematics and Computer Science*, A. P. Ershov and D. E. Knuth, eds., *Lecture Notes in Computer Science* **122** (1981), 390–396.

16. Eric Weiss, "Mathematics on the beach," *Abacus* **1**, 3 (Spring 1984), 44.