STANFORD
UNIVERSITY

DONALD E. KNUTH
Professor Emeritus of
The Art of Computer Programming

Computer Science
353 Serra Mall
Stanford CA 94305-9045
PHONE (650) 723-4367

24 April 2009

Alison J Brimelow
President, European Patent Office
80298 Munich, Germany

Dear Ms Brimelow,

A friend in Europe just told me that you are interested in "amicus curiæ" letters to explain why so many computer scientists around the world have long been alarmed about patent trends, and that you hope to receive them by 30 April. I hope this letter reaches you in time; I could not send it by FedEx, having no complete address.

Enclosed is a copy of a letter that I wrote to the US Patent Commissioner in 1994; I believe it is self explanatory. Also enclosed is the transcript of a talk I gave at the Technical University of Munich in 2001, where I gave a somewhat more nuanced view of extremely unusual cases in which algorithms or even mathematical constants might conceivably be patentable in my view. [The latter remarks occur near the end of a rather long lecture; I have highlighted the relevant information, on page 324, for your convenience.]

Basically I remain convinced that the patent policy most fair and most suitable for the world will regard mathematical ideas (such as algorithms) to be not subject to proprietary patent rights. For example, it would be terrible if somebody were to have a patent on an integer, like say 1009, so that nobody would be able to use that number "with further technical effect" without paying for a license. Although many software patents have unfortunately already been granted in the past, I hope that this practice will not continue in future. If Europe leads the way in this, I expect many Americans would want to emigrate so that they could continue to innovate in peace!

Sincerely,

Donald E Knuth
Professor of The Art of Computer Programming

STANFORD UNIVERSITY STANFORD, CALIFORNIA 94305-2140

DONALD E. KNUTH
Professor Emeritus of The Art of
Computer Programming
Department of Computer Science
Telephone [415] 723-4367

February 23, 1994

Commissioner of Patents and Trademarks
Box 4
Patent and Trademark Office
Washington, DC 20231

Dear Commissioner:

Along with many other computer scientists, I would like to ask you to reconsider the current policy of giving patents for computational processes. I find a considerable anxiety throughout the community of practicing computer scientists that decisions by the patent courts and the Patent and Trademark Office are making life much more difficult for programmers.

In the period 1945–1980, it was generally believed that patent law did not pertain to software. However, it now appears that some people have received patents for algorithms of practical importance—e.g., Lempel-Ziv compression and RSA public key encryption—and are now legally preventing other programmers from using those algorithms.

This is a serious change from the previous policy under which the computer revolution became possible, and I fear this change will be harmful for society. It certainly would have had profoundly negative effect on my own work: For example, I developed software called TEX that is now used to produce more than 90% of all books and journals in mathematics and physics and to produce hundreds of thousands of technical reports in all scientific disciplines. If software patents had been commonplace in 1980, I would not have been able to create such a system, nor would I probably have ever thought of doing it, nor can I imagine anyone else doing so.

I am told that the courts are trying to make a distinction between mathematical algorithms and nonmathematical algorithms. To a computer scientist, this makes no sense, because every algorithm is as mathematical as anything could be. An algorithm is an abstract concept unrelated to physical laws of the universe.

Nor is it possible to distinguish between "numerical" and "nonnumerical" algorithms, as if numbers were somehow different from other kinds of precise information. All data are numbers, and all numbers are data. Mathematicians work much more with symbolic entities than with numbers.

Therefore the idea of passing laws that say some kinds of algorithms belong to mathematics and some do not strikes me as absurd as the 19th century attempts of the Indiana legislature to pass a law that the ratio of a circle's circumference to its diameter is exactly 3, not approximately 3.1416. It's like the medieval church ruling that the sun revolves about the earth. Man-made laws can be significantly helpful but not when they contradict fundamental truths.

Congress wisely decided long ago that mathematical things cannot be patented. Surely nobody could apply mathematics if it were necessary to pay a license fee whenever the theorem of Pythagoras is employed. The basic algorithmic ideas that people are now rushing to patent are so fundamental, the result threatens to be like what would happen if we allowed authors to have patents on individual words and concepts. Novelists or journalists would be unable to write stories unless their publishers had permission from the owners of the words. Algorithms are exactly as basic to software as words are to writers, because they are the fundamental building blocks needed to make interesting products. What would happen if individual lawyers could patent their methods of defense, or if Supreme Court justices could patent their precedents?

I realize that the patent courts try their best to serve society when they formulate patent law. The Patent Office has fulfilled this mission admirably with respect to aspects of technology that involve concrete laws of physics rather than abstract laws of thought. I myself have a few patents on hardware devices. But I strongly believe that the recent trend to patenting algorithms is of benefit only to a very small number of attorneys and inventors, while it is seriously harmful to the vast majority of people who want to do useful things with computers.

When I think of the computer programs I require daily to get my own work done, I cannot help but realize that none of them would exist today if software patents had been prevalent in the 1960s and 1970s. Changing the rules now will have the effect of freezing progress at essentially its current level. If present trends continue, the only recourse available to the majority of America's brilliant software developers will be to give up software or to emigrate. The U.S.A. will soon lose its dominant position.

Please do what you can to reverse this alarming trend. There are far better ways to protect the intellectual property rights of software developers than to take away their right to use fundamental building blocks.

Sincerely,

Donald E. Knuth
Professor

All Questions Answered

Donald Knuth

On October 5, 2001, at the Technische Universität München, Donald Knuth presented a lecture entitled "All Questions Answered". The lecture drew an audience of around 350 people. This article contains the text of the lecture, edited by Notices *senior writer and deputy editor Allyn Jackson.*

Originally trained as a mathematician, Donald Knuth is renowned for his research in computer science, especially the analysis of algorithms. He is a prolific author, with 160 entries in MathSciNet. Among his many books is the three-volume series The Art of Computer Programming [TAOCP], *for which he received the AMS Steele Prize for Exposition in 1986. The citation for the prize stated that* TAOCP *"has made as great a contribution to the teaching of mathematics for the present generation of students as any book in mathematics proper in recent decades." The long awaited fourth volume is in preparation and some parts are available through Knuth's website,* `http://www-cs-faculty.stanford.edu/~knuth/`.

Knuth is the creator of the TEX and METAFONT languages for computer typesetting, which have revolutionized the preparation and distribution of technical documents in many fields, including mathematics. In 1978 he presented the AMS Gibbs Lecture entitled "Mathematical Typography". The lecture was subsequently published in the Bulletin of the AMS *[MT].*

Knuth earned his Ph.D. in mathematics in 1963 from the California Institute of Technology under the direction of Marshall Hall. He has received the Turing Award from the Association for Computing Machinery (1974), the National Medal of Science (1979), the Adelsköld Medal from the Royal Swedish Academy of Sciences (1994), the Harvey Prize from the Technion of Israel (1995), the John von Neumann Medal from the Institute of Electrical and Electronics Engineers (1995), and the Kyoto Prize from the Inamori Foundation (1996). Since 1968 Knuth has been on the faculty of Stanford University, where he currently holds the title of Professor Emeritus of The Art of Computer Programming.

—Allyn Jackson

Knuth: In every class that I taught at Stanford, the last day was devoted to "all questions answered". The students didn't have to come to class if they didn't want to, but if they did, they could ask any question on any subject except religion or politics or the final exam. I got the idea from Richard Feynman, who did the same thing in his classes at Caltech, and it was always interesting to see what the students really wanted to know. Today I'll answer any question on any subject. Do we allow religion or politics? I don't know. But there is no final exam to worry about. I'll try to answer without taking too much time so that we can get a lot of questions in.

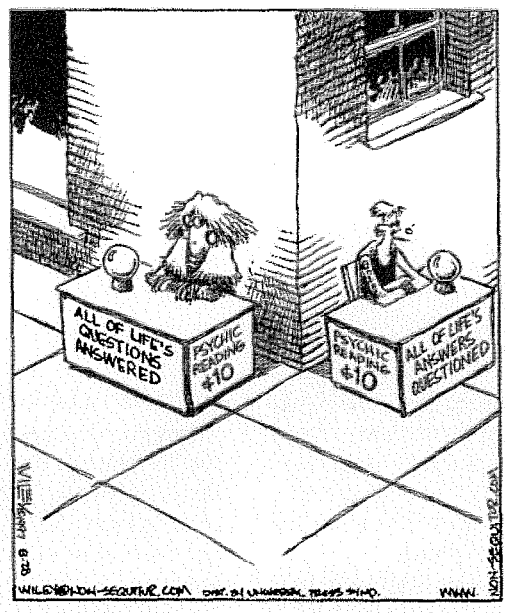
So, who wants to ask the first question?... Well, if there are no questions...[Knuth makes as if to leave.]

Question: *There was a special report to the American president, the PITAC report [PITAC], containing some recommendations. I am wondering whether you would be willing to comment on the priorities outlined in these recommendations: better software engineering, building a teraflop*

computer, improvements in the Internet including higher security and higher bandwidth, and the socio-economic impacts of managing information available via computer networks.

Knuth: I think that's a brilliant solution of the problem of what to present to a president. But in fact what I would like to see is thousands of computer scientists let loose to do whatever they want. That's what really advances the field. From my experience writing *The Art of Computer Programming*, if you asked me any year what was the most important thing that happened in computer science that year, I probably would have no answer for the question, but over five years' time the whole field changes. Computer science is a tremendous collaboration of people from all over the world adding little bricks to a massive wall. The individual bricks are what make it work, and not the milestones.

Next question?



Question: *Mathematicians say that God has the "Book of Proofs", where all the most aesthetic proofs are written. Can you recommend some algorithms for the "Book of Algorithms"?*

Knuth: That's a nice question. It was Paul Erdős who promulgated the idea that God has a book containing the best mathematical proofs, and I guess my friend Günter Ziegler in Berlin has recently written about it [PFB].

I remember that mathematicians were telling me in the 1960s that they would recognize computer science as a mature discipline when it had 1,000 deep algorithms. I think we've probably reached 500. There are certainly lots of algorithms that I think have to be considered absolutely beautiful and immortal, in some sense. Two examples are the Euclid algorithm and a corresponding one that works in binary notation and that may have been developed independently in China, almost as early as Euclid's algorithm was invented in Greece. In my books I am mostly concerned with the algorithms that are classical and that have been around for a long time. But still, every year we find brand new ideas that I think are going to remain forever.

Question: *Do you have thoughts on quantum computing?*

Knuth: Yes, but I don't know a great deal about it. It's quite a different paradigm from what I'm used to. It has lots of things in common with the kind of computing I know, but it's also quite mysterious in that you have to get all the answers at the end; you don't make a test and then have that determine what you do next. A lot of you have seen the movie *Lola rennt* (called *Run Lola Run* in the U.S.), in which the plot is played out three different times, with the outcome taking three different branches. Quantum computing is something like that: The world goes into many different branches, and we're interested in the one where the outcome is the nicest.

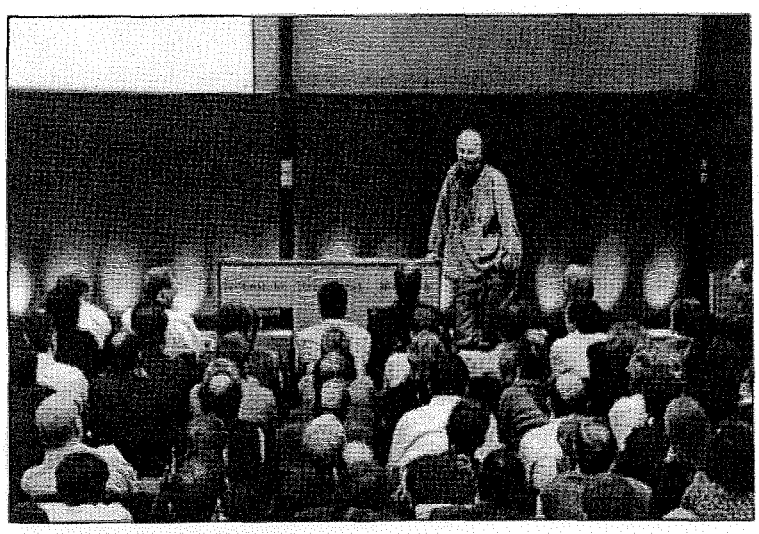
I'm good at nonquantum computing myself, so it's quite possible that if quantum computing takes over, I won't be able to do the new stuff. My life's work is with computers not because I'm interested so much in computation, but because I happen to be good at this kind of computing. Fortunately for me, I found that the thing I could do well was interesting to other people. I didn't develop an ability to think about algorithms because I wanted to help people solve problems. Somehow, by the time I was a teenager, I had a peculiar way of thinking that made me good at programming. But I might not be good at quantum programming. It seems to be a different world from my own.

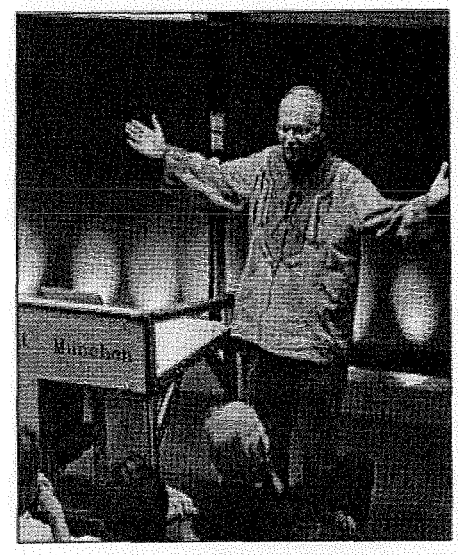
I'll take a question from the back.

Question: *I am working in theorem proving, and one of the most important papers is your paper "Simple word problems in universal algebra" [KB] from 1970, written with P. B. Bendix. I have two questions. The first is, do you still follow this area and what do you think of it? And the second is, who is and what became of P. B. Bendix?*

Knuth: This work was published in 1970, but I actually did it in 1967 while I was at Caltech. It was a simple idea, but fortunately it's turned out to be very widely applicable. The idea is to take a set of mathematical axioms and find all the implications of those axioms. If I have a certain set of axioms and you have a possible theorem, you ask, does this theorem follow from those axioms or not? I called my paper "Simple word problems in universal algebra", and I said a problem was "simple" if my method could handle it. Now people have extended the method quite a lot, so that a lot more problems are "simple". I think their work is beautiful.

The year 1967 was the most dramatic year of my life by far. I had no time for research. I had two children less than two years old; I had been scheduled to be a lecturer for ACM (Association for Computing Machinery) for three weeks; I had





to give lectures in a NATO summer school in Copenhagen; I had to speak in a conference at Oxford; and so on. And I was getting the page proofs for *The Art of Computer Programming*, of which the first volume was being published in 1968. All of this was in addition to the classes I was teaching, and an attack of ulcers that put me in the hospital, and being an editor for twelve journals. That year I thought of two little ideas. One has become known as the Knuth-Bendix algorithm; the other one is known as attribute grammars [AG]. That was the most creative year of my life, and it was also the most hectic.

You asked about Peter Bendix. He was a sophomore in a class I taught at Caltech, "Introduction to Algebra". Every student was supposed to do a class project, and Peter did his term paper on the implementation of the algorithm. He was a physics major. This was the time of the Vietnam War, and he became an objector. He went to Canada and worked as a high school teacher for about five years and later got a degree in physics. I found he was living near Stanford a couple of years ago, so I called him up and found out that he has had a fairly happy life in recent years.

In the 1960s, if I wrote a joint paper with my advisor Marshall Hall, it meant that he did the theory and I did the programming. But if I wrote a paper with anybody else, it meant that I did the theory and the other person did the programming. So Pete Bendix was a good programmer who implemented the method.

Question: *It seems to me it's easier to revise a book than the huge software programs we see day to day. How can we apply theory to improve software?*

Knuth: Certainly errors in software are more difficult to fix than errors in books. In fact, my main conclusion after spending ten years of my life working on the TEX project is that software is hard. It's harder than anything else I've ever had to do. While I was working on the TEX program, I was unable to do full-time teaching. Although I love teaching, I had to take a year off from it because there was just too much to keep in my head at one time. Writing a book is a little more difficult than writing a technical paper, but writing software is a lot more difficult than writing a book.

In my books I offer rewards for the first person who finds any particular error, and I must say that I've written more checks to people in Germany than in any other country in the world. I get letters from all over, but my German readers are the best nitpickers that I've ever had! In software I similarly pay for errors in the TEX and METAFONT programs. The reward was doubling every year: It started out at $2.56, then it went to $5.12, and so on, until it reached $327.68, at which time I stopped doubling. There has been no error reported in TEX since 1994 or 1995, although there is a rumor that somebody has recently found one. I'm going to have to look at it again in a year or two. I do everything in batch mode, by the way. I am going to look again at possible errors in TEX in, say, the year 2003.

I think letting users know that you welcome reports of errors is one important technique that could be used in the software industry. I think Microsoft should say, "You'll get a check from Bill Gates every time you find an error."

Question: *What importance do you give to the design of efficient algorithms, and what emphasis do you suggest giving this area in the future?*

Knuth: I think the design of efficient algorithms is somehow the core of computer science. It's at the center of our field. Computers are incredibly fast now compared to what they were before, so for many problems there is no need to have an efficient algorithm. I can write programs that are in some sense extremely inefficient, but if it's only going to take one second to get the answer, who cares? Still, some things we have to do millions or billions of times, and just knowing that the number of times is finite doesn't tell us that we can handle it. So the number of problems that are in need of efficient algorithms is huge. For example, many problems are NP complete, and NP complete is just a small level of complexity. Therefore I see an almost infinite horizon for the need for efficient algorithms. And that makes me happy because those are the kinds of problems I like the best.

Question: You have a big interest in puzzles, including the "Tower of Hanoi" puzzle on more than 3 pegs. I won't ask a harder question—what is the shortest solution?—because I am not sure everyone knows this puzzle. But I will ask a more philosophical question: Is it possible to show this can never be solved?

Knuth: Do people know the "Tower of Hanoi" problem? You have 3 pegs, and you have disks of different sizes. You're supposed to transfer the disks from one peg to another, and the disks have to be sorted on each peg so that the biggest is always on the bottom. You can move only one disk at a time. Henry Dudeney invented the idea of generalizing this puzzle to more than 3 pegs, and the task of finding the shortest solution to the 4-peg problem has been an open question for more than a hundred years. The 3-peg problem is very simple; we teach it to freshmen.

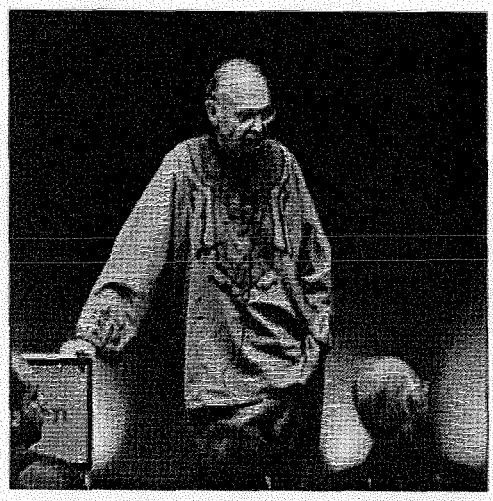
But take another, more famous problem, the Goldbach conjecture in mathematics: Every even integer is the sum of two odd primes. Now, I think that's a problem that's never going to be solved. I think it might not even have a proof. It might be one of the unprovable theorems that Gödel showed exist. In fact, we now know that in some sense almost all correct statements about mathematics are unprovable. Goldbach's conjecture is just, sort of, true because it can't be false. There are so many ways to represent an even number as the sum of two odd numbers, that as the numbers grow the number of representations grows bigger and bigger. Take a $10^{10^{10}}$-digit even number, and imagine how many ways there are to write that as the sum of two odd integers. For an n-bit odd number, the chances are proportional to $1/n$ that it's prime. How are all of those pairs of odd numbers going to be nonprime? It just can't happen. But it doesn't follow that you'll find a proof, because the definition of primality is multiplicative, while Goldbach's conjecture pertains to an additive property. So it might very well be that the conjecture happens to be true, but there is no rigorous way to prove it.

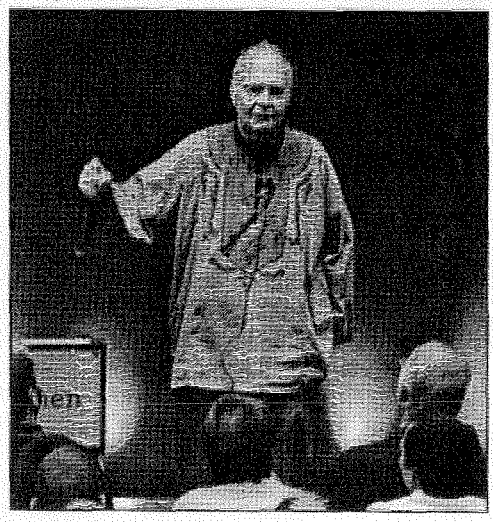
In the case of the 4-peg "Tower of Hanoi", there are many, many ways to achieve what we think is the minimum number of moves, but we have no good way to characterize all those solutions. So that's why I personally came to the conclusion that I was never going to be able to solve it, and I stopped working on it in 1972. But I spent a solid week working on it pretty hard.

Question: What are the five most important problems in computer science?

Knuth: I don't like this "top ten" business. It's the bottom ten that I like. I think you've got to go for the little things, the stones that make up the wall.

Question: You spent a lot of time on computer typesetting. What are your reflections on the impact of this work?

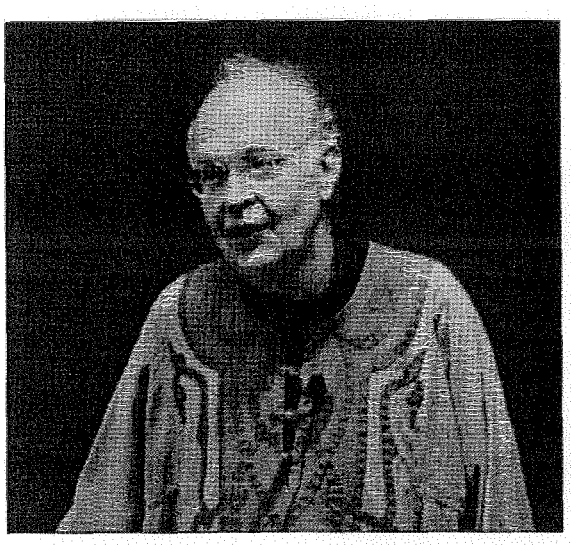


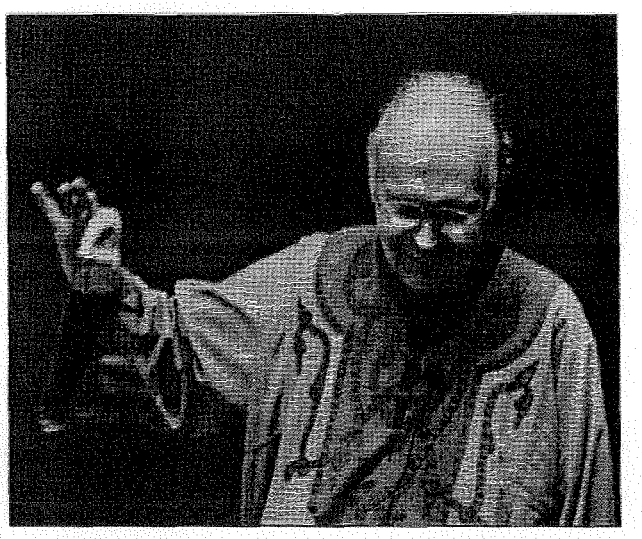


Knuth: I am extremely happy that my work was in the public domain and made it possible for people on all platforms to communicate with each other via the Internet. Especially now I'm thrilled by some recent projects. Two weeks ago I heard a great lecture by Bernd Wegner from the Technical University of Berlin about the plans for online journals by the European Mathematical Society. Such things would simply have been impossible without the open source software that came out of my work. So I'm extremely delighted this is helping to advance science.

I'm happy to see many books that look pretty good. Before I started my work, books on mathematics were looking worse and worse from year to year. It took a lot of skilled handwork to do it in the old system. The people who could do that were dying out, and high priority did not go to mathematical books. I never expected that TEX would take over the entire world of publishing. I'm not a very competitive person, and I did not want to take jobs away from anybody who was doing another way of printing. But I found that nobody wanted to do mathematical publishing well, so math was something I could improve without getting anybody upset about me being an upstart.

The downside is that I'm too sensitive to things now. I can't go to a restaurant and order food





because I keep looking at the fonts on the menu. Five minutes later I realize that it's also talking about food. If I had never thought about computer typesetting, I might have had a happier life in some ways.

Question: *Can you give us an outline for computer science, some milestones for the next ten or twenty years?*

Knuth: You're asking for milestones again. There is a lot of interest in applications to biology because so many things have opened up in that domain, with chances to cure diseases. The fact that human beings are based on a discrete code means that people like you and me, who are good at discrete problems, are able to do relevant work for this area. The problems are very difficult and challenging, and that's why I foresee an important future there.

But in all aspects of our field, I really don't see any slowing down. Every time I think I've discovered something interesting, I look on the Internet and find that somebody else has done it too. So we have a field that at the moment still seems to be like a boiling kettle, where you can't keep the lid on.

In the field of biology, I think we can confidently predict that it's going to have rich problems to solve for at least 500 years. I can't make that claim for computer science.

Question: *What is the connection between mathematics and computer programming viewed as an art?*

Knuth: Art is *Kunst*. The American movie *Artificial Intelligence* is called *Kunstlicher Intelligenz* in Germany—that is, artificial as well as artistic. I think of programming with beauty in mind, as being something elegant, something that you can be proud of for the way it fits together. Mathematics in the same way has elegance. Both fields, computing and mathematics, are different from other sciences because they are artificial; they are not in nature. They're totally under our own control. We make up the axioms, and when we solve a problem, we can prove that we've solved it. No astronomer will ever know whether his theories of astronomy are correct. You can't go up to the sun and measure it.

So these are my first thoughts on that connection. But there is a difference between mathematics and computer programming, and sometimes I can feel when I'm putting on one hat or the other. Some parts of me like mathematics, and some parts of me like `emacs` hacking. I think they go together okay, but I don't see that they're the same paradigm.

Question: *What is the relationship between God and computers?*

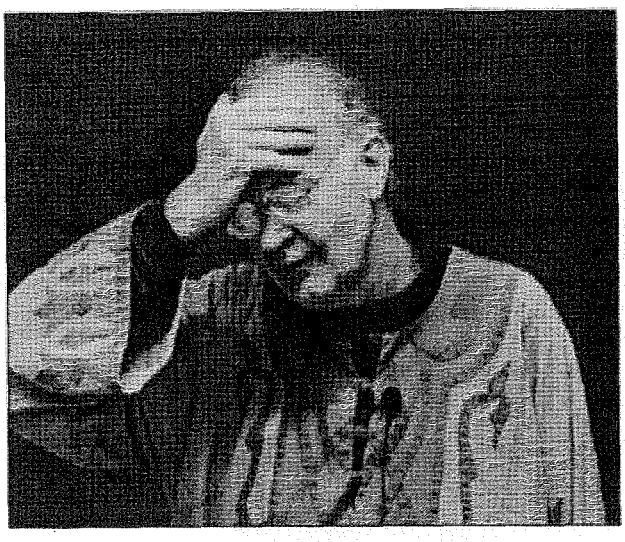
Knuth: In one of my books, *3:16 Bible Texts Illuminated* [BTI], I used random sampling to study sixty different verses of the Bible and what people from all different religious persuasions and different centuries have said about those verses. I did the study at first on my own, and then I found it was interesting enough that I ought to make a book about it. I got sixty of the best artists in the world to illustrate the book, many of them in Germany. The artwork was exhibited twice in Germany, and in other countries around the world. It was also shown in the National Cathedral in Washington, DC. In that book I used methodology that computer scientists often use for understanding a complicated subject, to see if that method would give some insight into the Bible, which is a complicated subject. In the book, I don't give answers. I just say I think it's good that life should be an ongoing search. The journey is more important than the destination.

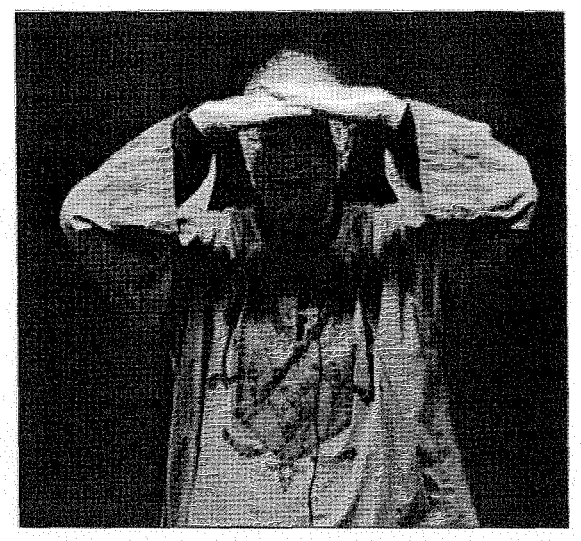
Question: *Do you know whether "P equals NP" has been proved? I heard a rumor that it has.*

Knuth: Which rumor did you hear?

Question: *One from Russia.*

Knuth: From Russia? That's new to me. Well, I don't think anybody has proved that P equals NP yet. But I know that Andy Yao has retired and hopes to solve the problem in the next five to ten years. He is inspired by Andrew Wiles, who





devoted several years to proving Fermat's Last Theorem. They're both Princeton people. Andy can do it if anybody can.

Three or four years ago, there was a paper in a Chinese journal of computer science and technology by a professor who claimed he could solve an NP-hard problem in polynomial time. The problem was about cliques, and he had a very clever way to represent cliques. The method was supposedly polynomial time, but it actually took something like n^{12} steps, so you couldn't even check it when n equals 5. So it was very hard to see the bug in his proof. I went to Stanford and sat down with our graduate students, and we needed a couple of hours before we found the flaw. I wrote the author a letter pointing out the error, and he wrote back a couple of months later, saying, "No, no, there is no error." I decided not to pursue it any further. I had done my part. But I don't believe it's been solved. That's the most mind-boggling problem facing theoretical computer science, and maybe all of science at the moment.

Question: *What do you think of research in cryptographic algorithms? And what do you think of efforts by politicians today to put limits on cryptography research?*

Knuth: Certainly the whole area of cryptographic algorithms has been one of the most active and exciting areas in computer science for the past ten years, and many of the results are spectacular and beautiful. I can't claim that I'm good at that particular subject, though, because I can't think of sneaky attacks myself. But the key problem is, what about the abuse of secure methods of communication? I don't want criminals to use these methods to become better criminals.

I'm a religious person, and I think that God knows all my secrets, so I always feel that whatever I'm thinking is public knowledge in some way. I come from this kind of background. I don't feel I have to encrypt everything I do. On the other hand, I would certainly feel quite differently if somebody started to use such openness against me, by stealing my bank accounts or whatever. So I am supportive of a high level of secrecy. But whether it should be impossible for the authorities to decode things even in criminal investigations, in extreme cases—there I tend to come down on the side of wanting to have some way to break some keys sometimes.

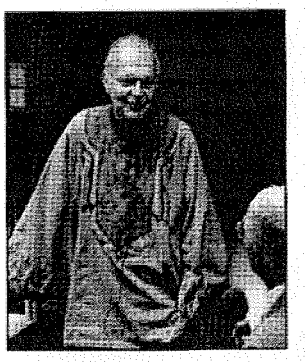
Question: *Will we have intelligent machines sometime in the future? Should we have them?*

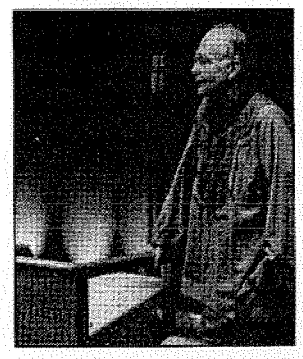
Knuth: There have always been inflated estimates as to how soon we are going to have a machine that's intelligent. I still see no signs of getting around the central problem of understanding what is cognition, what it means to think. Neurologists are making better measurements than they ever have before, but we are still so far from finding an answer that I can't yet rank neuroscience as one of the most active fields of current work. Biology has been getting answers, with DNA and stem cells and so on. But with cognition we are still looking for the secret.

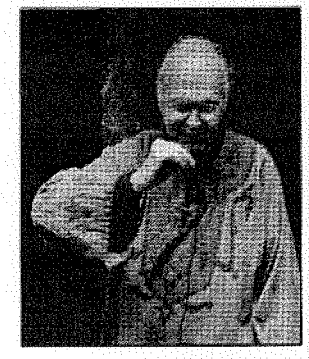
Some very thought-provoking books came out a year or two ago, one by Hans Moravec [Mo], and one by Ray Kurzweil [Ku]. Both of them are saying that in twenty or thirty years we are going to have machines smarter than humans. Some people were worried about that. My attitude is, if that's true, more power to them. If they are smarter than us, so what? Then we can learn from them. But I see no signs that there are any breakthroughs around the corner.

Two weeks ago in Greece I was at the inauguration of a new book by Christos Papadimitriou, who is chairman of computer science at Berkeley. He published a novel in the Greek language called *The Smile of Turing* [Pa]. I don't want to give away the story, but when it gets published in German or English, you'll find that it has a very nice discussion of artificial intelligence and the Turing test for intelligence.

The most promising model of how the brain works that I've seen says that the brain is a dynamic genetic algorithm that operates all the time. As I







am talking to you now, your brains have a lot of competing theories about what I'm going to say. It's the survival of the fittest, a continual battle among the competing theories. Some come to the surface and actually enter your consciousness, but the others are all there. Some kind of mating of concepts might be going on in our heads all the time. This model seems to have the right properties to account for how we can do what we do with the relatively slow response time that our neurons have. But I am by no means an expert on this.

Question: What is your thinking about software patents? There is a big discussion going on in Europe right now about whether software should be patentable.

Knuth: I'm against patents on things that any student should be expected to discover. There have been an awful lot of software patents in the U.S. for ideas that are completely trivial, and that bothers me a lot. There is an organization that has worked for many years to make patents on all the remaining trivial ideas and then make these available to everyone. The way patenting had been going was threatening to make the software industry stand still.

Algorithms are inherently mathematical things that should be as unpatentable as the value of π. But for something nontrivial, something like the interior point method for linear programming, there's more justification for somebody getting a right to license the method for a short time, instead of keeping it a trade secret. That's the whole idea of patents; the word patent means "to make public".

I was trained in the culture of mathematics, so I'm not used to charging people a penny every time they use a theorem I proved. But I charge somebody for the time I spend telling them which theorem to apply. It's okay to charge for services and customization and improvement, but don't make the algorithms themselves proprietary.

There's an interesting issue, though. Could you possibly have a patent on a positive integer? It is not inconceivable that if we took a million of the greatest supercomputers today and set them going, they could compute a certain 300-digit constant that would solve any NP-hard problem by taking the GCD of this constant with an input number, or by some other funny combination. This integer would require massive amounts of computation time to find, and if you knew that integer, then you could do all kinds of useful things. Now, is that integer really discovered by man? Or is it something that is God given? When we start thinking of complexity issues, we have to change our viewpoint as to what is in nature and what is invented.

***Question:** You have been writing checks to people who point out errors in your books. I have never heard of anyone cashing these checks. Do you know how much money you would be out of, if everyone suddenly cashed the checks?*

Knuth: There's one man who lives near Frankfurt who would probably have more than $1,000 if he cashed all the checks I've sent him. There's a man in Los Gatos, California, whom I've never met, who cashes a check for $2.56 about once a month, and that's been going on for some years now. Altogether I've written more than 2,000 checks over the years, and the average amount exceeds $8.00 per check. Even if everybody cashed their checks, it would still be more than worth it to me to know that my books are getting better.

References

[TAOCP] *The Art of Computer Programming*, by Donald E. Knuth. *Volume 1: Fundamental Algorithms* (third edition, Addison-Wesley, 1997). *Volume 2: Seminumerical Algorithms* (third edition, Addison-Wesley, 1997). *Volume 3: Sorting and Searching* (second edition, Addison-Wesley, 1998). *Volume 4: Combinatorial Algorithms* (in preparation).

[MT] Mathematical typography, by Donald E. Knuth. *Bull. Amer. Math. Soc. (N.S.)* 1 (1979), no. 2, 337–372. Reprinted in *Digital Typography* (Stanford, California: CSLI Publications, 1998), pp. 19–65.

[PITAC] President's information technology advisory committee. See `http://www.itrd.gov/ac/`.

[PFB] *Proofs from The Book*, by Martin Aigner and Günter Ziegler. Second edition, Springer Verlag, 2001.

[KB] Simple word problems in universal algebras, by Peter B. Bendix and Donald Knuth. *Computational Problems in Abstract Algebra*, J. Leech, ed. (Oxford: Pergamon, 1970), pp. 263–297. Reprinted in *Automation of Reasoning*, Jörg H. Siekmann and Graham Wrightson, eds. (Springer, 1983), pp. 342–376.

[BTI] *3:16 Bible Texts Illuminated*, by Donald E. Knuth. A-R Editions, Madison, Wisconsin, 1990.

[AG] Semantics of context-free languages, by Donald E. Knuth. *Mathematical Systems Theory* 2 (1968), 127–145. See also The genesis of attribute grammars, in *Lecture Notes in Computer Science* 461 (1990), 1–12.

[Pa] *TO XAMOGELO TOY TOYRINGK* (*The Smile of Turing*), by Christos Papadimitriou. Livani Publishers, Athens, Greece, 2001.

[Ku] *The Age of Spiritual Machines: When Computers Exceed Human Intelligence*, by Ray Kurzweil. Penguin USA, 2000.

[Mo] *Robot: Mere Machine to Transcendent Mind*, by Hans P. Moravec. Oxford University Press, 2000.

Photographs used in this article are courtesy of Andreas Jung, Technische Universität München.