Teaching Statement

Scott Aaronson

January 6, 2007

My basic proposal is to *sing the ideas of theoretical computer science from the rooftops*—by creating new undergraduate courses, training graduate students, helping talented undergraduates reach the research frontier, blogging, and writing popular books and articles.

Let me start by describing some of the most satisfying teaching experiences I've had. As an undergraduate at Cornell, I volunteered for a program called EYES (Encourage Youth Educate Society), which involved going to local high schools for math and computer science enrichment. Some of my shenanigans in this program became infamous. I unrolled a giant Turing machine tape on the floor, and had a student volunteer be the "tape head," wearing a posterboard that recorded her current state. I had boys and girls act out the algorithm for the Stable Marriage Problem (surprisingly, it didn't seem to embarrass them). I broke the students up into teams, and had them face off in an Iterated Prisoner's Dilemma competition. I also had them compete to connect thumbtacks on a board with the shortest possible length of string, in a Traveling Salesman championship. When one student asked me why all the mathematicians I was referring to were men, I returned the next day with a mini-lesson about Sophie Germain and her correspondence with Gauss, so the student in question would no longer have any excuse to tune out.

Later, as a second-year graduate student at Berkeley, I designed and (with Allison Coates) co-taught a two-credit course entitled "Physics, Philosophy, Pizza." We had about 35 students, mostly undergraduates, with majors as varied as computer science, neurobiology, and English. The basic concept was simple: we would convene once a week, eat 8-10 pizzas, and then spend two hours discussing Gödel's Theorem, P versus NP, quantum computing, special relativity, the Turing test, or other topics depending on student interest. The philosophical debates that often erupted were so intense that I was surprised a fistfight never broke out. On the last day of class, I was gratified when some of the students told me that this was the best course they'd taken at Berkeley.

This fall, I taught a course at Waterloo entitled "Quantum Computing Since Democritus," which was only a little bit more conventional than "Physics, Philosophy, Pizza." The attendees were mostly students and faculty from the Institute for Quantum Computing. Since many of them came from physics and engineering backgrounds, I made it my goal to show them the glories of a view of reality based on bits, logic, and computation. To that end, I started out with set theory and computability; proceeded through complexity theory, randomized algorithms, and cryptography; and finished with quantum mechanics, computational learning theory, interactive proofs, the anthropic principle, Newcomb's Paradox, closed timelike curves, the black hole information loss problem, and an "Ask Me Anything Day." The lectures were recorded, and many of the lecture notes are now on the web.[1]

In addition to these classroom experiences, I've long had a presence on the Internet as an archivist and popularizer of theoretical computer science. Four years ago, I created the Complexity Zoo,[2] an online encyclopedia of over 460 complexity classes that is now a standard reference for the field. I've also written popular articles about large integers,[3] primality testing,[4] quantum computing,[5] and other topics that are

[1] www.scottaaronson.com/democritus
[2] www.complexityzoo.com
[3] www.scottaaronson.com/writings/bignumbers.html
[4] www.scottaaronson.com/writings/prime.pdf
[5] www.scottaaronson.com/writings/highschool.html

available online; some of the most gratifying emails I receive are from high-school students who've found these articles. Today I write a popular blog[6] where I often explain and comment on recent work in quantum computing and complexity theory. I've decided to post the lecture notes from "Quantum Computing Since Democritus" to my blog as they became available, and I'm pleased that many readers around the world have been following the course.

In addition to these satisfying educational experiences, I've also had some profoundly *un*satisfying ones. Many of these came from taking courses in which the goal was not to learn anything about the world, but rather to "study the material" and thereby not harm one's GPA. In high school, this was the *only* type of course available, which is one reason why I left high school early. Though the situation was vastly better in college, by and large I still found more emphasis on "covering the material" than on "teaching" as I would understand the term.

But while "covering the material" might be dull for both the students and the instructor, doesn't it achieve an important purpose? For most students, doesn't it *work*? As a teaching assistant for an algorithms class at Berkeley, I gained a new perspective on these questions. I still remember having to grade hundreds of exams where the students started out by assuming what had to be proved, or filled page after page with gibberish in the hope that, somewhere in the mess, they *might* accidentally have said something correct. I remember students who spent great energy finding creative new ways to cheat, or who memorized the proofs from the CLRS textbook like sacred scripture, or who rushed to office hours on September 11, 2001, begging me for hints on the problem set.

Having learned from these experiences, how would I seek to make undergraduate teaching as rewarding as possible? Here are five ideas I'd like to try.

(1) **Teach Theoretical Computer Science as a Liberal-Arts Course.** Whenever students are taking our courses not out of intellectual curiosity but to fulfill the requirements for a degree, we will *always* be fighting an uphill battle. Not even the clearest, best-organized lecturer on earth can get around a fundamental fact: that the typical student headed for a career at Microsoft (i) will never need to know the Master Theorem, (ii) *knows* she'll never need to know the Master Theorem, (iii) doesn't *want* to know the Master Theorem, and (iv) will immediately *forget* the Master Theorem if forced to learn it.

Faced with this reality, I believe the best service we can provide for non-theory students is to teach them theoretical computer science *essentially as a liberal-arts course*. Aspiring computer scientists should know that they will not be glorified toaster repairmen, but part of a grand intellectual tradition stretching back to Euclid, Leibniz, and Gauss. They should know that, in Dijkstra's words, "computer science is no more about computers than astronomy is about telescopes"—that the quarry we're after is not a slightly-faster C compiler but a deeper understanding of life, mind, mathematics, and the physical world. They should know what the P versus NP question is asking and why it's so difficult. They should be regaled with stories of Turing's codebreaking in World War II, and of Gödel writing a letter to the dying von Neumann about "a mathematical problem of which your opinion would very much interest me." They should be challenged as to whether *they* would have had the insights of Edmonds, Cook, Karp, and Valiant, had they lived a few decades earlier. And they should be prepared to appreciate future breakthroughs in theoretical computer science on the scale of Shor's algorithm or PRIMES in P. From the discussions at websites such as Slashdot, it's clear that the "general nerd public" has enormous curiosity about these breakthroughs, and it's also clear that the level of curiosity greatly exceeds the level of understanding.

You might wonder whether it's *possible* to teach the sort of course I've described, and do so without compromising intellectual rigor. Fortunately, that question has been answered. There are several existing courses that are both extremely successful and similar in spirit to what I have in mind—including Steven Rudich's "Great Theoretical Ideas in Computer Science" at Carnegie Mellon[7], and

[6] www.scottaaronson.com/blog
[7] www.cs.cmu.edu/~15251/

Sanjeev Arora's "The Computational Universe" and Avi Wigderson's "The Efficient Universe" at Princeton.[8]

(2) **Raise the Ceiling.** Even at the undergraduate level, one hopes there will be a few students who are passionate about theoretical computer science for its own sake—and for these students, the goal is different. I will try to challenge these students to the utmost, nurture their competitive instincts, show them the ropes of academia, and then *let them loose on the research frontier as quickly as possible.* The *worst* thing one can do for these students is to saddle them with busywork or pointless prerequisite courses. In my experience, students who are motivated enough can usually pick up the background knowledge they need the way real researchers do: in the context of trying to solve actual research problems.

(3) **Modernize the Curriculum.** Regular grammars, pushdown automata, red-black trees, one-tape versus multi-tape Turing machines: all of these things would justifiably have seemed central thirty years ago, when our field was just getting off the ground. But today, these topics are *still* near the core of the undergraduate theory curriculum—and if students fail to understand why, can we honestly blame them? Why not base the undergraduate theory curriculum around topics that are no less simple and fundamental, but are closer to the frontier? As an example, instead of showing twenty NP-completeness reductions, why not show *two* reductions, and then spend the rest of the time talking about pseudorandom generators, PRIMES in P, natural proofs, or quantum computing?

(4) **Prove Non-Obvious Theorems.** Before students ever encounter a nontrivial theorem or algorithm, sometimes they have to spend most of a semester writing out painstaking proofs of intuitively obvious assertions. To me, this approach seems to be rooted in a mistaken assumption: that *the students already understand the point of mathematical rigor*. In reality, the innumerable examples of "parrot proofs"—NP-completeness reductions done in the wrong direction, "arguments" that look more like LSD trips than coherent chains of logic, and so on—make it clear that many students come to our courses appreciating mathematical rigor about as much as the ELIZA chat-bot appreciates romantic relationships. Students need to be *convinced* that mathematical rigor is something that is worth grasping in substance rather than merely emulating in its outward appearance. And the only way I know to convince them is to show them statements that are "obviously true," but that can nevertheless be shown to be false.

When I took Analysis of Algorithms with Jon Kleinberg my first semester at Cornell, Kleinberg didn't gradually work his way to something interesting. Instead, he devoted the very first lecture to two non-obvious algorithms: for the Stable Marriage problem and for Minimum Spanning Tree. As a result, even if the students didn't always follow along with Kleinberg (as, sometimes, we didn't), from the very beginning we understood the point of a formal approach to algorithms.

(5) **Reward Intellectual Honesty.** The countless "parrot proofs" I had to grade at Berkeley convinced me of another point: that for many students, the real problem is not that they don't know how to analyze algorithms or prove theorems. It's that they don't *know* that they don't know these things, or perhaps what it would even *mean* to know them. While it's easy to blame the students in such cases, to me it seems that our grading standards might inadvertently encourage intellectual dishonesty and laziness. For example, if a student writes an exam answer that is long, dense, and barely-comprehensible, many graders will give his "effort" the benefit of the doubt, and award at least partial credit. But if a student admits, clearly and frankly, that she has no idea how to solve a problem, then she'll immediately get a zero. If our goal is to produce *scientists*, who value clarity and intellectual honesty above facile impressiveness, then I believe our reward system has to change.

For concreteness, I wrote above about undergraduate courses. For lower-level graduate courses, I think the difference is only one of degree: one hopes that there will be more students who need to be pushed to

[8] www.cs.princeton.edu/courses/archive/spring06/cos116/, www.cs.princeton.edu/courses/archive/spring06/cos345/

the research frontier, and fewer who need to be taught what a proof is. Of course, for any instructor who's passionate about his or her own research, the teaching of upper-level seminar courses presents little problem, which is why I haven't talked about it at all.

Besides teaching courses, there are two other ways I hope to spread the gospel of theoretical computer science. The first is by taking on graduate students. My goal would be simple: to give my students the same kind of support and friendship that my adviser, Umesh Vazirani, gave me. Where other advisers might urge their students to take more courses, Umesh urged me to be more selective with courses, to focus not on jumping through administrative hoops but on tackling difficult open problems. "Concentrate on the high-order bits" is how he repeatedly put it—everything else would take care of itself. Umesh also knew how to tailor his advising approach to the individual student. With some of his students he collaborated closely, coauthoring multiple papers; others he gave an almost absurdly free rein, serving only as occasional sounding board. In my case, the very fact that Umesh gave me so much freedom inspired me to work harder so that I wouldn't disappoint him.

The last, crucial, component of my proposed teaching program consists of public outreach. I plan not only to continue blogging, but also to write popular books and articles about quantum computing, P versus NP, and anything else that I feel both passionate about and able to explain. Today, most theoretical computer scientists recognize that, if we are to survive as a field, then we'll have to do a better job of communicating our discoveries to the public. To that end, I'm currently doing final revisions for an article about the limitations of quantum computers, which will appear in *Scientific American*. I'm also writing a chapter for a popular book being put together by Michael Mitzenmacher. Crucially, I see these activities not as a drain on my time, but as an inextricable part of my future career. I've never been good at keeping secrets—so when I find myself privy to a body of knowledge that's every bit as deep as quantum mechanics or general relativity, but that most educated people have never even heard of, I can't imagine doing otherwise than sharing that knowledge with the world.