

Teaching Statement

Dr. Michael Dinitz

My teaching experience is, admittedly, quite limited. While a postdoc at the Weizmann Institute I have not done any teaching, and as a PhD student at Carnegie Mellon I was the TA for the undergraduate course on algorithms and for the graduate course on algorithms. Being a TA for the undergraduate course involved teaching weekly recitations, in which I presented new material as well as delved deeper into the material presented in lecture, and also answered any student questions about the lecture. There were no recitations for Graduate Algorithms, but on the other hand I participated much more in course development, including helping plan the syllabus and design homeworks and exams. I also ran a course blog, which I used both to clarify questions about homeworks and lectures as well as to point out interesting current algorithmic research and applications that I thought the students might be interested in. I held office hours for both classes, which many students told me were extremely useful and were consistently well-attended.

Despite my lack of teaching experience, I have a deep and abiding interest in undergraduate education. This is due in part to my own experience as an undergraduate at Princeton University, an institution that prides itself on delivering a high-quality undergraduate education while simultaneously conducting world-class research. While at Princeton I was exposed to the groundbreaking introductory computer science sequence in use there. These classes were designed to appeal not only to undergraduates with previous programming experience (which is relatively common these days), but also to students with no computing experience whatsoever. Obviously many students have used computational tools in their math and science classes before, so might be familiar with basic programming techniques, but many bright students can be turned off from computer science by introductory classes that are too focused on the mechanical act of programming. Instead, students can be turned into extremely successful computer science majors and (eventually) software engineers if they are shown early on why computer science is so interesting and useful, and are exposed to industrial-strength programming and software engineering throughout their time but not necessarily as a goal in and of itself (at least early on).

For example, the General Computer Science course that I took at Princeton obviously spent a significant amount of time covering basic programming in Java, but also served as a teaser for theoretical computer science (introducing different sorting algorithms and doing basic analyses of them) and computer architecture (introducing a toy architecture model and discussing how C and Java commands and data structures are actually executed and stored). Introducing these more advanced subjects early on made the subject of computer science seem more interesting than other majors, as it showed both the “science” and the “engineering” sides of CS. I should note that this sales job worked on me – I was originally intending on majoring in either math or physics, but was persuaded by this class to switch to CS.

This experience is what motivates my fundamental philosophy of teaching: show students computer science is the most interesting subject in the world, from both a practical point of view and from a more scientific or mathematical point of view. The best way to get students to learn is to get them excited, and for some students this happens by exposure to the engineering side of computer science (e.g. the satisfaction of building an actual working program that does something “cool”)

while others may be more excited by the scientific side or connections to other fields (e.g. the ability to solve interesting questions by using powerful computational techniques). Thus I believe it is important to show the different sides of computer science in all classes, even upper-level ones. I personally find most aspects of computer science to be inherently interesting, so I find it both useful and rewarding to share my excitement with the students of whatever class that I am teaching. This, I believe, makes me an extremely effective teacher.

Of course, there are still fundamental issues of pedagogy that cannot be handwaved away through student excitement and engagement. In order to learn more about these issues, I audited the course “Introduction to Computer Science Education” at Carnegie Mellon. This course covered both general pedagogical techniques and research as well as computer science-specific educational issues. This course was mostly focused on introductory computer science education, but it gave me valuable experience in curriculum design and assessment. The main thing that it taught me, though, was that not all students are destined to be academics. This is a fact that I think we too often forget: the teaching strategies that worked for us, as high-achieving students who found the subject so inherently interesting that we ended up in graduate school, will not necessarily work for all students. Nevertheless, all students can be engaged and be made excited, and if we as teachers can achieve that then we are much of the way towards effective teaching.

Finally, a discussion of undergraduate education would not be complete without a discussion of undergraduate research. I strongly believe that conducting research is a valuable educational experience for students, but moreover that undergraduates really are capable of conducting real research. We in computer science have the advantage of a young field, and there are still plenty of problems on which a bright, motivated undergraduate can actually make progress. My belief in undergraduate research grew out of my own experiences: as an undergraduate I spent two summers at NSF Research Experiences of Undergraduates (which resulted in two papers) as well as completing a research thesis my senior year. These experiences were transformative, as they taught me that I really do enjoy research. This is an experience that any undergraduate who is considering graduate school should have, as far too many students who do well in classes decide to go to graduate school and only later find out that they do not actually enjoy research.

There are, in my view, two main types of research that I would like to do with undergraduates. The first, and simpler, type of research is simply writing some code. For example, I think it would be great for a student to implement some of the theoretical algorithms that I have designed in my own research to see how they do in practice, and maybe find out how best to tweak them to make them practical. This is something that almost any student can accomplish, so it builds their confidence in their own abilities, while at the same time being real, publishable research. On the other hand, more advanced students can try to do real theoretical computer science work. There are still many algorithmic problems that, if solved, would result in a paper, but aren’t necessarily being actively worked on by the algorithms community. Related to my own work, variants of the techniques that I used for some variants of graph spanners can almost certainly be used to solve other variants and similar problems, and a mathematically sophisticated undergraduate could definitely use these techniques to write a (relatively simple) paper.

While my teaching experience is limited, I have received positive feedback from what teaching I have done, and I believe that by encouraging excitement and engagement in every class I am an extremely effective teacher. My background as an undergraduate who did a significant amount of research has taught me how to do effective research with undergraduates, and has made me a strong proponent of undergraduate research. I look forward to engaging with the student population both inside the classroom and out, and I only hope that I will be able to influence them the same way that my professors affected me, and show them how great and interesting a discipline computer science is.