

## *Intuition and Dedication* – My Philosophy for Teaching Theoretical Computer Science

Liang Huang, Department of Computer and Information Science

I teach courses in Theoretical Computer Science, perhaps one of the most *abstract* and *mathematically rigorous* fields out of all subjects taught in college. This is also evidenced by the *Penn Course Review*, where the two courses I taught (CSE 262 and CSE 320) both received a difficulty rating higher than 3.5 (for previous offerings) and were both listed in the “most difficult courses across the University”.

These courses also form the core of the undergraduate CS programs. So how do I teach them? My basic idea is to make them more attractive to ordinary undergraduates, instead of just the more advanced students. In summary, my method consists of three steps: *intuition, illustration, and translation*.

Often students first came to my recitations or office hours saying: “*I am considering dropping this course...*” “*I am really struggling with the material... the stuff professor did in class was a bit too abstract for me... But I found the textbook even harder, you know, full of notations and definitions*”. This is true. So, rather than going over the concepts and definitions, I first give them *intuitive ideas*, often using real-world examples. For instance, when I taught *topological-sort*, I motivated the problem as “finding a valid ordering of taking courses with respect to their prerequisites”, a problem that every student encounters every semester. Apparently they understood it instantly and liked the analogy a lot.

I believe intuitions are best explained by illustrations and animations. It is well-known that many human-beings recognize visual information faster than textual. I try and draw various pictures on the blackboard to visualize almost every concept. When teaching *finite-automata*, however, I used software to show animations of the node-elimination algorithm, which I think is far more intuitive than blackboard teaching. Students now agree that once they have the “geometric interpretations” in mind, they can reach a better understanding of the original problem.

But intuitions are *not* enough for theoretical computer science. So I also teach students to translate their ideas back into the mathematical language. They often come to my office hours saying “*I think I do know the stuff, but just found it so hard to write them down rigorously*”. In this case, I always let them “show me the picture” first, and then help them hand-by-hand with the translation step, until everything is written formally, without any ambiguity or vagueness.

On the non-technical side, I find it important and rewarding to devote a lot of time to the undergraduates. The official responsibilities for a CSE 262 TA are just grading and office hours, but I volunteered to teach a recitation every one or two weeks and gave extra review lectures before the exams. I also made up my original “extra review problems” in addition to the professor’s practice midterms. The extra work finally pays off – It has never been more pleasure than getting up and having several emails sliding in my Inbox saying something like “*Thanks for your review lectures! They are extremely helpful for the midterm!*”

There are always some students who need extra support. They are usually struggling with the course and slower than other students to understand the materials. To encourage them, I always emphasize that no question is stupid in my recitations. Also, I usually stay an extra 15-20 minutes after each office hour for students with a lot of extra questions. I had two students who did poorly in the first midterm of CSE 262. At their request, I held special office hours for them in weekends and they both improved substantially in the second midterm. The professor was quite impressed by their progress.

Towards the end of semesters, I often heard students say, “*I really enjoyed this class. It’s demanding but turns out to be interesting.*” I just hope that my effort and dedication will help more students entering the world of theoretical computer science and let them enjoy the fun of thinking like a computer scientist.