Teaching Statement

Isaac Grosof

1 Teaching Experience & Philosophy

Over the last fourteen years, I have been teaching computer science and related fields. Throughout high school, I tutored other high school students in mathematics. In my undergrad, I ran weekly recitations as a TA for advanced undergraduate algorithms (MIT 6.1220) for two semesters and provided 1-on-1 in-class instruction as a lab assistant for fundamentals of programming (MIT 6.1010). As a graduate student, I have been a TA for two computer science theory courses, graduate performance modeling (CMU 15-857), where I ran weekly recitations, and graduate advanced algorithms (CMU 15-850). Based on my teaching experience, I have developed a philosophy of teaching which incorporates the following principles:

Convey Intuition I believe the primary goal of classroom instruction should be to convey the intuition behind the results, rather than just stating the results. Conveying formal definitions, theorems, or formal proofs, while important, need not take center stage. When I have run recitations as a TA for both MIT 6.1220 and for CMU 15-857, I have repeatedly encountered situations where many students initially couldn't solve a homework problem, then read the solution, and then were no closer to being able to generate that solution themselves. As a teacher, it is my responsibility to convey the thought process that leads to the solution, starting with a rough, intuitive guess to the solution, and then refining that guess into a proof, an algorithm, or a program.

Understand Students' Thinking Effective teaching is a two-way street: I need to understand where my students are coming from and what they are absorbing or having trouble with, in order to best help them learn. This is especially important when students are becoming lost, either during a lecture or over the course of a semester. As a TA for CMU 15-857, when such a situation arose, I would ask students to form into groups of four, discuss the current topic for a couple of minutes, and then ask people from several groups to report. I found this was very helpful in making sure that I understood what students were getting, and what I needed to delve into further. Fostering a comfortable and friendly classroom spirit is key to two-way communication, so that students feel confident about asking questions and expressing confusion.

Use Handouts to Provide Structure I believe in providing handouts to students to accompany a lecture, to provide structure to students' process of following along with the lecture. These handouts both provide an outline of the content of the class, as well as space for students to take notes and engage with the material. I believe effective note taking is challenging, but essential to internalizing material. Therefore, I want to guide students through the process, helping students focus throughout class and ensuring that students don't fall behind the material. As a TA for CMU 15-857, I created such handouts for my recitations, and provided my handouts to future TAs.

2 Teaching Interests

Undergraduate and Graduate Classes I am excited about teaching a variety of courses. I could teach a course in **programming**, such as an introductory programming course in Python, fundamentals of programming, or a more specialized course in a language such as Rust. I could teach a course in **probability**, such as courses on probabilistic modeling, simulation, statistics, stochastic processes, or Markov chains. I could teach a course on **algorithms**, such as courses

on data structures or randomized algorithms. I could teach a course on **proof techniques**, such as basic proof concepts, discrete math, or an overview of theoretical computer science. I could teach a course on **performance** of computer systems, such as performance modeling, applied probability, performance analysis, or queueing theory. I could teach a course on **complexity and computability theory**, including restricted computation classes, reductions, and time and space bounds. With a semester to prepare, I could additionally teach courses on optimization, software engineering, machine learning and data analysis, game theory, computer systems, cryptography, and quantum computation.

I am looking forward to teaching students anywhere from freshmen to advanced graduate students. For instance, I could teach a proof techniques course for freshmen focusing on topics such as induction and basic combinatorics, or an advanced graduate course covering topics such as concentration bounds, dimensionality reduction, and stochastic coupling. I have extensive experience teaching younger students in particular, having taught middle school and high school students as an undergraduate and graduate volunteer and in high school as a tutor of high school students.

I am also interested in teaching online classes, whether hybrid or completely online. I have experience as a TA for an online class, when I was a TA for CMU 15-850. Much earlier, my first university-level class was an online differential equations class through Stanford. I believe the best way to teach an online course involves a mixture of premade materials (e.g. slides) with live-written content. Having premade materials keeps the pace of the class from slowing down, while live writing keeps the material engaging and flexible to the students' needs.

Proposals for New Courses

Large-scale systems: Queueing, Scheduling and Performance Modeling (advanced undergrad and intro grad) In large-scale systems, such as datacenters, manufacturing pipelines, or call centers, tasks spend a massive amount of time waiting in queues. This course will teach students to model such systems using queueing theory, understand the sources of waiting time and inefficiency in such systems, and provide tools, such as resource allocation and scheduling, to alleviate waiting time and wasted capacity. The course will introduce students who are building systems to theoretical ideas that can guide their designs, and theoretically-minded students to practical systems that are worth modeling and analyzing.

Simulation as a Tool (undergrad) Simulation is a key technique for understanding and optimizing the performance of complex systems, allowing rapid exploration of a variety of designs and approaches. This course will teach students how to model and efficiently simulate complex systems, create simulations at varying degrees of fidelity, and verify the faithfulness of a simulation against trace data. The course will also cover confidence intervals around simulation results, the effects of heavy tails on confidence intervals, and the convergence of simulation to long-term behavior. Finally, I will teach students how to implement simulations to run efficiently using an event-driven programming paradigm, and efficiently gather and report results from those simulations.

Proofs: from Exploration to Intuition to Result (intro undergrad) To an experienced theorist, the process of proving a novel result often starts with feeling out a problem, then gathering a rough idea of why a theorem might be true, and finally selecting from a repertoire of proof outlines to create a formal proof. Newer students are often unaware of all the steps prior to the final formal proof, and are mystified as to how one stumbles upon the correct answer. This course will guide students through the entire proof process, particularly benefiting students who would otherwise struggle with basic proofs.