# Teaching Statement

Ranjit Jhala

I am excited about teaching for two reasons.

The first is that I love teaching. I discovered this when in high school I had a chance, as part of a "pupil-teacher" competition, to teach a math class to fifth graders. Since then I've been lucky enough to get several other opportunities: while in high school and as an undergraduate, I tutored underprivileged children in math and at Berkeley, I've been a teaching assistant(TA) for the undergraduate Algorithms and Computability classes, during which I got the chance to lecture a couple of times. While I cannot deny that I enjoy the performance aspect of teaching, what pleases me the most is when students "get it"; an idea, a proof, an algorithm, be it in class, section, or especially when they solve difficult problems in exams.
To me a teacher's task is to teach students *how to think*. It is *not* to teach them fast algorithms for sorting or searching, how parsers work or code is generated, proofs of undecidability or the design of an operating system. All of those are merely examples, that must be deconstructed, to reveal the anatomy of the solution: its basic parts, how they work together, and what approaches would fail. The goal is to teach students *how to solve problems* by showing them how to effectively navigate the vast search space of possible solutions. This is done by acquainting them with tools which identify the characteristics of a problem and which may be combined to arrive at solutions, which in turn enables them to devise new tools for tackling different problems. To be intimately acquainted with these tools and approaches, not only must students know when a particular approach to a problem works, but also when something doesn't work. This means they must be constantly urged to think about what would happen if something was tweaked a little, or if instead of designing a system or subroutine or a proof one way, it was designed another way. I design lectures, discussion sections, assignments and examinations with this principle in mind.
The high point of my stint as TA for the undergraduate algorithms was to see, near the end of the semester, one of my discussion sections, which when quizzed about a problem at the start of the semester would flail helplessly and toss out incoherent suggestions, preempt my lecture by working out, in class, as a team, Christofides approximate algorithm for the Travelling Salesman Problem. I began by hinting that they think of spanning trees, and was then amazed how the class arrived at the final algorithm by iteratively suggesting an idea, finding the hurdle raised by the idea, and then finding how to surmount the hurdle – with my role simply being that of writing what each person said on the board and ensuring that they spoke in turn!

The second reason why I am excited about teaching is its connection with my research goal of methods for building systems with guaranteed reliability. Ultimately, systems have to be designed and built keeping the goal of reliability in mind, and for this we need better development methodologies. The success of a methodology depends on how widely it can be accepted, which in turn depends on ways by which people can be taught to build systems using that methodology. Hence, I am interested not only in designing and teaching graduate and undergraduate classes about Software and Reliability engineering – including material on techniques for analyzing systems and development methodology, but also in devising ways of emphasizing the requirement of *guaranteed reliability* (in the way efficiency and execution time are highlighted) in every computer science course.

I won the pupil teacher competition in high school, and, for my first innings as a TA, received the EECS department's best teaching assistant award for the year 2002-03.

For these reasons, I am eager and qualified to take on the challenges of designing and teaching both undergraduate classes, especially, *Programming Languages, Compilers, Software Engineering, Data Structures, Algorithms*, and *Formal Languages, Automata Theory and Computability,* as well as graduate classes: particularly *Advanced Programming Languages*, *Verification (Model checking, Automated Deduction, Program Analysis)*, *Embedded/Concurrent Programming* and *Software Reliability Techniques.*