# Teaching Statement

Yan Gu

**Teaching experience.** Over the ten years that I have been teaching, I have always enjoyed the experience. My teaching experience started during my final year of high school. As an elite competitor in the programming contest (Olympiad in Informatics) in China, I was invited to teach algorithms and the history of computing at another high school in my city. That was a great experience, which sparked my passion for teaching. Even though my undergrad university does not offer teaching opportunities for undergrads, I still managed to get the chance to teach algorithms in various high schools and neighboring universities in Beijing, as well as at some national-level camps for high school students. Among these experiences, I am particularly satisfied with two achievements. One is the best teaching award at the national camp of China Olympiad in Informatics in Spring 2010, where I gave a lecture on chess-playing and related algorithms (pre-AlphaGo era). The other one was also in Spring 2010, when I taught a semester-length course on algorithm design at Beijing Institute of Technology (BIT) in preparation for the ACM Collegiate Programming Contest. The course was a great success, and the BIT team won the first gold medal (ranking 7th among all schools) in their history in the Asian regional contests.

My PhD experience at CMU promoted me to a higher level in both research and teaching. Although the mandatory teaching task for PhDs is for 2 semesters, I have actually been involved in teaching in 7 semesters—4 semesters (F13, S14, F14, S15) for 15-295: Competition Programming and Problem Solving (programming course, undergrad level), 2 semesters (F15, S18) for 15-853: Algorithms in the Real World (theory course, PhD level), and 1 semester (Summer 17) for 15-418/618: Parallel Computer Architecture and Programming (system course, undergrad and master level). Other than making and grading homework and exams, I also delivered many lectures in all these courses. These experiences helped establish my teaching skills and teaching philosophy and bolstered my confidence about becoming a successful teacher in the future.

In addition to teaching courses, I also had the opportunity to mentor several bachelor's and master's theses and summer interns from CMU, KIT and Tsinghua. While discharging these responsibilities, I learned to communicate and guide students with different levels of knowledge, which will be beneficial for me for advising students in time to come. Mentoring students has also helped me become better at identifying interesting problems and related resources to look at. My research collaboration with the students has led to several ongoing papers in preparation.

**Teaching interests.** Given my widespread research interests, I am capable of teaching almost all theory courses for undergrads and can offer several different PhD-level theory/algorithm courses. I can also teach many system and programming courses (especially those related to parallelism), as well as application courses such as computer graphics, image processing, multimedia/visual computing. In particular, based on my own teaching experience, I would especially like to teach the following two topics since I believe that they are in timely need.

**Parallel thinking.** Unlike in 2005, nowadays parallel machines are prevalent from smartphones to servers. This change has been so rapid that parallelism is no longer an option, but a necessity out of performance considerations on almost all computing platforms. Consequently, most schools offer at least one parallel programming course that teaches basic techniques for utilizing libraries or tools to support parallelism, scheduling, architectural challenges such as memory consistency, and dealing with concurrency issues.

Although these low-level details are extremely crucial, I believe that "thinking in parallel" is also very essential and should be taught prior to the low-level details to support parallel computing. Taking the classic CS curriculum as an example, although the basic concepts in compilers, systems, and architectures are important, students should first learn algorithms as high-level ideas, and then consider the low-level support for efficient computing. Similarly, directly parallelizing complicated sequential code is generally very difficult, but it should be relatively easy to be reconstructed based on the parallel primitives and concepts such as prefix sum, divide-and-conquer, and priority updates. Meanwhile, I believe that considering parallelism at the beginning can help to comprehend algorithms at a more abstract level, and provide another angle to understand and design algorithms. As a result, students in the modern era should be exposed to "parallel thinking" first, and the techniques to implement the ideas can follow later.

To be more specific, I suggest a three-pronged curriculum of parallel computing: first algorithms and programming, and then system and architecture, in a top-down manner a similar manner to the existing CS curriculum. They

should all be mandatory for all CS undergrads. The course workload remains the main challenge. Namely, adding multiple "heavy" mandatory courses is impossible, so it is crucial to merge the new materials into the existing curriculum. This is my long-term plan, which may take years to implement, and require expertise in different areas. Given my previous teaching and research experience in all three areas, I am confident that I am the right person to help bring about such a change. Considering that today even an iPhone has eight processors, exposing CS students to the parallel world is of great importance.

**Competitive Programming.** The programming competitions are welcome from high school level to college level. Many universities offer courses to prepare for such competitions, which are also very helpful for students in finding jobs. Meanwhile, having a good programming team bolsters the school's reputation. As an international student, I got to know many US schools by the award lists. Again, based on my previous teaching experience, I can be a suitable candidate to teach or assist with such a course for an undergrad program. For me, there is another main reason for such a course.

In the typical algorithm courses, the professor shows interesting algorithms in the lectures and proofs of work bounds (usually regarding time complexity), and students finish paper assignments usually on proving the correctness and work bounds for some variant cases. Then paper-based exams are taken several times during the semester. At the meantime, programming courses either teach programming languages or assign students to implement (part of) an OS, a compiler, a ray tracer, an image-processing system, etc.

However, there is usually a gap between theory and practice—modeling and solving real-world problems and implementing the algorithms. Such *problem-solving ability* is crucial. For example, all major IT companies nowadays ask the job interviewees to write code for algorithmic problems in a very limited time, and treat this as an indicator of the candidates' ability to solve the real-world problems they encounter in practice. One option is to let the students solve 100+ problems on LeetCode on their own, but the better way is to provide systematic training to inculcate such skills, since the ability is useful both during interviews and in work scenarios. A course for competitive programming exactly meets this need—limited time to implement algorithms using known knowledge for new problems.

According to my conception, such a course should be a "light" course. The students come together once a week. The lecturer teaches some key points in designing and implementing a certain type of algorithm, and follows up with several related problems for the students to finish during the lecture. The students doing well over a semester can be selected to participate in competitions, and winning an award is a significant bonus point on their resumes when applying for an industrial job or for a graduate school position related to algorithms or programming. But again, the primary goal is to provide training to develop problem-solving ability. I have run a similar course 15-295 at CMU for two years (four semesters), and it was extremely rewarding that many students found this course fruitful. I recommend strongly that every undergraduate student should take such a course for one or two times.

**Teaching summary.** My broad research interests and teaching experience equip me to teach classes on various topics, as well as to establish several new courses which I believe to be the need of the hour. I would love to teach future generations with my knowledge, and my passion.