Course Objectives and Outcomes

Course Objectives

This course assumes that students know how to analyze simple algorithms and data structures from having taken 6.006, and introduces students to design of computer algorithms, as well as analysis of sophisticated algorithms. Upon completion of this course, students will be able to do the following:

- Analyze the asymptotic performance of algorithms.
- Demonstrate a familiarity with major algorithms and data structures.
- Apply important algorithmic design paradigms and methods of analysis.
- Synthesize efficient algorithms in common engineering design situations.
- Understand the difference between tractable and intractable problems, and be familiar with strategies to deal with intractability.

Course Outcomes

Students who complete the course will have demonstrated the ability to do the following:

- Argue the correctness of algorithms using inductive proofs and loop invariants.
- Analyze worst-case running times of algorithms using asymptotic analysis. Compare the asymptotic behaviors of functions obtained by elementary composition of polynomials, exponentials, and logarithmic functions. Describe the relative merits of worst-, average-, and best-case analysis.
- Analyze average-case running times of algorithms whose running time is probabilistic. Employ indicator random variables and linearity of expectation to perform the analyses. Recite analyses of algorithms that employ this method of analysis.
- Explain the basic properties of randomized algorithms and methods for analyzing them. Recite algorithms that employ randomization. Explain the difference between a randomized algorithm and an algorithm with probabilistic inputs.
- Describe the divide-and-conquer paradigm and explain when an algorithmic design situation calls for it. Recite algorithms that employ this paradigm. Synthesize divide-and-conquer algorithms. Derive and solve recurrences describing the performance of divide-and-conquer algorithms.
Handout 2: Course Objectives and Outcomes

- Describe the dynamic-programming paradigm and explain when an algorithmic design situation calls for it. Recite algorithms that employ this paradigm. Synthesize dynamic-programming algorithms, and analyze them.

- Describe the greedy paradigm and explain when an algorithmic design situation calls for it. Recite algorithms that employ this paradigm. Synthesize greedy algorithms, and analyze them.

- Explain the major graph algorithms and their analyses. Employ graphs to model engineering problems, when appropriate. Synthesize new graph algorithms and algorithms that employ graph computations as key components, and analyze them.

- Describe a linear program and cite problems that can be solved using linear programming. Reduce problems to linear programming formulations. Understand the complexity of various linear programming approaches.

- Explain basic complexity classes such as P, NP, and NP-complete, and be able to use analysis and reduction techniques to show membership or non-membership of a problem in these classes.

- Describe approaches to speeding up algorithms by making use of distributed and/or parallel computation.

- Explain the concept and basic examples of basing cryptographic security on hard computational problems.

- Understand and explain approaches to dealing with problems that are NP-complete such as the design of heuristic, approximation, or fixed-parameter algorithms.