Conclusions

Algorithmic toolkit

- Algorithms is like household repairs – you see a problem, you want to have the right tools to fix it. (Don't use a hammer to tighten a screw.)
- One goal of this class: give you a basic algorithmic toolkit.
  - Greedy algorithms
  - Divide and conquer
  - Dynamic programming
  - Network flows
  - Linear programming
- As you gain experience, finding the right tool for the job becomes easier. (I often start with the easiest, greedy, and try the more complex tools when the easier tools fail. You may have your own approach.)
- This toolkit is just the start – more tools out there, which you'll learn in other classes....

Models of computation and formal systems

- Word-RAMs, TMs, DFAs, NFAs, Context-Free Grammars, Regular Expressions, ...
- How to to formally model computation
- Asymptotic perspective (fixed program for all input lengths)
- Design your own models as circumstances demand (eg interactive/distributed computation, randomized computation, biological systems, economic systems)

Models of problems

- We also think about how we model individual problems.
- Graphs are one general model useful for many different problems.
- Linear programming is another general approach to model problems.
- But more generally, in practice, problems don't come in quite the nice package you usually see in class. The first step is to think about the problem, and find the right model.

Classification of computational problems

- Positive results: regular, polynomial-time, decidable, Turing-recognizable languages
- Negative results: non-regular, NP-complete(?), undecidable, non-recognizable languages
- Notion of reduction between problems
- The systematic methodology for proving things impossible is one of the most important achievements of computer science.
- NP-completeness is one of the most important "exports" of computer science to the rest of science.

The power of reduction

- Reductions are used for both positive results (designing algorithms) and negative results (proving hardness).
- Easiest way to solve a problem – reduce it to something you already can solve.
- (And then go back and see if you can make it more efficient.)

Understanding Intractability

- Many important problems are NP-complete (or even undecidable).
- But also some great positive results in algorithm design
  - E.g. poly-time algs for LINEAR PROGRAMMING, PRIMALITY TESTING, POLYNOMIAL FACTORIZATION, NETWORK FLOWS, ...
- What does NP-completeness mean? (assuming P $\neq$ NP)

- No algorithm can be guaranteed to solve the problem perfectly in polynomial time on all instances
- Exhaustive search is often unavoidable
- Mathematical nastiness: no nice, closed form solutions.

Coping with Intractability

What if you need to solve an NP-complete (or undecidable) problem?

- Ask your boss for a new assignment. :-)
- Identify additional constraints that make the problem easier (eg bounded-degree graphs, ILP with fixed number of variables, 2-SAT)
- Approximation algorithms, e.g. find a TSP tour of length at most 1.01 times the shortest.
- Average-case analysis — analyze running time or correctness on "random" inputs. (Often hard to find distribution that models "real-life" inputs well.)
- Heuristics — techniques that seem to work well in practice but do not have rigorous performance guarantees.
- Change the problem
  - Instead of verifying that general programs satisfy desired security properties (undecidable), ask programmers to supply programs with (easily verifiable) "proofs" that the properties are satisfied
  - Change the programming language (CS 152, CS 252r)

Theory of Computation after CS 125

- Algorithms
  - CS 224: Advanced Algorithms
  - CS 222: Algorithms at the End of the Wire
  - CS 223: Probabilistic Analysis and Algorithms (Spring '15)
  - AM 121: Introduction to Optimization — Models and Methods
  - AM 221: Advanced Optimization
- Computational Complexity

  - CS 221: Computational Complexity
  - CS 225: Pseudorandomness (Spring '15)

- Cryptography and Privacy
  - CS 127: Introduction to Cryptography
  - CS 227r: Topics in Cryptography & Privacy

- Algorithmic Economics and Social Science
  - CS 186: Economics and Computation
  - CS 284r: Topics on Computation in Networks and Crowds
  - CS 286r: Topics at the Interface between Computer Science & Economics

- Other
  - CS 228: Computational Learning Theory
  - CS 229r: Topics in the Theory of Computation (Spring '15 on Biology & Complexity)
  - AM 106/206: Applied Algebra
  - AM 107: Graph Theory and Combinatorics
  - Math 155r: Combinatorics
  - ES 250: Information Theory
  - Math 141: Introduction to Mathematical Logic
  - Philosophy 144: Logic & Philosophy
  - Many courses in CS & Math at MIT.

Theory Research

- Theory of Computation research group
  - Group webpage: `http://toc.seas.harvard.edu/`
  - Weekly seminar: usually with pizza!
  - `http://toc.seas.harvard.edu/seminar.html`

- Also MIT Theory of Computation group and its seminars
  - `http://theory.csail.mit.edu/`
- Many research opportunities

Connections to the Rest of CS (Partial List)

Circuit Design (CS 141)	Finite Automata
Parsing + Compiling (CS 153)	Context-free Grammars
Programming	Finite Automata (Regular Expressions)
Languages (CS 152)	Formalization in General
Natural Language +	Context-Free Grammars
Linguistics (CS 187)	Finite Automata
Program Analysis +	Uncomputability
Synthesis (CS 153)	
Artificial Intelligence (CS 181,182)	Formal Systems, Logic
Pretty much everywhere	Algorithms of various sorts