Design and Analysis of Algorithms I

Introduction

About The Course

Course Topics

- Vocabulary for design and analysis of algorithms
- Divide and conquer algorithm design paradigm
- Randomization in algorithm design
- Primitives for reasoning about graphs
- Use and implementation of data structures

Course Topics

- Vocabulary for design and analysis of algorithms
  - E.g., “Big-Oh” notation
  - “sweet spot” for high-level reasoning about algorithms

Course Topics

- Vocabulary for design and analysis of algorithms
- Divide and conquer algorithm design paradigm

Course Topics

- Vocabulary for design and analysis of algorithms
- Divide and conquer algorithm design paradigm
  - Will apply to: Integer multiplication, sorting, matrix multiplication, closest pair
  - General analysis methods ("Master Method/Theorem")

Course Topics

- Vocabulary for design and analysis of algorithms
- Divide and conquer algorithm design paradigm
- Randomization in algorithm design
  - Will apply to: QuickSort, primality testing, graph partitioning, hashing.

Course Topics

- Vocabulary for design and analysis of algorithms
- Divide and conquer algorithm design paradigm
- Randomization in algorithm design
- Primitives for reasoning about graphs
  - Connectivity information, shortest paths, structure of information and social networks.

Course Topics

- Vocabulary for design and analysis of algorithms
- Divide and conquer algorithm design paradigm
- Randomization in algorithm design
- Primitives for reasoning about graphs
- Use and implementation of data structures
  - Heaps, balanced binary search trees, hashing and some variants (e.g., bloom filters)

Topics in Sequel Course

- Greedy algorithm design paradigm

Topics in Sequel Course

- Greedy algorithm design paradigm
- Dynamic programming algorithm design paradigm

Topics in Sequel Course

- Greedy algorithm design paradigm
- Dynamic programming algorithm design paradigm
- NP-complete problems and what to do about them

Topics in Sequel Course

- Greedy algorithm design paradigm
- Dynamic programming algorithm design paradigm
- NP-complete problems and what to do about them
- Fast heuristics with provable guarantees
- Fast exact algorithms for special cases
- Exact algorithms that beat brute-force search

Skills You'll Lean

- Become a better programmer

Skills You’ll Lean

- Become a better programmer
- Sharpen your mathematical and analytical skills

Skills You'll Lean

- Become a better programmer
- Sharpen your mathematical and analytical skills
- Start "thinking algorithmically"

Skills You'll Lean

- Become a better programmer
- Sharpen your mathematical and analytical skills
- Start "thinking algorithmically"
- Literacy with computer science's "greatest hits"

Skills You'll Lean

- Become a better programmer
- Sharpen your mathematical and analytical skills
- Start "thinking algorithmically"
- Literacy with computer science's "greatest hits"
- Ace your technical interviews

Who Are You?

- It doesn't really matter. (It's a free course, after all.)

Who Are You?

- It doesn't really matter. (It's a free course, after all.)
- Ideally, you know some programming.

Who Are You?

- It doesn't really matter. (It's a free course, after all.)
- Ideally, you know some programming.
- Doesn't matter which language(s) you know.
  - But you should be capable of translating high-level algorithm descriptions into working programs in *some* programming language.

Who Are You?

- It doesn't really matter. (It's a free course, after all.)
- Ideally, you know some programming.
- Doesn't matter which language(s) you know.
- Some (perhaps rusty) mathematical experience.
  - Basic discrete math, proofs by induction, etc.

Who Are You?

- It doesn't really matter. (It's a free course, after all.)
- Ideally, you know some programming.
- Doesn't matter which language(s) you know.
- Some (perhaps rusty) mathematical experience.
  - Basic discrete math, proofs by induction, etc.

- *Excellent free reference: "*Mathematics for Computer Science", by Eric Lehman and Tom Leighton. (Easy to find on the Web.)

Supporting Materials

- All (annotated) slides available from course site.

Supporting Materials

- All (annotated) slides available from course site.
- No required textbook. A few of the many good ones:
  - Kleinberg/Tardos, *Algorithm Design*, 2005.
  - Dasgupta/Papadimitriou/Vazirani, *Algorithms*, 2006.
  - Cormen/Leiserson/Rivest/Stein, *Introduction to Algorithms*, 2009 (3rd edition).
  - Mehlhorn/Sanders, *Data Structures and Algorithms: The Basic Toolbox*, 2008.

biggest influence on instructor

freely available online!

Supporting Materials

- All (annotated) slides available from course site.
- No required textbook. A few of the many good ones:
  - Kleinberg/Tardos, *Algorithm Design*, 2005.
  - Dasgupta/Papadimitriou/Vazirani, *Algorithms*, 2006.
  - Cormen/Leiserson/Rivest/Stein, *Introduction to Algorithms*, 2009 (3rd edition).
  - Mehlhorn/Sanders, *Data Structures and Algorithms: The Basic Toolbox*, 2008.
- No specific development environment required.
  - But you should be able to write and execute programs.

Assessment

- No grades per se. (Details on a certificate of accomplishment TBA.)
- Weekly homeworks.
  - Test understand of material
  - Synchronize students, greatly helps discussion forum
  - Intellectual challenge

Assessment

- No grades per se. (Details on a certificate of accomplishment TBA.)
- Weekly homeworks.
- Assessment tools currently just a “1.0” technology.
  - We’ll do our best!
- Will sometimes propose harder “challenge problems”
  - Will not be graded; discuss solutions via course forum