

CS 7200-02, FALL 2015

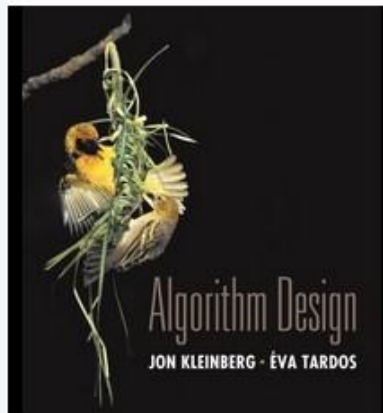
**ALGORITHM DESIGN
AND
ANALYSIS**

CEMIL KIRBAS



Lecture Slides (by Kevin Wayne)

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos/>



ALGORITHM DESIGN
1. Representative Problems
2. Algorithm Analysis
3. Graphs
4. Greedy Algorithms
5. Divide and Conquer
6. Dynamic Programming
7. Network Flow
8. Intractability
9. PSPACE
10. Limits of Tractability
11. Approximation Algorithms
12. Local Search
13. Randomized Algorithms
EXTRA TOPICS
Data Structures
Linear Programming

LECTURE SLIDES FOR ALGORITHM DESIGN

These are a revised version of the lecture slides that accompany the textbook *Algorithm Design* by Jon Kleinberg and Éva Tardos. Here are the original and official version of the [slides](#), distributed by Addison-Wesley.

TOPIC	SLIDES	READINGS	DEMOS
Introduction (administrative information)	1up · 4up	Preface · ToC	—
Stable Matching (Gale-Shapley)	1up · 4up	Chapter 1	Gale-Shapley
Algorithm Analysis (big O notation)	1up · 4up	Chapter 2	—
Graphs (graph search)	1up · 4up	Chapter 3	—
Greedy Algorithms I (basic techniques)	1up · 4up	Chapter 4	interval scheduling · interval partitioning
Greedy Algorithms II (shortest paths and MSTs)	1up · 4up	Chapter 4	Dijkstra · red-blue · Prim · Kruskal · Borůvka · Edmonds
Divide and Conquer I (sorting and selection)	1up · 4up	Chapter 5	merging · inversions · quickselect
Divide and Conquer II (integer and polynomial multiplication)	1up · 4up	Chapter 5	—
Dynamic Programming I (basic techniques)	1up · 4up	Chapter 6	—
Dynamic Programming II (sequence alignment, Bellman-Ford)	1up · 4up	Chapter 6	—
Network Flow I (maximum flow theory)	1up · 4up	Chapter 7	Ford-Fulkerson · pathological
Network Flow II (maximum flow applications)	1up · 4up	Chapter 7	—
Network Flow III (assignment problem)	1up · 4up	Chapter 7	—
Intractability I (polynomial-time reductions)	1up · 4up	Chapter 8	—

Algorithm definitions

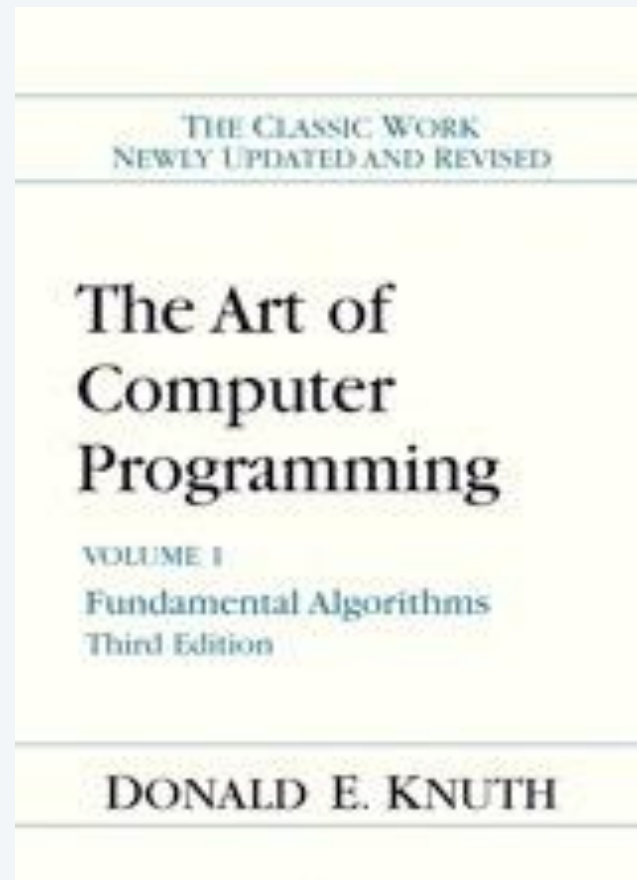
“ A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation. ” — [webster.com](https://www.merriam-webster.com/dictionary/algorithm)



“ An algorithm is a finite, definite, effective procedure, with some input and some output. ”

— *Donald Knuth (TAOCP)*

Donald Knuth



Everyday life is like programming, I guess. If you love something you can put beauty into it.

The most important thing in the programming language is the name. A language will not succeed without a good name. I have recently invented a very good name and now I am looking for a suitable language.



Algorithm Etymology

Etymology. [Knuth, TAOCP]

- **Algorithm:** A step-by-step procedure for calculation.
- **Algorism:** process of doing arithmetic using Hindu-Arabic numerals.
- **Word origin:** Abu 'Abd Allah Muhammad ibn Musa al-Khwarizm was a famous 9th century Persian textbook author who wrote *Kitāb al-jabr wa'l-muqābala*, which evolved into today's high school algebra text.
- His book “*On the Calculation with Hindu-Arabic Numerals*” in about 825 AD was translated into Latin as “*Algoritmi de numero Indorum*”
- **History:** Originally used for performing arithmetic using Hindu-Arabic numerals and solving linear and quadratic equations.



CS 3100 / CS 5100 vs. CS 7200

CS 3100/CS5100. Data Structures and Algorithms

- Stacks and queues.
- Sorting.
- Searching.
- Graph algorithms.
- String processing.

This is a fundamental course for students majoring in Computer Science. Students will learn: basic algorithm analysis techniques; asymptotic complexity; big- O and big- Ω notations; efficient algorithms for discrete structures including lists, trees, stacks, and graphs; fundamental computing algorithms including sorting, searching, and hashing techniques.

Emphasizes critical thinking, problem-solving, and **code**.

CS 7200. Algorithm Design and Analysis

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomized algorithms.
- Intractability.
- Coping with intractability.
- Data structures.

Emphasizes critical thinking, problem-solving, and **rigorous analysis**.

Algorithms: *What*, Why, How?

Example 1:

```
Mystery(int m, n) {  
     $m := n - m; n := n - m; m := m + n; \}$ 
```

Mystery(5,3):

$$m = 3 - 5 = -2$$

$$n = 3 - (-2) = 5$$

$$m = -2 + 5 = 3$$

Example?: Recipes and directions for making coffee, pasta, cakes, curries, etc.

Example?: Summarizing a story.

Example-Not: Determine whether the two given programs compute the same function?

Algorithms: What, *Why*, How?

- Normally focus on techniques for Efficient computation in terms of time and space.
- Ironically, even though Efficiency is “less important” than orthogonal features such as correctness, ease of use, cost, and security, in practice, efficiency (is like a currency that) can be traded with additional desirable features to maintain acceptable performance.

Algorithms: What, Why, *How*?

- Efficient Problem Solving Strategy

E.g., $2^8 = (2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2)$

E.g., Recursive doubling $2^8 = ((2^2)^2)^2$

- Detailed description of the solution steps

Algorithms: What, Why, *How*?

- **Design:** Naive exponential algorithms (e.g., generate and test approaches based on declarative specification) to efficient polynomial-time algorithms (using various techniques such as divide and conquer (recursion), and dynamic programming)
- **Analysis:** Time and space complexity

Algorithms: What, Why, *How*?

- **Proof of Correctness:** Using Mathematical Induction, Contradiction, Loop Invariants, etc.
- **Theory:** Complexity classes, Lower bounds, Reduction.
- **Practice:**
 - Overcome intractability via approximation;
 - Improve performance through randomization

Why study algorithms?

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, ...

Computers. Circuit layout, databases, caching, networking, compilers, ...

Computer graphics. Movies, video games, virtual reality, ...

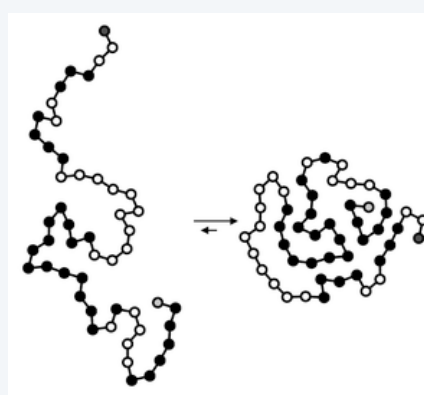
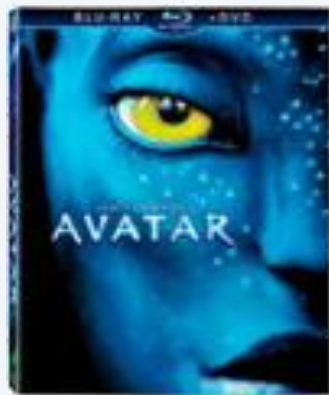
Security. Cell phones, e-commerce, voting machines, ...

Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...

Social networks. Recommendations, news feeds, advertisements, ...

Physics. N-body simulation, particle collision simulation, ...

⋮



We emphasize **algorithms** and **techniques** that are **useful in practice**.

Useful Sites

www.coursera.org

Stanford

Algorithms: Design and Analysis, Part 1

In this course you will learn several fundamental principles of algorithm design: divide-and-conquer methods, graph algorithms, practical data structures (heaps, hash tables, search trees), randomized algorithms, and more.

[Preview Lectures](#)



About the Course

In this course you will learn several fundamental principles of algorithm design. You'll learn the divide-and-conquer design paradigm, with applications to fast sorting, searching, and multiplication. You'll learn several blazingly fast primitives for computing on graphs, such as how to compute connectivity information and shortest paths. Finally, we'll study how allowing the computer to "flip coins" can lead to elegant and practical algorithms and data structures. Learn the answers to questions such as: How do data structures like heaps, hash tables, bloom filters, and balanced search trees actually work, anyway? How come QuickSort runs so fast? What can graph algorithms tell us about the structure of the Web and social networks? Did my 3rd-grade teacher explain only a suboptimal algorithm for multiplying two numbers?

Stanford

Algorithms: Design and Analysis, Part 2

In this course you will learn several fundamental principles of advanced algorithm design: greedy algorithms and applications; dynamic programming and applications; NP-completeness and what it means for the algorithm designer; the design and analysis of heuristics; and more.

[Preview Lectures](#)



About the Course

In this course you will learn several fundamental principles of advanced algorithm design. You'll learn the greedy algorithm design paradigm, with applications to computing good network backbones (i.e., spanning trees) and good codes for data compression. You'll learn the tricky yet widely applicable dynamic programming algorithm design paradigm, with applications to routing in the Internet and sequencing genome fragments. You'll learn what NP-completeness and the famous "P vs. NP" problem mean for the algorithm designer. Finally, we'll study several strategies for dealing with hard (i.e., NP-complete problems), including the design and analysis of heuristics. Learn how shortest-path algorithms from the 1950s (i.e., pre-ARPANET!) govern the way that your Internet traffic gets routed today; why efficient algorithms are fundamental to modern genomics; and how to make a million bucks in prize money by "just" solving a math problem!

Sessions

March 16, 2015 - May 16, 2015

[Join for Free!](#)

Course at a Glance

- 6 weeks of study
- 6-10 hours/week
- English

Useful Sites

ocw.mit.edu

MITOPENCOURSEWARE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Subscribe to the OCW News

Home

Courses

About

Donate

Featured Sites

Search

Home » Courses » Electrical Engineering and Computer Science » Introduction to Algorithms

Introduction to Algorithms

COURSE HOME

SYLLABUS

CALENDAR

READINGS

LECTURE VIDEOS


RECITATION VIDEOS

ASSIGNMENTS

EXAMS

RELATED RESOURCES

DOWNLOAD COURSE MATERIALS



In Problem Set 6, students develop algorithms for solving the 2x2x2 Rubik's Cube.

Like 726

Tweet 131

+1 109

Share

Submit

submit

Course Description

Other Versions

Related Content

Course Features

> [Video lectures](#)

> [Assignments and solutions](#)

> [Recitation videos](#)

> [Subtitles/transcript](#)

> [Exams and solutions](#)

Course Description

This course provides an introduction to mathematical modeling of computational problems. It covers the common algorithms, algorithmic paradigms, and data structures used to solve these problems. The course emphasizes the relationship between algorithms and programming, and introduces basic performance measures and analysis techniques for these problems.

Questions?

