

Algorithms for the masses

Robert Sedgewick
Princeton University



This talk is dedicated to the memory of Philippe Flajolet



Philippe Flajolet 1948–2011

Prelude: my first job

1969 Western Electric Engineering Research Center

PDP-9

- 18-bit words
- 16K memory
- switches+lights
- paper tape reader/punch
- display w/ lightpen

Task: Drafting (Illustration)

Workflow

- write cross-assembler in IBM 360/50 assembly
- write Illustrator application in PDP-9 assembly
- load paper tape into PDP-9
- run application

PDP-9



Prelude: my first job

Problem: Cross-assembler too slow

Solution: Binary search!

- M searches, N symbols $M \gg N$
- Improved running time from $\sim MN$ to $\sim M \lg N$

IBM 360/50



Lesson 1: Good algorithms matter

Lesson 2: Not many programmers appreciate that fact

Brief history of Algorithms



Algorithms (central thesis)

1975: What are the algorithms that *everyone* should know?

“Everyone” means “everyone”

- scientists
- engineers
- mathematicians
- software/hardware designers
- cryptanalysts
- COBOL programmers
- ...

Context

- IBM 360/50
- Algol W
- **one** run per day



Algorithms (brief history)

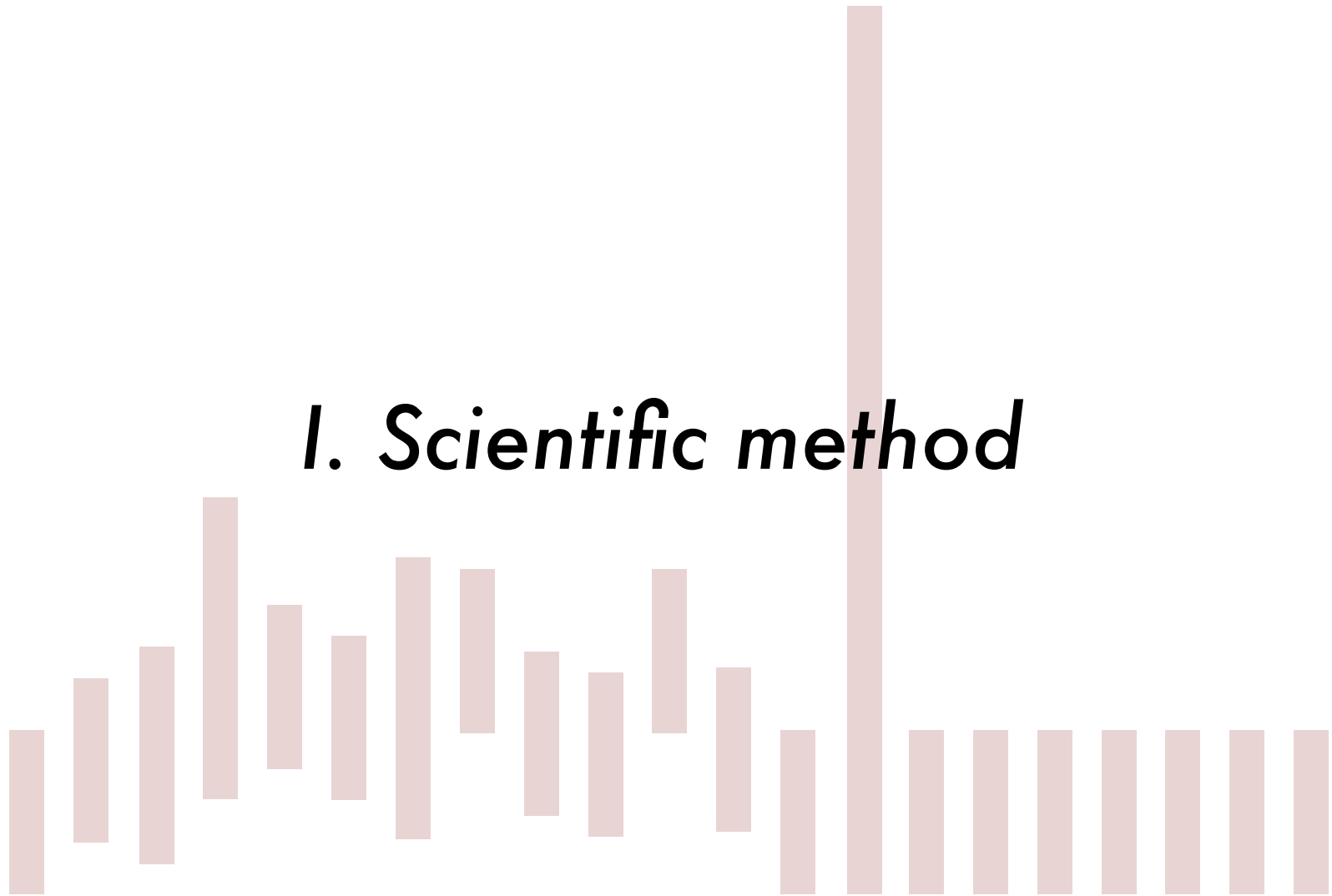
		edition	code validation
1982		1st	<i>compiles</i>
1986		2nd	<i>runs</i>
1994		3rd	<i>performance comparisons within reasonable practical model</i>
2011	[this talk]	4th	

Four challenges

- I. Many algorithms implemented/tested in back rooms, not open literature
- II. Need appropriate mathematical models
- III. Masses can't program, don't know CS
- IV. How to disseminate?



I. Scientific method



Fact of life in applied computing

Performance matters in a large number of important applications

Example: quadratic algorithms are useless in modern applications

- millions or billions of inputs
- 10^{12} nanoseconds is 15+ minutes
- 10^{18} nanoseconds is 31+ years

- *indexing and search*
- *Bose-Einstein model*
- *N-body*
- *signal processing*
- *string matching for genomics*
- *natural language analysis*
- *[very long list]*

Important lessons of the past several decades

1. Efficient algorithms enable solution of problems that could not otherwise be addressed.
2. Scientific method is essential in understanding program performance

Important lessons for

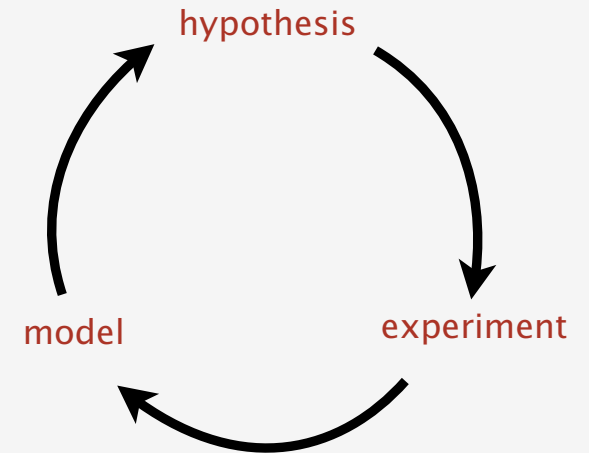
- *beginners*
- *engineers*
- *scientists*
- *programmers*

The scientific method

is essential in understanding program performance

Scientific method

- create a model describing natural world
- use model to develop hypotheses
- run experiments to validate hypotheses
- refine model and repeat



1950s: Uses scientific method.



2010s: Uses scientific method?

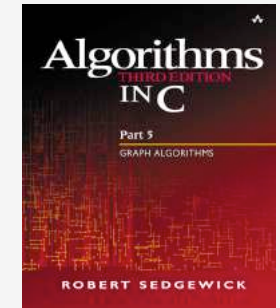
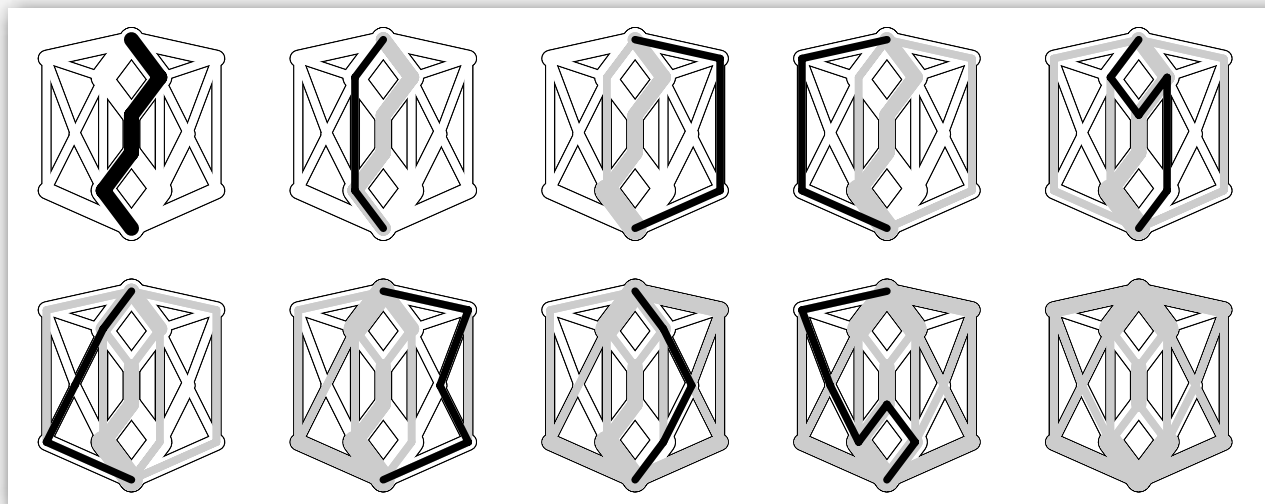


Algorithm designer who does not experiment gets lost in abstraction
Software developer who ignores cost risks catastrophic consequences
Scientist/engineer needs to control costs of experiments/simulations

Motivating example: maxflow

Ford-Fulkerson maxflow scheme

- find any s-t path in a (residual) graph
- augment flow along path (may create or delete edges)
- iterate until no path exists



Goal: compare performance of two basic implementations

- **shortest** augmenting path
- **maximum capacity** augmenting path

Key steps in analysis

- How many augmenting paths?
- What is the cost of finding each path?

research literature

this talk

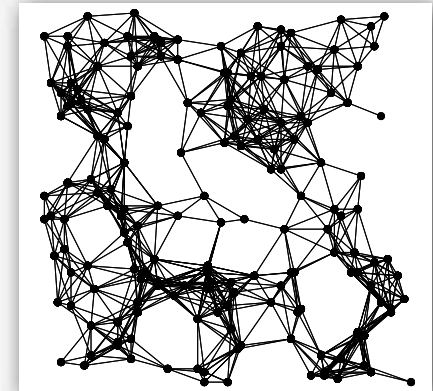
Motivating example: max flow

Compare performance of Ford-Fulkerson implementations

- shortest augmenting path
- maximum-capacity augmenting path

Graph parameters for a reasonable model

V E C
vertices *edges* *max capacity*



How many augmenting paths?

	<i>upper bound</i>
<i>shortest</i>	$\frac{VE}{2C}$
<i>max capacity</i>	$2E \lg C$

How many steps to find each path?

E (upper bound)

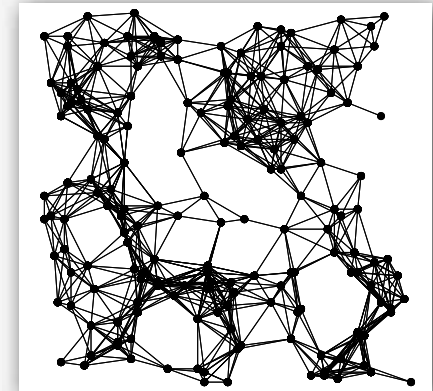
Motivating example: max flow

Compare performance of Ford-Fulkerson implementations

- shortest augmenting path
- maximum-capacity augmenting path

Graph parameters for a reasonable model

$V = 177$ $E = 2000$ $C = 100$
vertices *edges* *max capacity*



How many augmenting paths?

	<i>upper bound</i>	<i>for example</i>
<i>shortest</i>	$VE/2$	177,000
	VC	17,700
<i>max capacity</i>	$2E \lg C$	26,575

How many steps to find each path?

2000 (upper bound)

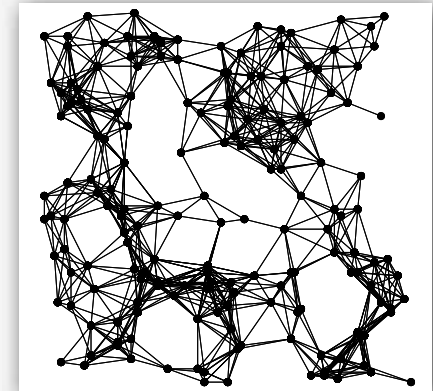
Motivating example: max flow

Compare performance of Ford-Fulkerson implementations

- shortest augmenting path
- maximum-capacity augmenting path

Graph parameters for a reasonable model

$V = 177$ $E = 2000$ $C = 100$
vertices *edges* *max capacity*



How many augmenting paths?

	<i>upper bound</i>	<i>for example</i>	<i>actual</i>
<i>shortest</i>	$VE/2$	177,000	37
	VC	17,700	
<i>max capacity</i>	$2E \lg C$	26,575	7

How many steps to find each path? < 20, on average, for randomized search

Prediction of total cost is a factor of **1 million** high for thousand-node graphs

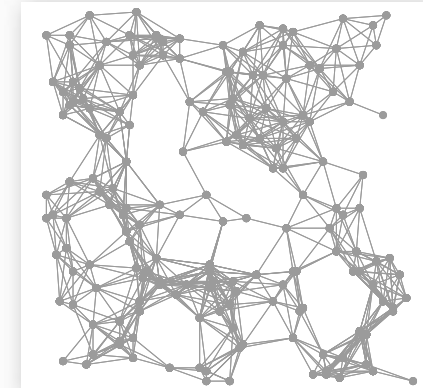
Motivating example: max flow

Compare performance of Ford-Fulkerson implementations

- shortest augmenting path
- maximum-capacity augmenting path

Graph parameters for a reasonable model


V vertices E edges C max capacity



How many augmenting paths?

shortest $VE/2$
max capacity VC
 $2E \lg C$

How many steps to find each path? E (upper bound)

 **Warning:**
Such analyses are **useless** for predicting performance or comparing algorithms

Motivating example: lessons

Goals of algorithm analysis

- predict performance (running time) or
- guarantee that cost is below specified bounds

Common wisdom

- random graph models are unrealistic
- average-case analysis of algorithms is too difficult
- worst-case performance bounds are the standard

Unfortunate truth about worst-case bounds

- often useless for prediction (fictional)
- often useless for guarantee (too high)
- often misused to compare algorithms

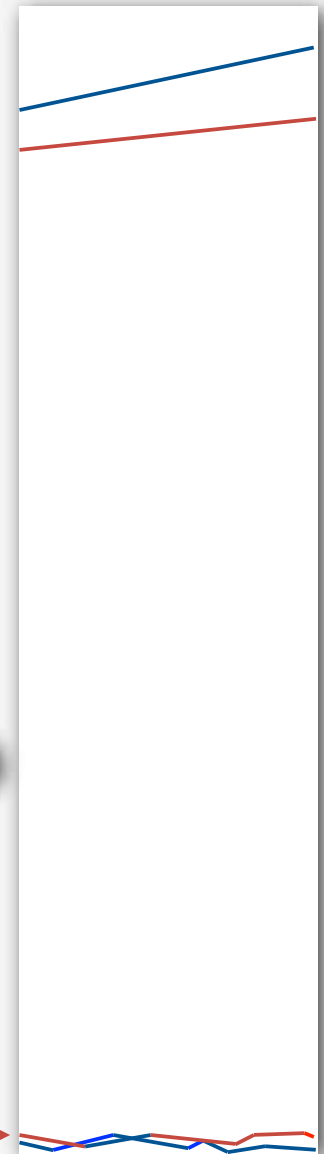
Bounds are useful in some applications.

Open problem: Do better!

worst-case bounds

which ones??

actual costs



O-notation considered harmful

How to predict performance (and to compare algorithms)?

Not the scientific method: O -notation

Theorem: Running time is $O(N^c)$ ❌

- not at all useful for predicting performance

Scientific method calls for tilde-notation.

Hypothesis: Running time is $\sim aN^c$ ✓

- an effective path to predicting performance (stay tuned)

O -notation is useful for many reasons, BUT

Common error: Thinking that O -notation **is** useful for predicting performance.

Surely, we can do better

A typical exchange in Q&A

RS (in a talk): O-notation considered harmful.
Cannot use it to predict performance.

Q: ?? $O(N \log N)$ surely beats $O(N^2)$

RS: Not by the definition. O expresses upper bound.

Q: So, use Theta.

RS: Still (typically) bounding the worst case.
Is the input a worst case?

Q: (whispers to colleague) I'd use the $\Theta(N \log N)$ algorithm, wouldn't you?

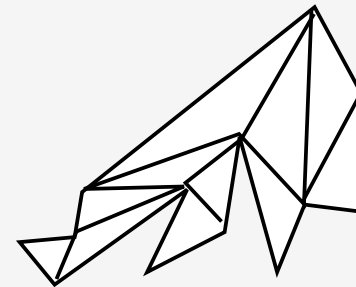
Galactic algorithms

R.J. Lipton: A **galactic algorithm** is one that will never be used in practice

Why? Any effect would never be noticed in this galaxy

Ex. Chazelle's linear-time triangulation algorithm

- theoretical tour-de-force
- too complicated to implement
- cost of implementing would exceed savings in this galaxy, anyway



One blogger's conservative estimate: 75% SODA, 95% STOC/FOCS are galactic

OK for basic research to drive agenda, BUT

Common error: Thinking that a galactic algorithm **is** useful in practice.

Surely, we can do better

An actual exchange with a theoretical computer scientist:

TCS (in a talk):

Algorithm A is bad.
Google should be interested in my new Algorithm B.

RS:

What's the matter with Algorithm A?

TCS:

It is not optimal. It has an extra $O(\log \log N)$ factor.

RS:

But Algorithm B is very complicated, $\lg \lg N$ is less than 6 in this universe, and that is just an upper bound. Algorithm A is certainly going to run 10 to 100 times faster in any conceivable real-world situation. Why should Google care about Algorithm B?

TCS:

Well, I like Algorithm B. I don't care about Google.

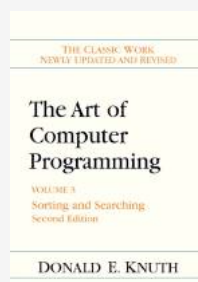
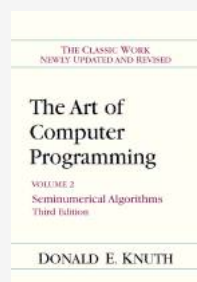
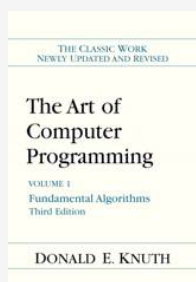
II. Analytic Combinatorics



Analysis of algorithms and analytic combinatorics

Appropriate mathematical models are essential for scientific studies of program behavior

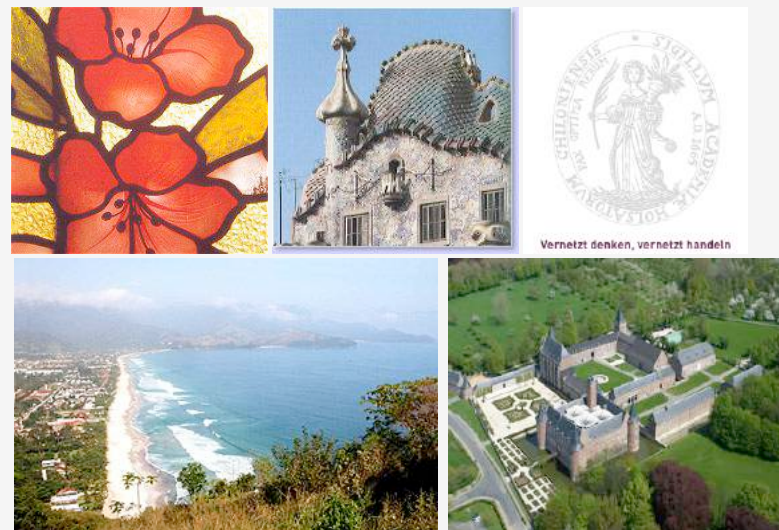
Pioneering work by Don Knuth



Active AofA community is building on classical research in

- probability
- combinatorics
- analysis
- information theory

and is developing new models, methods, and applications



AofA09
20th International Conference on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms
Fréjus, France – June 14-19 2009

AofA'10
21st International Meeting on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms
Vienna, Austria
June 28 - July 2, 2010

Analytic Combinatorics

is a modern basis for studying discrete structures

Developed by

Philippe Flajolet and many coauthors (including RS)

based on

classical combinatorics and analysis

Generating functions (GFs) encapsulate sequences

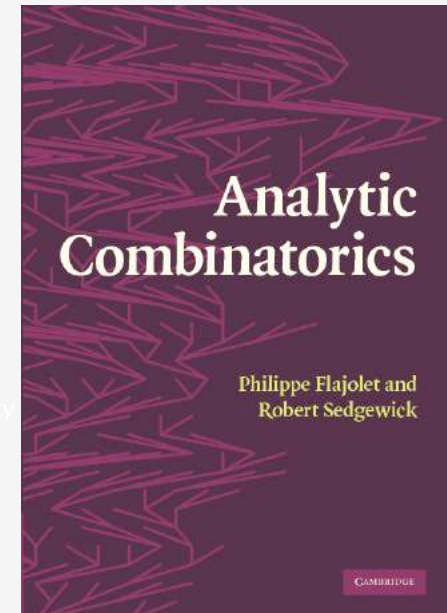
Symbolic methods treat GFs as formal objects

- formal definition of combinatorial constructions
- direct association with generating functions

Complex asymptotics treat GFs as functions in the complex plane

- Study them with singularity analysis and other techniques
- Accurately approximate original sequence

Cambridge University



↑
Cambridge 2009
also available
on the web

Complexification

Assigning complex values to the variable z in a GF gives a **method of analysis** to estimate the coefficients.

The **singularities** of the function determine the method.

<i>singularity type</i>	<i>method of analysis</i>
meromorphic (just poles)	Cauchy (elementary)
fractional powers logarithmic	Cauchy (Flajolet-Odlyzko)
none (entire function)	saddle point

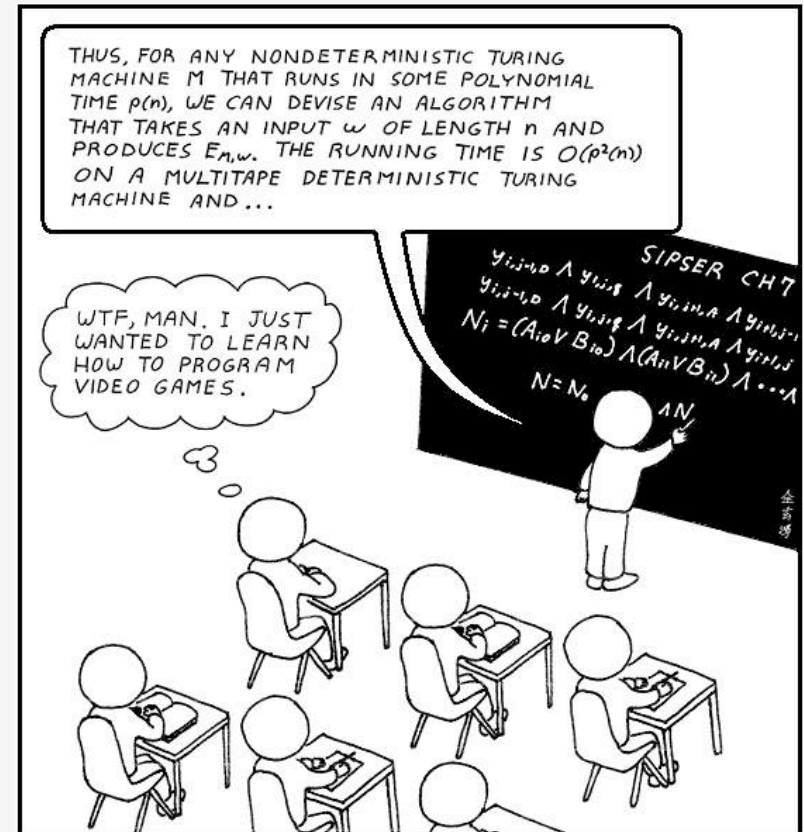
First Principle. Exponential growth of a function's coefficients is determined by the **location** of its singularities.

Second Principle. Subexponential factor in a function's coefficients is determined by the **nature** of its singularities.

Analytic combinatorics

Q. Wait, didn't you say that the masses don't need to know all that math?

RS. Well, there is **one** thing...



A general hypothesis from analytic combinatorics

The running time of **your program** is $\sim a b^N N^c (\lg N)^d$

- the constant a depends on both complex functions and properties of machine and implementation
- the exponential growth factor b should be 1
- the exponent c depends on singularities
- the log factor d is reconciled in detailed studies

Why?

- data structures evolve from combinatorial constructions
- universal laws from analytic combinatorics have this form

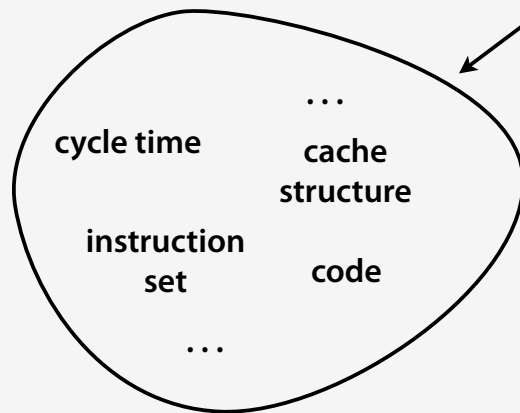
Plenty of caveats, but provides, in **conjunction with the scientific method**, a basis for studying program performance

Computing the constants (the hard way)

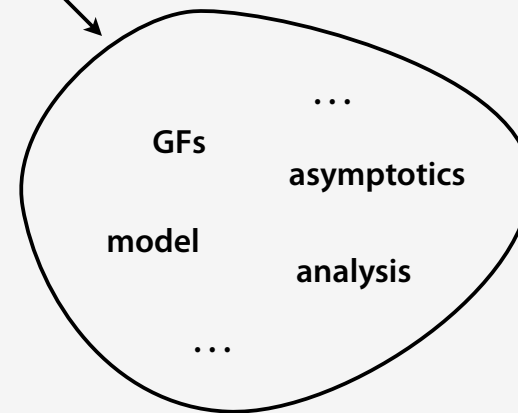
Knuth showed that it is possible *in principle* to precisely predict running time

- develop a mathematical model for the frequency of execution of each instruction in the program
- determine the time required to execute each instruction
- multiply and sum

$$\text{Hypothesis: } T(N) \sim aN^c$$



*engineer's part of the constant
(harder to determine now than in the 1970s)*



*mathematician's part of the constant
(easier to determine now than in the 1970s)*

Computing the constants (the easy way)

Run the program!

Hypothesis: $T(N) \sim aN^c$

Note: log factors are more difficult

1. Implement the program
2. Compute $T(N_0)$ and $T(2N_0)$ by running it
3. Calculate c as follows:

$$\begin{aligned}\frac{T(2N_0)}{T(N_0)} &\sim \frac{a(2N_0)^c}{aN_0^c} \\ &= 2^c\end{aligned}$$

$\lg(T(2N_0)/T(N_0)) \rightarrow c$ as N_0 grows

4. Calculate a as follows:

$T(N_0)/N_0^c \rightarrow a$ as N_0 grows

Predicting performance (the easy way)

Don't bother computing the constants!

Hypothesis: $T(N) \sim aN^c$

1. Implement the program
2. Run it for $N_0, 2N_0, 4N_0, 8N_0, \dots$ if log factors exist, estimate improves as N grows
3. Note that ratio of running times approaches 2^c

$$\frac{T(2N_0)}{T(N_0)} \sim \frac{a(2N_0)^c}{aN_0^c} = 2^c$$

4. Multiply by 2^c to predict next value

1000	1.1 sec
2000	4.5 sec
4000	18 sec
8000	73 sec
16000	295 sec

Plenty of caveats, but provides a basis for teaching the masses about program performance

← borders on malpractice *not* to do so!

III. Introduction to CS



The masses

Scientists, engineers and modern programmers need

- extensive specialized knowledge in their field
- an understanding of the scientific method.

They also need to know how to

- write programs
- design and analyze algorithms

Do they need to know?

- Detailed analysis
- Galactic algorithms
- Overly simple input models



They **do** need to know

- Classic algorithms
- Realistic input models and randomization
- How to predict performance and compare algorithms

Unfortunate facts

Many **scientists/engineers** lack basic knowledge of **computer science**

Many **computer scientists** lack back knowledge of **science/engineering**

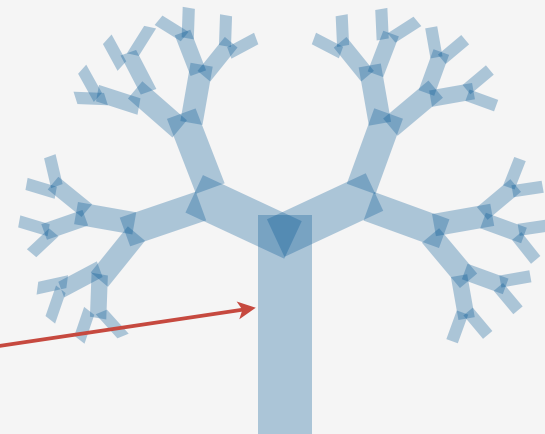
1970s: Want to use the computer? Take intro CS.

1990s: Intro CS course relevant only
to future cubicle-dwellers



One way to address the situation

- identify fundamentals
- teach them to all students who need to know them
- **as early as possible**



Intro course model: typical

CS
for
CS majors

CS
for
physicists

CS
for
math majors

CS
for
biochemists

CS
for
EE

CS
for
poets

CS
for
rocket
scientists

CS
for
economists

CS
for
civil engineers

CS
for
idiots

Intro course model: RS view

CS
for
everyone



Original motivation (1992)

Why not?

Works for biology, math, physics, economics.

Responsibility to identify and teach fundamental tenets of discipline.

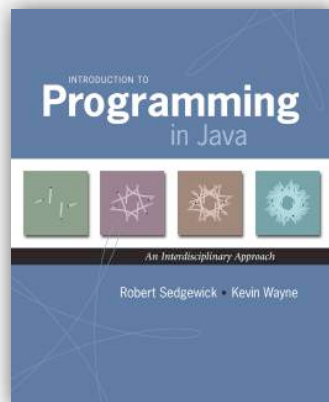
Current status (2012)

[Joint work with Kevin Wayne since 2000]

Anyone can learn the importance of

- modern programming models
- **the scientific method in understanding program behavior**
- fundamental precepts of computer science
- computation in a broad variety of applications
- preparing for a lifetime of engaging with computation

Textbook and booksite available and widely used [stay tuned]



intros.cs.princeton.edu

INTRODUCTION TO PROGRAMMING IN JAVA
a textbook for a first course in computer science for the next generation of scientists and engineers

Welcome to our website!

Textbook: Our textbook *Introduction to Programming in Java: An Interdisciplinary Approach* (Amazon - [Intros](#)) is an interdisciplinary approach to the traditional CS1 curriculum. We teach all of the classic elements of programming, using an "object-in-the-middle" approach that emphasizes data abstraction. The book is organized around four stages of learning to program:

- **Chapter 1: Elements of Programming** introduces variables; assignment statements; built-in types of data; conditionals and loops; arrays; and input/output, including graphics and sound.
- **Chapter 2: Functionals** introduces modular programming. We stress the fundamental idea of dividing a program into components that can be independently debugged, maintained, and reused.
- **Chapter 3: Object-Oriented Programming** introduces data abstraction. We emphasize the concept of a data type and its implementation using Java's class mechanism.
- **Chapter 4: Algorithms and Data Structures** introduces classical algorithms for sorting and searching, and fundamental data structures, including stacks, queues, and symbol tables.

A key feature of the book is the manner in which we motivate each programming concept that we address by examining its impact on specific applications, taken from fields ranging from materials science to genomics to astrophysics to Internet commerce. This approach highlights the essential idea that mathematics, science, engineering, and computing are intertwined in the modern world.

To preview our material, you can download the [preface](#) and [Chapter 1](#).

Booksite: Reading a book and surfing the web are two different activities. This booksite is intended for your use while online (for example, while programming and while browsing the web); the textbook is for your use when visiting learning new material and when reviewing your understanding of that material (for example, when reviewing to prepare for an exam). The booksite consists of the following elements:

- **Excerpts:** A condensed version of the text narrative for reference while online.
- **Exercises:** Hundreds of exercises, including many from the text and some solutions.
- **Java code:** Hundreds of easily downloadable complete Java programs, along with text data.
- **References:** Extensive links throughout to related material on the web.

To adopt: Here are some of the distinctive features of our textbook and a [marketing flyer](#). If you wish to consider adoption, please fill out this form to request a copy of the textbook or ask for more information.

Copyright © 2007 Robert Sedgwick and Kevin Wayne. All rights reserved.

Messages for first-year students

Reading, writing, and **computing**

Programming is for everyone, including **you**

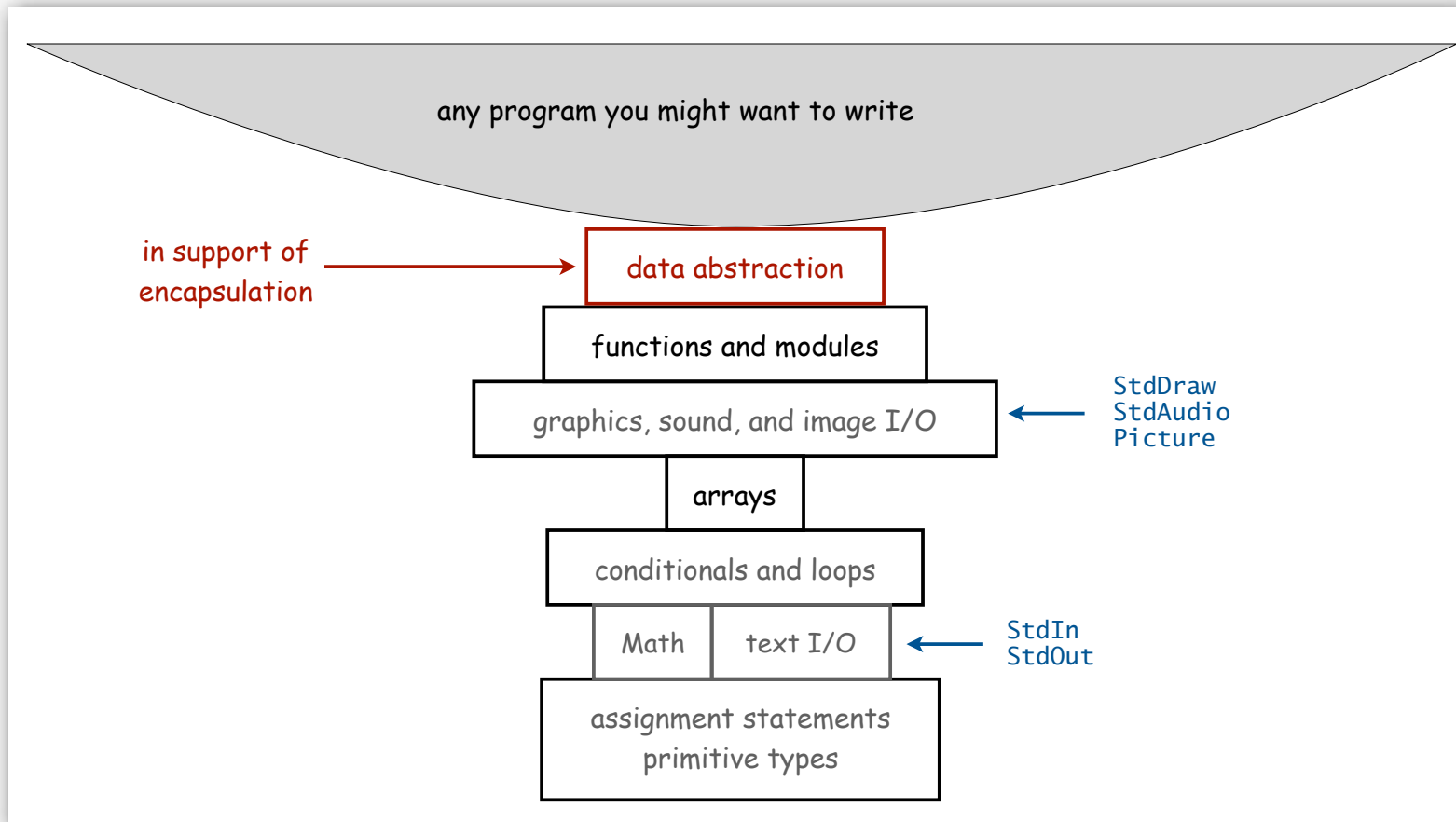
- it is easier than most challenges you're facing
- you cannot be successful in any field without it

Performance matters

There is more to computer science than programming

Computer science is intellectually challenging, worth knowing

Key ingredient: a modern programming model



Basic requirements

- full support of essential components
- freely available, widely used

1990: C/C++, 2010: Java, 2020: ??

CS in scientific context

Ideal programming example/assignment

- teaches a basic CS concept
- solves an important problem
- intellectually engaging and appealing
- illustrates modular programming
- is open-ended

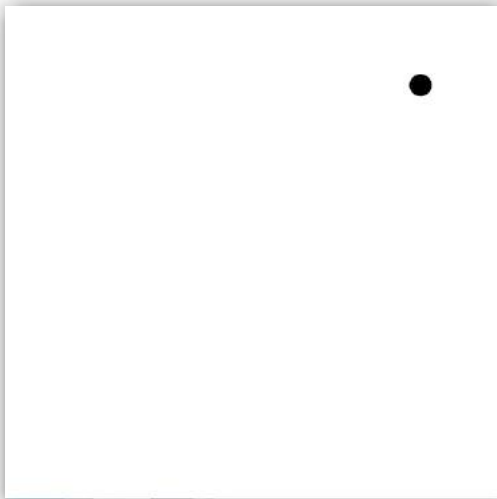
functions	<i>sqrt(), log()</i>
libraries	<i>I/O, data analysis</i>
1D arrays	<i>sound</i>
2D arrays	<i>images</i>
recursion	<i>fractal models</i>
strings	<i>genomes</i>
I/O streams	<i>web resources</i>
OOP	<i>Brownian motion</i>
data structures	<i>small-world</i>

Familiar and easy-to-motivate applications

Ideal programming example/assignment

- teaches a basic CS concept
- solves an important problem
- intellectually engaging and appealing
- illustrates modular programming
- is open-ended

Bouncing ball



Simulation is easy

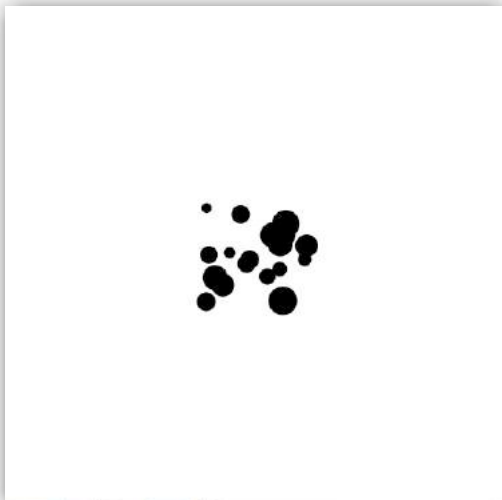
```
public class BouncingBall
{
    public static void main(String[] args)
    { // Simulate the movement of a bouncing ball.
        StdDraw.setXscale(-1.0, 1.0);
        StdDraw.setYscale(-1.0, 1.0);
        double rx = .480, ry = .860;
        double vx = .015, vy = .023;
        double radius = .05;
        int dt = 20;
        while(true)
        { // Update ball position and draw it there.
            if (Math.abs(rx + vx) + radius > 1.0) vx = -vx;
            if (Math.abs(ry + vy) + radius > 1.0) vy = -vy;
            rx = rx + vx;
            ry = ry + vy;
            StdDraw.clear();
            StdDraw.filledCircle(rx, ry, radius);
            StdDraw.show(dt);
        }
    }
}
```

Familiar and easy-to-motivate applications

Ideal programming example/assignment

- teaches a basic CS concept
- solves an important problem
- appeals to students' intellectual interest
- illustrates modular programming
- is open-ended

Bouncing balls



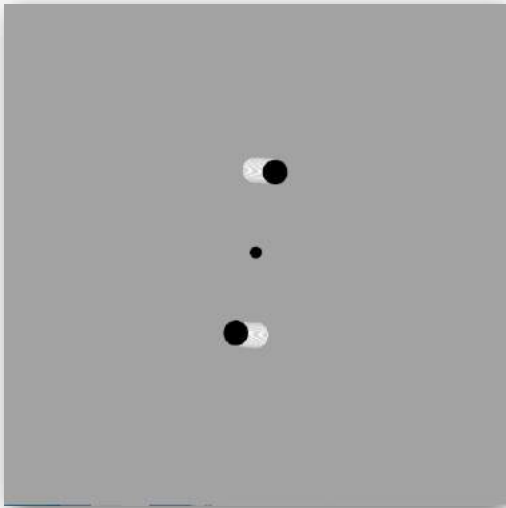
OOP is helpful

Familiar and easy-to-motivate applications

Ideal programming example/assignment

- teaches a basic CS concept
- solves an important problem
- appeals to students' intellectual interest
- illustrates modular programming
- is open-ended

N-body



data-driven programs are useful

Distinctive features of our approach

address some traditional barriers

No room in curriculum?

- appeal to familiar concepts from HS science and math saves room
- broad coverage provides real choice for students choosing major
- modular organization gives flexibility to adapt to legacy courses
- detailed examples useful throughout curriculum

Incorrect perceptions about CS?

- scientific basis gives students the big picture
- students are enthusiastic about addressing real applications

Excessive focus on programming?

- careful introduction of essential constructs
- nonessential constructs left for later CS courses
- library programming restricted to key abstractns
- taught in context with plenty of other material

Distinctive features of our approach

address some traditional barriers

One course fits all?

- few students get adequate CS in high school nowadays
- 90+ percent on level playing field by midterms
- open-ended assignments appeal even to experienced programmers
- not harmful for CS students to learn scientific context before diving into abstraction

CS is for cubicle-dwellers?

- “learned more in this course than in any other”
- “came here to study physics/math/bio/econ, now I want to do CS”
- “cool”

Progress report (2011)

Stable intro CS course for all students

modern programming model

- Basic control structures
- Standard input and output streams
- Drawings, images and sound
- Data abstraction
- Use any computer, and the web

relevant CS concepts

- Understanding of the costs
- Fundamental data types
- Computer architecture
- Computability and Intractability

Goals

- demystify computer systems
- empower students to exploit computation
- build awareness of intellectual underpinnings of CS



Course Catalog

All Courses | Undergraduate Courses | Graduate Courses

This is the list of courses that the department may offer. The [Course Information](#) page lists the currently scheduled courses.
Note: Undergrad course numbers go to 499, grad courses are 500 and above. Not all students can take all courses.

COS109 - Computers in Our World (Fall)

Computers are all around us. How does this affect the world we live in? This course is a broad introduction to computing technology for humanities and social science students. Topics will be drawn from current issues and events, and will include discussion of how computers work, what programming is and why it is hard, how the Internet and the Web work, security and privacy. Two 90-minute lectures, one three-hour laboratory.

COS116 - The Computational Universe (Spring)

Computers have brought the world to our fingertips. We will try to understand, at a high level, the science -old and new- underlying this new Computational Universe. Our quest takes us on a broad sweep of scientific knowledge (and technologies they enable): propositional logic of the ancient Greeks (microprocessors); quantum mechanics (silicon chips); network and system phenomena (Internet); and search engines; computational intractability (secure encryption); efficient algorithms (genomic sequencing). Ultimately, this study makes us look anew at ourselves-our genome; language; music; "knowledge"; and above all, the mystery of our intelligence.

COS126 - General Computer Science (Fall, Spring)

An introduction to computer science in the context of scientific, engineering, and commercial applications. The goal of the course is to teach basic principles and practical issues, while at the same time preparing students to use computers effectively for applications in computer science, physics, biology, chemistry, engineering, and other disciplines. Topics include: hardware and software systems; programming in Java; algorithms and data structures; fundamental principles of computation; and scientific computing, including simulation, optimization, and data analysis. No prior programming experience required. Two lectures, two classes.

COS217 - Introduction to Programming Systems (Fall, Spring)

An introduction to computer organization and system software. The former includes topics such as processor and memory organization, input/output devices, and interrupt structures. The latter includes assemblers, loaders, libraries, and compilers. Programming assignments are implemented in assembly language and C using the UNIX operating system. Three lectures. **Prerequisite(s):** 126 or instructor's permission.

COS226 - Algorithms and Data Structures (Fall, Spring)

The study of fundamental data structures such as lists, queues, stacks, trees, heaps, hash tables, and their variations. The implementation and analysis of important algorithms for sorting, searching, string processing, geometric applications, and graph manipulation. Introduction to advanced algorithms and techniques. Two lectures, one preceptorial. **Prerequisite(s):** 126 or instructor's permission.

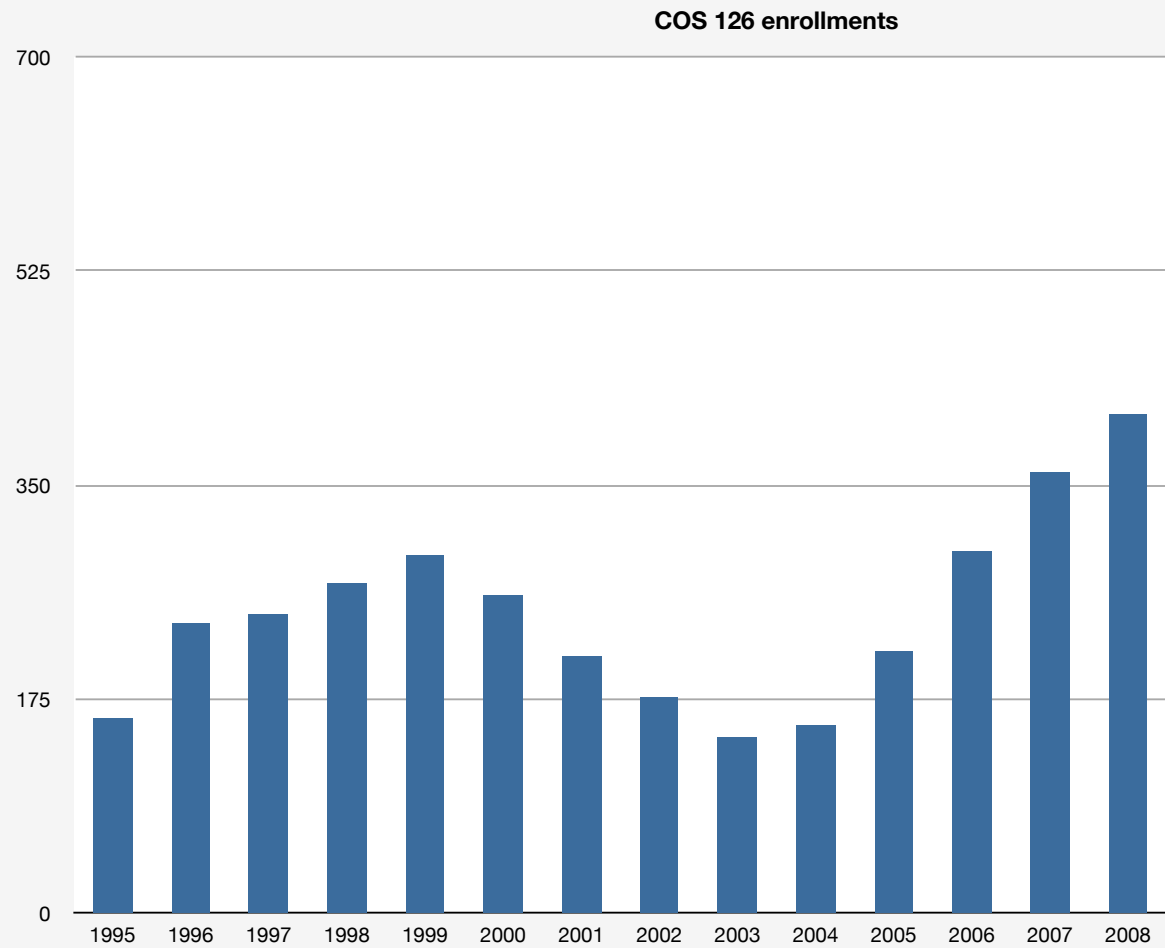
COS231/COS232 - An Integrated, Quantitative Introduction to the Natural Sciences

scientific content

- Scientific method
- Data analysis
- Simulation
- Applications

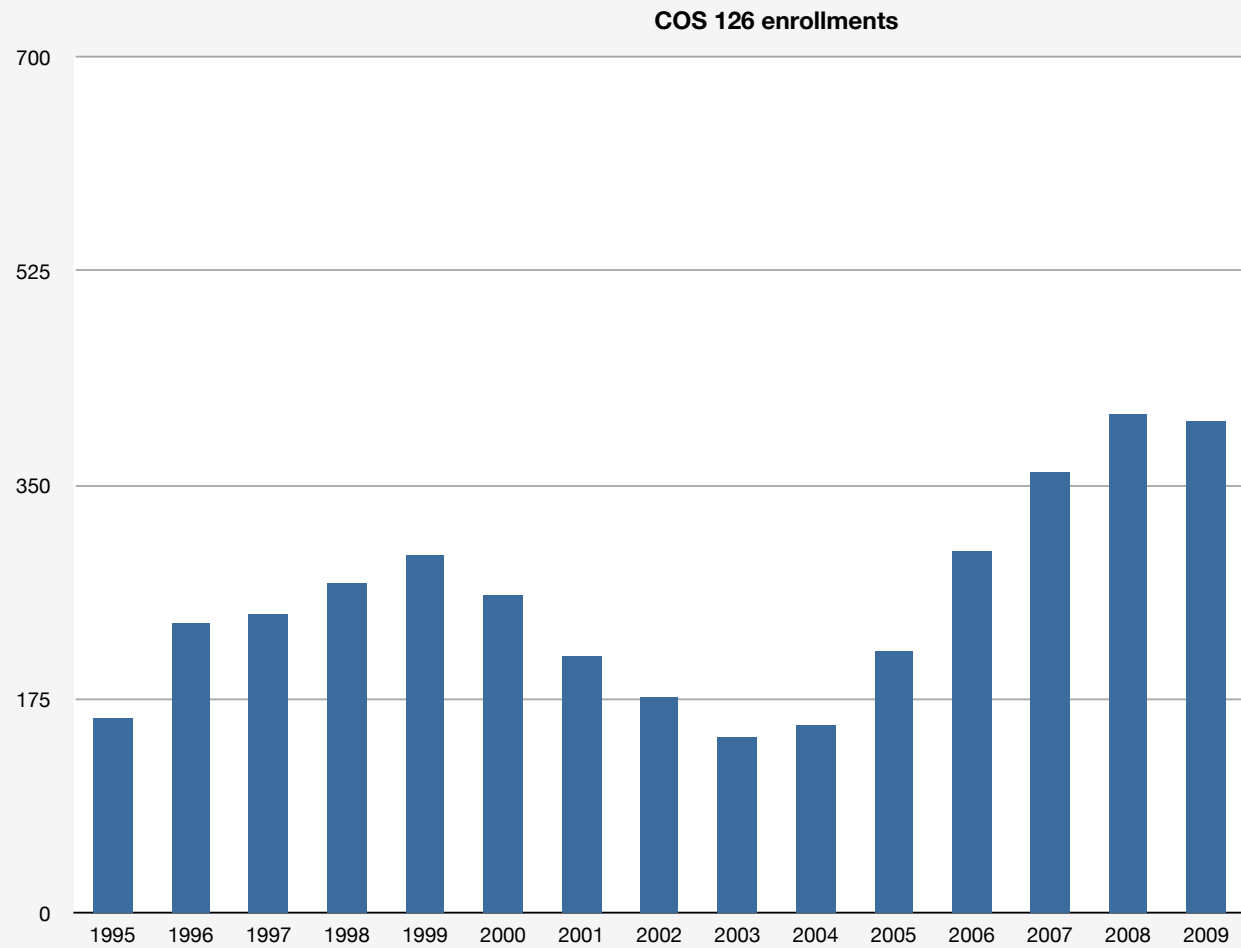
Progress report

2008: Enrollments are up. Is this another “bubble”?



Progress report

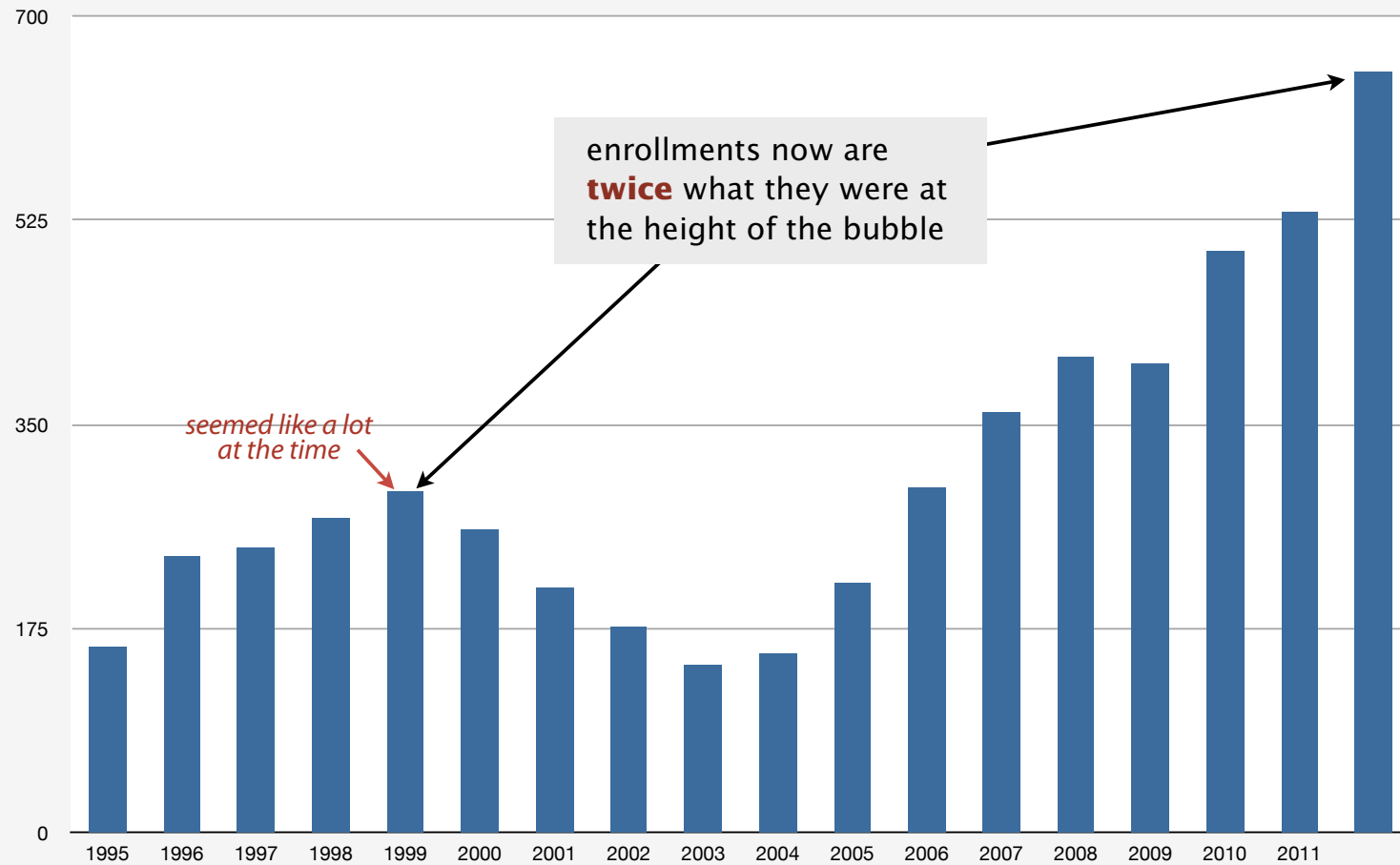
2009: Maybe.



Progress report

2012: Enrollments are skyrocketing.

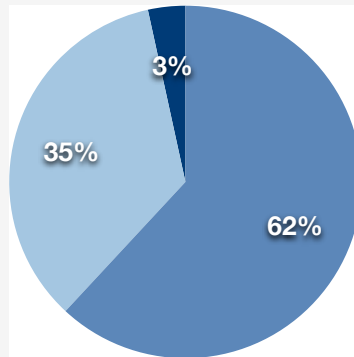
COS 126 enrollments



Who are they?

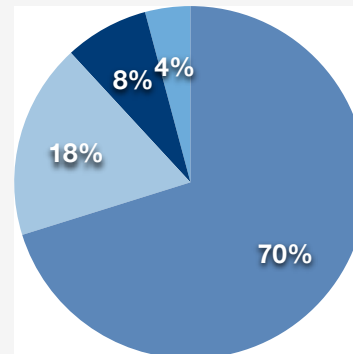
Over half of all Princeton students.

PROGRAMMING EXPERIENCE



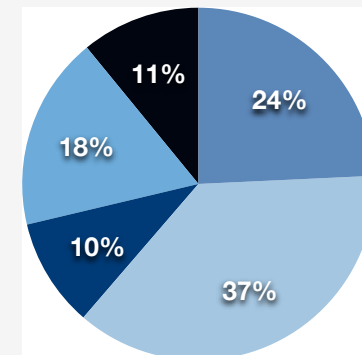
- none
- some
- lots

CLASS



- First-year
- Sophomore
- Junior
- Senior

INTENDED MAJOR



- other Science/Math
- other Engineering
- Humanities
- Social sciences
- CS

IV. Future of publishing



Seismic changes are afoot

Books?

Libraries?

Textbooks?

Why try to write a new textbook in this environment?

Grafton: Save the libraries

FUTURE READING

Digitization and its discontents.

by Anthony Grafton

The New Yorker

November 5, 2007

*...Sit in your local coffee shop, and your laptop can tell you a lot. If you want deeper, more local knowledge, you will have to take the narrower path that leads between the lions and up the stairs. There—as in great libraries around the world—you'll use all the new sources, the library's and those it buys from others, all the time. You'll check musicians' names and dates at Grove Music Online, read Marlowe's "Doctor Faustus" on Early English Books Online, or decipher Civil War documents on Valley of the Shadow. But these streams of data, rich as they are, will illuminate, rather than eliminate, books and prints and manuscripts that only the library can put in front of you. **The narrow path still leads, as it must, to crowded public rooms where the sunlight gleams on varnished tables, and knowledge is embodied in millions of dusty, crumbling, smelly, irreplaceable documents and books.***

RS: Think about the future

The New Yorker
Letter to the editor

*While Grafton's reservations about putting knowledge online are well taken, I would also point out that there is quite a bit going on now in the academic world that doesn't have much to do with old books. Indeed, as the author of many books, I wonder whether perhaps the book is not quite sacred as a means of disseminating knowledge. **What is the most effective way to produce and disseminate knowledge with today's technology? How can we best structure what we know and learn so that students, researchers, and scholars of the future can best understand the work of today's researchers and scholars?** I think that questions like these are more important and more difficult to address than whether we can put the contents of libraries on the Web.*

Robert Sedgewick
December 10, 2007

Future of libraries?

1990 Every student spent significant time in the library

2010 Every student spends significant time online

Few faculty members in the sciences use the library **at all** for research

YET, the library's budget continues to grow!

2020?

- A few book museums (for Grafton)
- Digital library infrastructure (for everyone else)

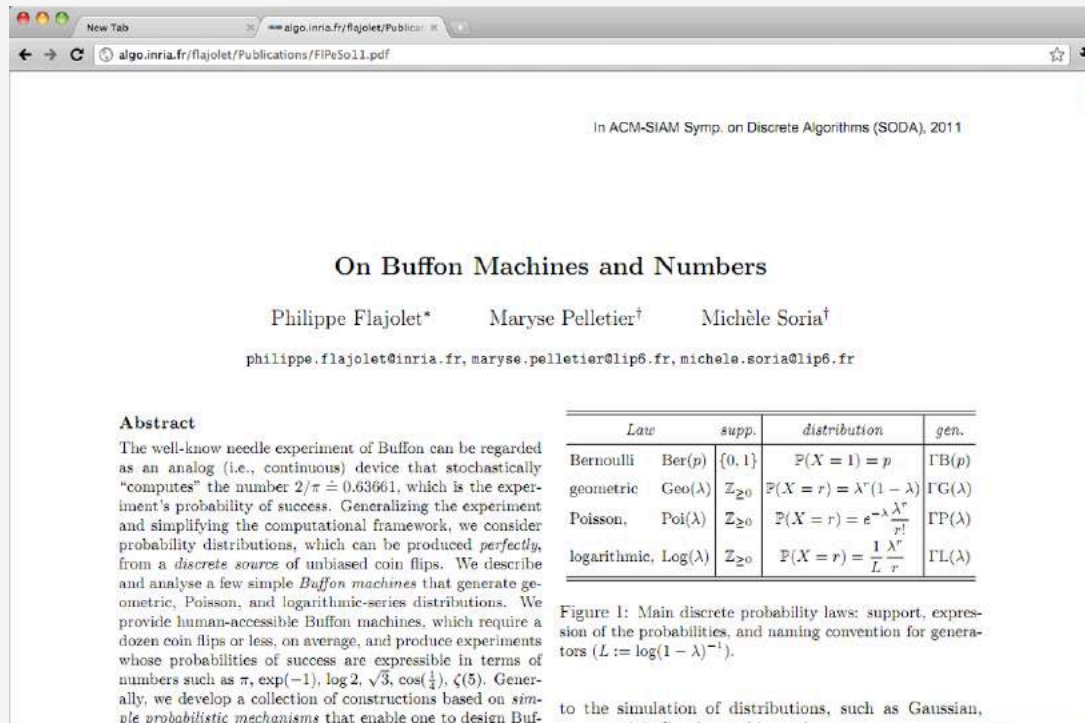
Scientific papers?

Alan Kay: “**The best way to predict the future is to invent it.**”

Scientific papers

When is the last time you visited a library to find a paper?

Did you print the papers to read the last time you refereed a conference?



- Color?
- Links to references?
- Links to detailed proofs?
- Simulations?

↑
why not?

*"I could read it on my iPad
...if I had an iPad
D. E. Knuth*

↑
why?

Question: If it will not be read on paper, why write it as if it will?

Prediction: Someone will soon invent the future (should be easy)

Textbooks

A road to ruin

- prices continue to escalate
- students now rent, not own books
- planned obsolescence? walled garden?

The screenshot shows the Chegg website interface. At the top, there is a navigation bar with links for 'Rent Textbooks', 'Sell Textbooks', 'My Account', and 'Help'. A search bar is present with the text 'Enter ISBN, Title or Author's Name' and buttons for 'FIND BOOKS' and 'SEARCH TIPS'. The search results are for 'Sedgewick algorithms c++'. The first result is 'Bundle of Algorithms in C++, Parts 1-5' with ISBN 020172084X, Edition 3 (Revised), and Author Sedgewick, Robert, van Wyk, Christopher J. The pricing options are 'BUY USED UNAVAILABLE' and 'BUY NEW \$92.49', with a 'LIST PRICE' of \$112.50 and a 'BUY IT' button. The second result is 'Algorithms in C++, Parts 1-4' with ISBN 0201350862, Edition 3 (Revised, Illustrated), and Author Sedgewick, Robert. The pricing options are 'BUY USED \$43.49' and 'BUY NEW \$53.49', with a 'LIST PRICE' of \$69.99 and a 'BUY IT' button. A banner at the top of the results area reads '21-Day "Any Reason" GUARANTEE'.

Is there room for a good textbook?

Will free web resources prevail?

Sedgewick-Wayne publishing model

Two components

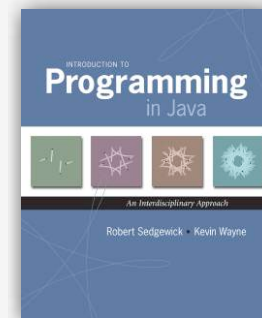
- traditional **textbook** (priced to own)
- forward-looking **booksite** (free)

Textbook

- traditional look-and-feel
- builds on 500 years of experience
- for use while **learning**

Booksite

- supports search
- has code, test data, animations
- links to references
- a living document
- for use while **programming, exploring**



http://www.cs.princeton.edu/introcs/home/

INTRODUCTION TO PROGRAMMING IN JAVA

a textbook for a first course in computer science for the next generation of scientists and engineers

Welcome to our website!

Textbook. Our textbook *Introduction to Programming in Java: An Interdisciplinary Approach* [Amazon - Addison-Wesley] is an interdisciplinary approach to the traditional CS1 curriculum. We teach all of the classic elements of programming, using an "objects-in-the-middle" approach that emphasizes data abstraction. The book is organized around four stages of learning to program:

- **Chapter 1: Elements of Programming** introduces variables; assignment statements; built-in types of data; conditionals and loops; arrays, and input/output, including graphics and sound.
- **Chapter 2: Functions** introduces modular programming. We stress the fundamental idea of dividing a program into components that can be independently debugged, maintained, and reused.
- **Chapter 3: Object Oriented Programming** introduces data abstraction. We emphasize the concept of a data type and its implementation using Java's class mechanism.
- **Chapter 4: Algorithms and Data Structures** introduces classical algorithms for sorting and searching, and fundamental data structures, including stacks, queues, and symbol tables.

A key feature of the book is the manner in which we motivate each programming concept that we address by examining its impact on specific applications, taken from fields ranging from materials science to genomics to astrophysics to internet commerce. This approach highlights the essential idea that mathematics, science, engineering, and computing are intertwined in the modern world.

To preview our material, you can download the preface and Chapter 1.

Booksite. Reading a book and surfing the web are two different activities: This booksite is intended for your use while online (for example, while programming and while browsing the web); the textbook is for your use when initially learning new material and when reinforcing your understanding of that material (for example, when reviewing to prepare for an exam). The booksite consists of the following elements:

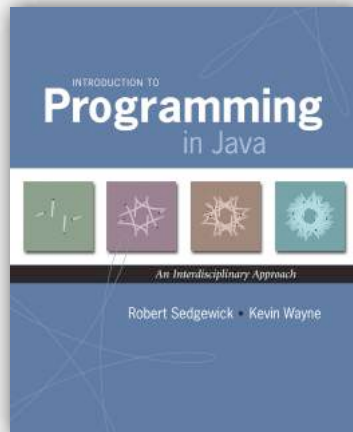
- **Excerpts.** A condensed version of the text narrative for reference while online.
- **Exercises.** Hundreds of exercises, including many from the text and some solutions.
- **Java code.** Hundreds of easily downloadable complete Java programs, along with test data.
- **References.** Extensive links throughout to related material on the web.

To adopt. Here are some of the distinctive features of our textbook and a marketing flyer. If you wish to consider adoption, please fill out this form to request a copy of the textbook or ask for more information.

Copyright © 2007 Robert Sedgewick and Kevin Wayne. All rights reserved.

Textbook

Part I: Programming (2009)



Prolog

1 Elements of Programming

- Your First Program
- Built-in types of Data
- Conditionals and Loops
- Arrays
- Input and Output
- Case Study: Random Surfer

2 Functions and Modules

- Static Methods
- Libraries and Clients
- Recursion
- Case Study: Percolation

3 Data Abstraction

- Data Types
- Creating DataTypes
- Designing Data Types
- Case Study: N-body

4 Algorithms/Data Structures

- Performance
- Sorting and Searching
- Stacks and Queues
- Symbol Tables
- Case Study: Small World

Part II: Computer science (in preparation)

5 A Computing Machine

- Data representations
- TOY machine
- Instruction Set
- Machine Language Coding
- Simulator

6 Building a Computer

- Boolean Logic and Gates
- Combinational Circuits
- Sequential Circuits
- TOY machine architecture

7 Theory of Computation

- Formal Languages
- Turing Machines
- Universality
- Computability
- Intractability

8 Systems

- Library Programming
- Compilers and Interpreters
- Operating Systems
- Networks
- Applications Systems

9 Scientific Computation

- Precision and Accuracy
- Differential Equations
- Linear Algebra
- Optimization
- Data Analysis
- Simulation

introcs.cs.princeton.edu

Introduction to Programming in Java: An Interdisciplinary Approach

<http://introcs.cs.princeton.edu/java/home/>

Reader Google

home ▾ Princeton ▾ reference (1,412) ▾ rsrch ▾ save ▾ shop ▾ travel ▾ teach ▾ Yahoo! YouTube Wikipedia

INTRODUCTION TO PROGRAMMING IN JAVA

*a textbook for a first course in computer science
for the next generation
of scientists and engineers*

Textbook. Our textbook *Introduction to Programming in Java* [Amazon · Addison-Wesley] is an interdisciplinary approach to the traditional CS1 curriculum. We teach all of the classic elements of programming, using an "objects-in-the-middle" approach that emphasizes data abstraction. A key feature of the book is the manner in which we motivate each programming concept by examining its impact on specific applications, taken from fields ranging from materials science to genomics to astrophysics to internet commerce. The book is organized around four stages of learning to program:

- *Chapter 1: Elements of Programming* introduces variables; assignment statements; built-in types of data; conditionals and loops; arrays; and input/output, including graphics and sound.
- *Chapter 2: Functions* introduces modular programming. We stress the fundamental idea of dividing a program into components that can be independently debugged, maintained, and reused.
- *Chapter 3: Object-Oriented Programming* introduces data abstraction. We emphasize the concept of a data type and its implementation using Java's class mechanism.
- *Chapter 4: Algorithms and Data Structures* introduces classical algorithms for sorting and searching, and fundamental data structures, including stacks, queues, and symbol tables.

Booksite. Reading a book and surfing the web are two different activities: This booksite is intended for your use while online (for example, while programming and while browsing the web); the textbook is for your use when initially learning new material and when reinforcing your understanding of that material (for example, when reviewing for an exam). The booksite consists of the following elements:

- *Excerpts.* A condensed version of the text narrative for reference while online.
- *Exercises.* Hundreds of exercises and some solutions.
- *Java code.* Hundreds of easily downloadable Java programs and real-world data sets.

To get started. Here are instructions for installing a Java programming environment [Mac OS X · Windows · Linux]. We also provide I/O libraries for reading and writing text and binary data, drawing graphics, and producing sound.

To adopt. Here are some of the distinctive features of our textbook and a marketing flyer. To preview our material, you can download the preface and Chapter 1. If you wish to consider adoption, please fill out this form to request a copy of the textbook or ask for more information.

Last modified on February 05, 2012.

Copyright © 2002–2012 Robert Sedgwick and Kevin Wayne. All rights reserved.

INTRO TO PROGRAMMING

1. Elements of Programming
2. Functions
3. OOP
4. Data Structures

INTRO TO CS

0. Prologue
5. A Computing Machine
6. Building a Computer
7. Theory of Computation
8. Systems
9. Scientific Computation

ALGORITHMS, 4TH EDITION

WEB RESOURCES

- FAQ
- Data
- Code
- Errata
- Appendices
- Lecture Slides
- Programming Assignments

Search booksite...

- Text digests
- Ready-to-use code
- Supplementary exercises/answers
- Links to references and sources
- Modularized lecture slides
- Programming assignments
- Demos for lecture and precept
- Simulators for self-study
- Scientific applications

10000+ files

2000+ Java programs

50+ animated demos

1.2 million unique visitors in 2011

Algorithms for the masses



Central thesis for *Algorithms* (1975)

All science/engineering students need an **algorithms** course

Algorithms embraces a significant body of knowledge that is

- intellectually challenging
- pervasive in modern life
- critical to modern science and engineering

Barriers

- no room in curriculum
- need to implement all the algorithms (!)
- need to analyze all the algorithms (!)
- need to pick the most important ones

Current status of “Algorithms” (2012)

[Joint work with Kevin Wayne since 2007]

Any science/engineering student can appreciate

- data abstraction and modular programming
- 50+ classic and important algorithms and data structures
- historical context, applications
- relationships to OR, theory of algorithms

Algorithms (4th edition) and booksite (2011)

back to basics
(one book) →



algs4.cs.princeton.edu

ALGORITHMS, 4TH EDITION

essential information that every serious programmer needs to know about algorithms and data structures

Textbook. The textbook *Algorithms, 4th Edition* by Robert Sedgwick and Kevin Wayne [Amazon : Pearson : InformIT] surveys the most important algorithms and data structures in use today. The textbook is organized into six chapters:

- **Chapter 1: Fundamentals** introduces a scientific and engineering basis for comparing algorithms and making predictions. It also includes our programming model.
- **Chapter 2: Sorting** considers several classic sorting algorithms, including insertion sort, mergesort, and quicksort. It also includes a binary heap implementation of a priority queue.
- **Chapter 3: Searching** describes several classic symbol table implementations, including binary search trees, red-black trees, and hash tables.
- **Chapter 4: Graphs** surveys the most important graph processing problems, including depth-first search, breadth-first search, minimum spanning trees, and shortest paths.
- **Chapter 5: Strings** investigates specialized algorithms for string processing, including radix sorting, substring search, tries, regular expressions, and data compression.
- **Chapter 6: Context** highlights connections to systems programming, scientific computing, commercial applications, operations research, and intractability.

Applications to science, engineering, and industry are a key feature of the text. We motivate each algorithm that we address by examining its impact on specific applications.

Booksite. Reading a book and surfing the web are two different activities: This booksite is intended for your use while online (for example, while programming and while browsing the web); the textbook is for your use when initially learning new material and when reinforcing your understanding of that material (for example, when reviewing for an exam). The booksite consists of the following elements:

- Excerpts. A condensed version of the text narrative, for reference while online.
- Java code. The algorithms and clients in this textbook.
- Exercise solutions. Solutions to selected exercises.

To get started. Here are instructions for setting up our recommended Java programming environment [Mac OS X - Windows - Linux].

To adopt. Here is a marketing flyer. Here is the preface. If you are considering adoption, you can ask the authors for more information or request an examination copy.

Last modified on September 16, 2011.

algs4.cs.princeton.edu

Algorithms, 4th Edition by Robert Sedgwick and Kevin Wayne

http://algs4.cs.princeton.edu/home/

ALGORITHMS, 4TH EDITION

essential information that every serious programmer needs to know about algorithms and data structures

Textbook. The textbook *Algorithms, 4th Edition* by Robert Sedgwick and Kevin Wayne [[Amazon](#) · [Pearson](#) · [InformIT](#)] surveys the most important algorithms and data structures in use today. The textbook is organized into six chapters:

- [Chapter 1: Fundamentals](#) introduces a scientific and engineering basis for comparing algorithms and making predictions. It also includes our programming model.
- [Chapter 2: Sorting](#) considers several classic sorting algorithms, including insertion sort, mergesort, and quicksort. It also includes a binary heap implementation of a priority queue.
- [Chapter 3: Searching](#) describes several classic symbol table implementations, including binary search trees, red-black trees, and hash tables.
- [Chapter 4: Graphs](#) surveys the most important graph processing problems, including depth-first search, breadth-first search, minimum spanning trees, and shortest paths.
- [Chapter 5: Strings](#) investigates specialized algorithms for string processing, including radix sorting, substring search, tries, regular expressions, and data compression.
- [Chapter 6: Context](#) highlights connections to systems programming, scientific computing, commercial applications, operations research, and intractability.

Applications to science, engineering, and industry are a key feature of the text. We motivate each algorithm that we address by examining its impact on specific applications.

Booksite. Reading a book and surfing the web are two different activities: This booksite is intended for your use while online (for example, while programming and while browsing the web); the textbook is for your use when initially learning new material and when reinforcing your understanding of that material (for example, when reviewing for an exam). The booksite consists of the following elements:

- *Excerpts.* A condensed version of the text narrative, for reference while online.
- *Java code.* The [algorithms and clients](#) in this textbook.
- *Exercise solutions.* Solutions to selected exercises.

To get started. Here are instructions for setting up our recommended Java programming environment [[Mac OS X](#) · [Windows](#) · [Linux](#)].

To adopt. Here is a [marketing flyer](#). Here is the [preface](#). If you are considering adoption, you can [ask the authors for more information](#) or [request an examination copy](#).

Last modified on September 16, 2011.

Copyright © 2002–2012 Robert Sedgwick and Kevin Wayne. All rights reserved.

Navigation menu (left sidebar):

- ALGORITHMS, 4TH EDITION
- 1. Fundamentals
- 2. Sorting
- 3. Searching
- 4. Graphs
- 5. Strings
- 6. Context
- RELATED BOOKSITES
- WEB RESOURCES
- FAQ
- Data
- Code
- Errata
- References
- Lecture Slides
- Programming Assignments
- Search booksite...

- Text digests
- Ready-to-use code
- Supplementary exercises/answers
- Links to references and sources
- Modularized lecture slides
- Programming assignments
- Demos for lecture and precept
- Simulators for self-study
- Scientific applications

Top 100 algorithms

Algs4 code (and much more) all available online

The screenshot shows the website for "Java Algorithms and Clients". The page title is "JAVA ALGORITHMS AND CLIENTS". Below the title, there is a paragraph explaining the book's goal: "Our original goal for this book was to cover the 50 algorithms that every programmer should know. We use the word *programmer* to refer to anyone engaged in trying to accomplish something with the help of a computer, including scientists, engineers, and applications developers, not to mention college students in science, engineering, and computer science."

Below the paragraph, there is a section titled "Algorithms and clients in the textbook." which states: "The list below includes a few more than 100 Java programs (some are clients, some others are basic infrastructure). Click on the program name to access the Java code; click on the description to access the javadoc; click on the data file names to access the data. You can download all of the programs as [algs4.jar](#) and the data as [algs4-data.zip](#)."

The main content is a table with columns for "1", "FUNDAMENTALS", and "DATA". The table lists various algorithms and their corresponding data files.

1	FUNDAMENTALS	DATA
-	BinarySearch.java binary search	tinyW.txt tinyT.txt largeW.txt largeT.txt
-	RandomSeq.java random numbers in a given range	-
-	Average.java average of a sequence of numbers	-
-	Cat.java concatenate files	in1.txt in2.txt
-	Shuffle.java Knuth shuffle	cards.txt
-	Counter.java counter	-
-	StaticSETofInts.java set of integers	-
-	Whitelist.java whitelist client	tinyW.txt tinyT.txt largeW.txt largeT.txt
-	Vector.java mathematical vector	-
-	Date.java date	-
-	Transaction.java transaction	-
-	Point2D.java point	-
-	Interval1D.java 1-d interval	-
-	Interval2D.java 2-d interval	-
1.1	ResizingArrayStack.java LIFO stack (resizing array)	tobe.txt
1.2	Stack.java LIFO stack (linked list)	tobe.txt
-	ResizingArrayQueue.java FIFO queue (resizing array)	tobe.txt
1.3	Queue.java FIFO queue (linked list)	tobe.txt
1.4	Bag.java multiset	-
-	Stopwatch.java timer	-
-	ThreeSum.java brute-force three sum	1Kints.txt 2Kints.txt 4Kints.txt 8Kints.txt



Modular programming style

- one-click download
- test data
- variants
- robust library versions
- typical clients

Messages for algorithms students

Modern programming models are for you

Algorithms are important and useful in scientific, engineering, and commercial applications of all sorts

Performance matters

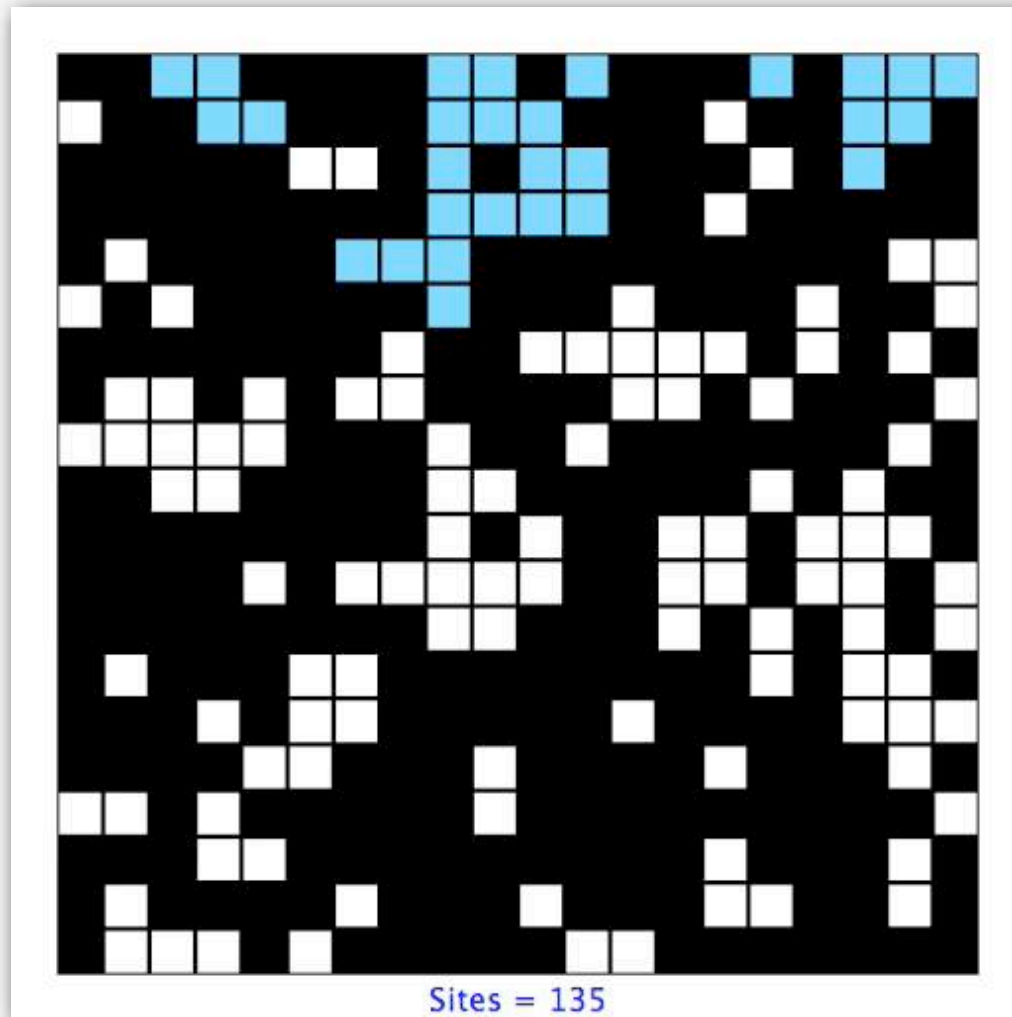
Classic algorithms for sorting, searching, graphs and strings have enabled the development of the computational infrastructure that surrounds us

A great many more important and useful algorithms remain to be discovered

Intrinsic limitations exist

Familiar and easy-to-motivate applications

Percolation



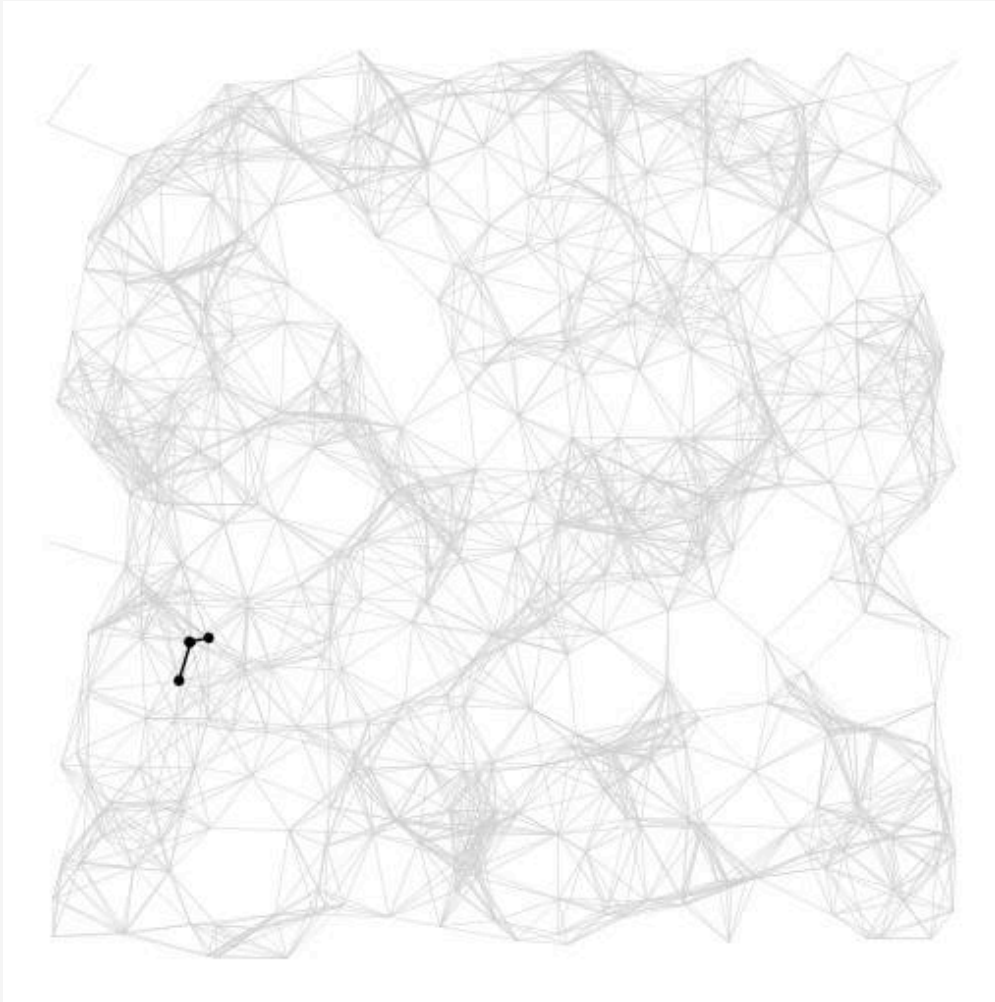
union-find

Ideal example/assignment

- teaches a basic CS concept
- solves an important problem
- intellectually engaging
- modular program
- is open-ended

Familiar and easy-to-motivate applications

Prim's MST algorithm



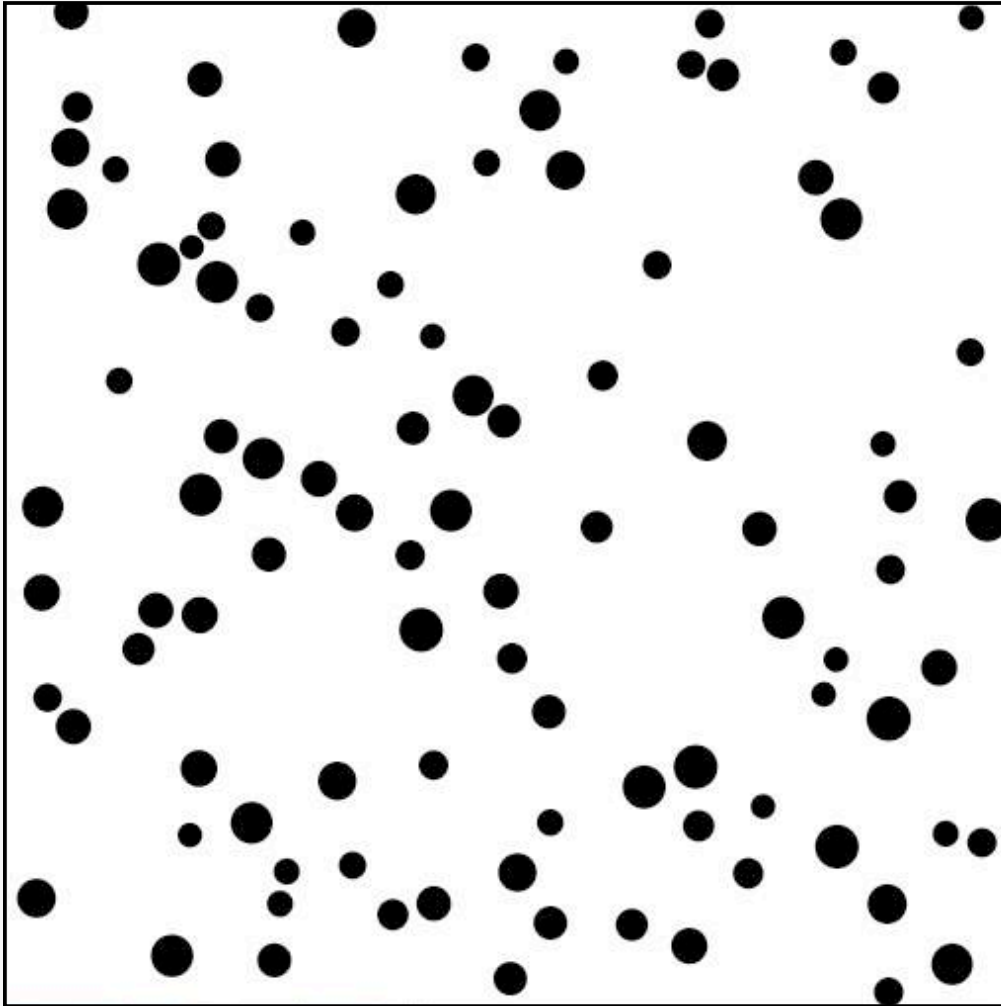
graph search

Ideal example/assignment

- teaches a basic CS concept
- solves an important problem
- intellectually engaging
- modular program
- is open-ended

Familiar and easy-to-motivate applications

Bose-Einstein colliding particle simulation



priority queue

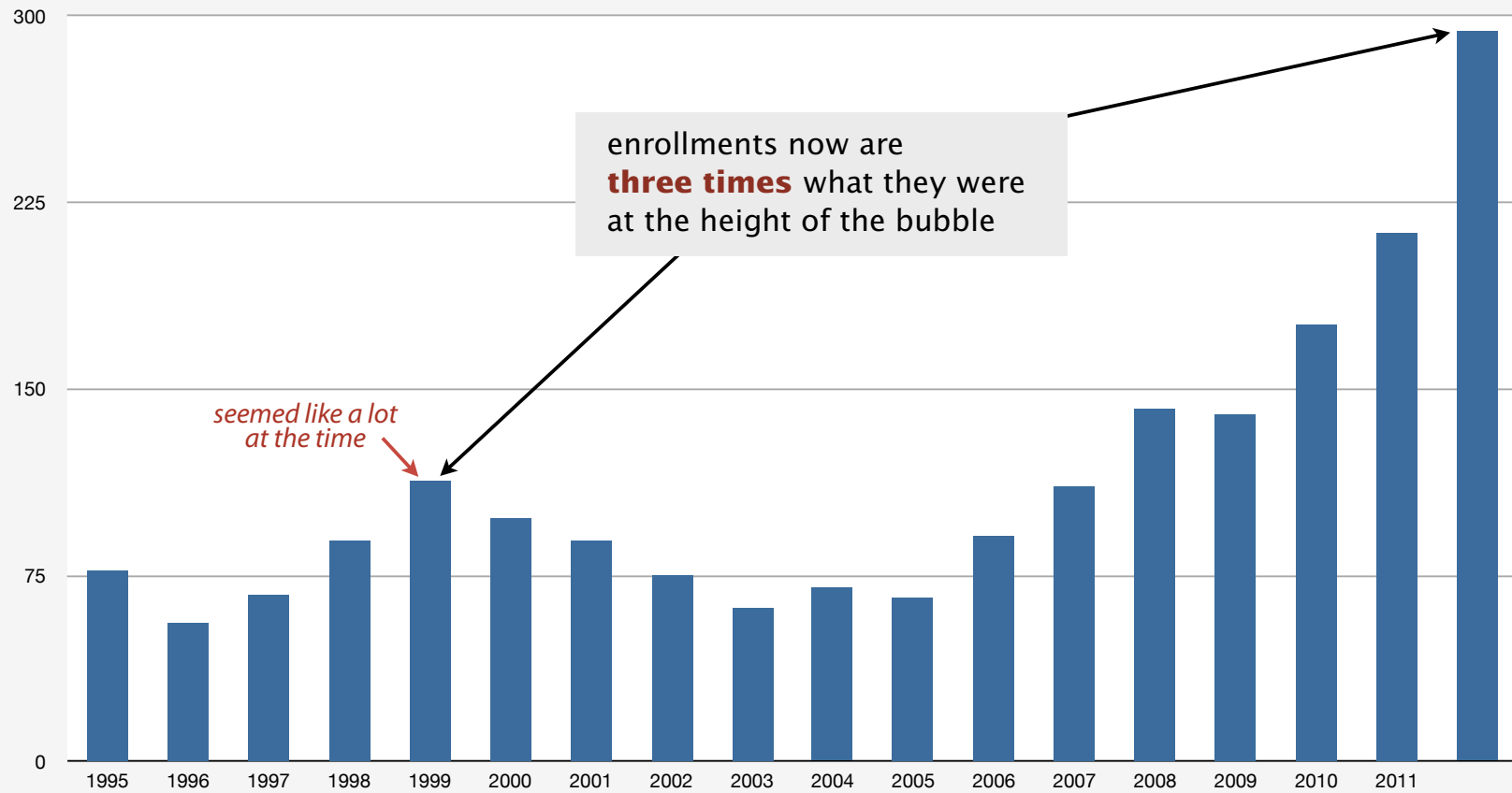
Ideal example/assignment

- teaches a basic CS concept
- solves an important problem
- intellectually engaging
- modular program
- is open-ended

Enrollments in algorithms course

are also skyrocketing

“Algorithms” enrollments



25+% of all Princeton students

Key factor in increase: All students in “CS for everyone” can take “Algorithms”

Summary

The scientific method is an essential ingredient in programming.
Embracing, supporting, and leveraging science in
intro CS and algorithms courses can serve large numbers of students.

Proof of concept: First-year courses at Princeton

- 50+% of Princeton students in a single intro course
- 25+% of Princeton students in a single **algorithms** course

Next goals:

- 50+% of **all college students** in an intro CS course
- 25+% of **all college students** in an algorithms course



ALGORITHMS FOR THE MASSES

Algorithms for the masses

Robert Sedgewick
Princeton University

