



COMP 482 / ELEC 420: Design & Analysis of Algorithms

John Greiner

Course Overview

Pragmatics

- Prerequisites

- Textbook  & online errata 

- Assignments

- Mostly written homeworks, roughly weekly
- One programming & writing project

- Exams

- Midterm & final, both take-home

- Slip days

Course Web Page

For all pragmatic details, see
www.owl.net.rice.edu/~comp482/




Read course web pages!

Includes

- Course notes
- Assignments, exams, solutions
- How to contact course staff
- Policies

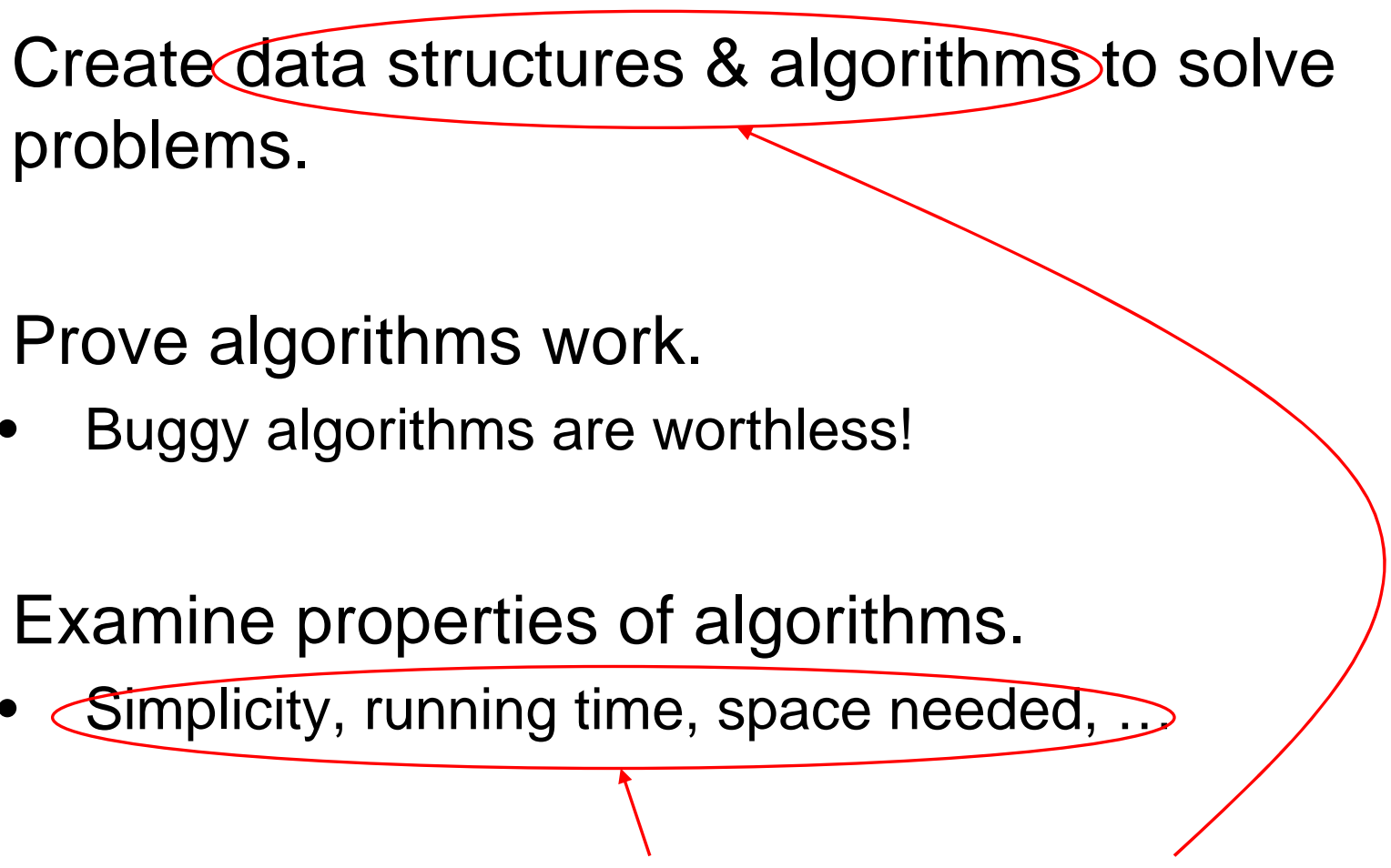
Your To-Do List

- Buy textbook.
- Read course administrative details on web.
- Subscribe to course mailing list: `comp482-discussion-1`. 
- Skim [CLRS] 1-2 – background material.



So, what is this course all about?

Solving Interesting Problems

1. Create data structures & algorithms to solve problems.
 2. Prove algorithms work.
 - Buggy algorithms are worthless!
 3. Examine properties of algorithms.
 - Simplicity, running time, space needed, ...
- 

Algorithm efficiency depends on data structure used.
Two parts of the same idea.

Three Kinds of Problems: 1

Decision problems

Given **input**, does it have a certain property?

- Is an integer prime?
- Is a graph planar? (Can be drawn on a plane without overlapping edges?)

Also called an *instance* of the problem.

Three Kinds of Problems: 2

Function problems

Given input, compute the unique solution.

- Compute the product of two matrices.
- Sort an array.

Three Kinds of Problems: 3

Search problems

Given input, compute any one of a set of solutions.

- Find the factors of an integer.
- Find shortest path between two points in graph.
- Find shortest path between each pair of points in graph.
- Find longest path between two points in graph.

Properties of Algorithms: 1

Time efficiency

- As a function of its input size, how long does it take?

Space efficiency

- As a function of its input size, how much additional space does it use?

Especially, worst- & average-case asymptotic efficiency.

Properties of Algorithms: 2

Simplicity

- Informal notion
- Easier to code correctly
- Lower constant factors in time efficiency (typically)
- Easier to explain
- Easier to prove correct (probably)
- More mathematically “elegant”

A Quick Example of Interesting Results

Roughly how much time for each?

1. Find shortest path between two points in graph.
2. Find shortest path between each pair of points in graph.
3. Find longest path between two points in graph.



1. $O(V \times E)$

2. $O(V \times E)$

3. No known polynomial-time algorithm.

Despite #2's demand for more info!

Despite their similarity.

Abstract vs. Concrete Data Structures

Abstract:

What can it do?

I.e., its interface – what operations?

Queue:

enqueue(elt,Q)

dequeue(Q)

isEmpty(Q)

Concrete:

How is it implemented?

How efficient?

Array-Queue:

Elements stored circularly,
with first & last indices.

Constant-time operations.

Familiar idea, but...
people (& algorithm texts) often don't distinguish.

Mutability

Mutable:

**Operations can change
data structure.**

enqueue(elt,Q)
modifies Q to have new elt.

Immutable:

**Operations can't change
data structure.**

enqueue(elt,Q)
creates & returns Q' to have
new elt. Q unchanged.

Traditional & this course: mutable.

Why Should You Care?

Efficiency important in most programs.
Choose/design appropriate data structures & algorithms.

No one familiar with all data structures & algorithms.
Know the basics. Know how to figure out the rest.

Syllabus Outline

- Finish course overview – extended example
 - Math background – review & beyond
 - Data structures & algorithms
 - Sorting-like
 - Tree-based
 - String-based
 - Optimization/Numeric
 - “NP-completeness”
 - What is it?
 - Why is it important?
 - What can we do about it?
- Techniques, as needed:
- Randomization
 - Amortized analysis
 - Probabilistic analysis

Questions?



Please ask at any time!



Extended Example: Multiplication

Illustrates some of course's interesting ideas.
Just an overview – not too much detail yet.

How to multiply two integers: $x \times y$.
Common & familiar. Simple?

Suggested algorithms & efficiency?



Multiplication Algorithm 1

It's a single basic machine operation, $O(1)$ time.

? Problem with this answer? ?

Only applies to bounded-sized integers.

Instead, assume unbounded-sized integers.

E.g., as in Scheme or Mathematica.

Explicitly include this assumption.

Multiplication Algorithm 2

Repeated addition: $x \times y = \underbrace{x + \dots + x}_{y \text{ copies}} = \underbrace{y + \dots + y}_{x \text{ copies}}$

How long for each addition?

- Grade-school algorithm takes time \propto result length.
 - For simplicity, assume x, y have same length.
 - Length $n = \log_2 x$ (choice of base 2 is arbitrary)
- $O(n)$

Back to multiplication:

- $O(n \times x) = O(n \times 2^n)$

Multiplication Algorithm 3

Grade-school “long-hand” algorithm.

$$\begin{array}{r} \mathbf{x=} \quad 38 \\ \mathbf{y=} \quad \times \underline{473} \\ \quad 114 \\ \quad 2660 \\ + \underline{15200} \\ \quad 17974 \end{array}$$

n multiplications of (x by 1 bit of y)
+
n additions of the resulting products.

Basically the same as
bit-shifting algorithms.

? Total? ?

$$O(n \times n + n \times n) = O(n^2)$$

Much better!

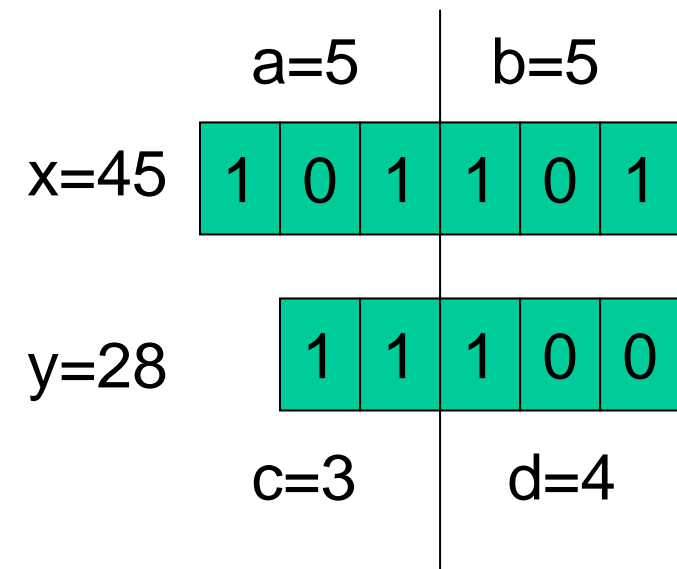
Multiplication Algorithm 4: Part 1

Karatsuba's algorithm, 1962

Break the bitstrings of x, y into halves.

$$x = a \times 2^{n/2} + b = a \ll n/2 + b$$

$$y = c \times 2^{n/2} + d = c \ll n/2 + d$$



$$xy = \underline{ac} \ll n + (\underline{ad} + \underline{bc}) \ll n/2 + \underline{bd}$$

Compute the 4 subproducts recursively.

Example of a *divide-and-conquer* algorithm.

Multiplication Algorithm 4: Part 1

How long does this take?

Form recurrence equation:

$$T(n) = \begin{cases} k & n = 1 \\ 4 \times T\left(\frac{n}{2}\right) + k \times n & n > 1 \end{cases}$$

k = arbitrary
constant to fill in
for the details

Solve recurrence equation.

- How? Discussed next week.
- $T(n) = O(n^2)$

4 recursive
subproducts +
additions & shifts

No better than previous algorithm.

Multiplication Algorithm 4: Part 2

Previous:

$$xy = \underline{ac} \ll n + (\underline{ad+bc}) \ll n/2 + \underline{bd}$$

Regroup (very non-obvious step!):

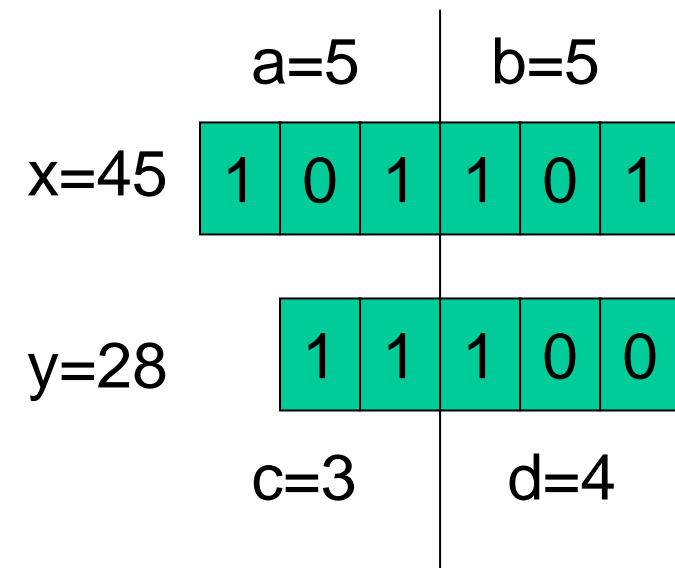
$$u = \underline{(a+b)} \times \underline{(c+d)}$$

$$v = \underline{ac}$$

$$w = \underline{bd}$$

$$xy = v \ll n + (u-v-w) \ll n/2 + w$$

Only 3 subproducts!



Multiplication Algorithm 4: Part 2

How long does this take?

$$T'(n) = \begin{cases} k' & n=1 \\ 3 \times T'\left(\frac{n}{2}\right) + k' \times n & n > 1 \end{cases}$$

k' = a new, larger constant

$$T'(n) = 3 \times k' \times n^{\log_2 3} - 2 \times k' \times n = O(n^{\log_2 3}) \approx O(n^{1.59})$$

More complicated, but asymptotically faster.

Multiplication Algorithm 4: Part 2

Previous:

$$u = \underline{(a+b)} \times \underline{(c+d)} \qquad v = \underline{ac} \qquad w = \underline{bd}$$

$$xy = v \ll n + (u-v-w) \ll n/2 + w$$

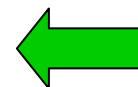
An overlooked detail:

$a+b$ and $c+d$ can be $n/2+1$ bit numbers. Doesn't fit our recurrence eq'n.

Solution: Single bits.

$$a+b = \underbrace{a_1}_{\text{Single bits}} \ll n/2 + b_1$$

$$c+d = \underbrace{c_1}_{\text{Single bits}} \ll n/2 + d_1$$



Break sums into
 $(n/2+1)^{\text{th}}$ bit & $n/2$ bits.

$$\underline{(a+b)} \times \underline{(c+d)} = \underbrace{(a_1 \ c_1)}_{\text{Single bits}} \ll n + \underbrace{(a_1 \ d_1 + b_1 \ c_1)}_{\text{Single bits}} \ll n/2 + \underline{b_1 \ d_1}$$

\uparrow
 $O(1)$ time

$\swarrow \searrow$
 $O(n/2)$ time

\uparrow
 $T'(n/2)$ time

Multiplication Algorithms 5—8

Toom-Cook algorithm, 1963, 1966: $O(n^{1+\epsilon})$

- Generalizes both long-hand & Karatsuba
- Based upon polynomial multiplication & interpolation.

Karp's FFT-based algorithm: $O(n \log^2 n)$

Schoenhage & Strassen's FFT-based algorithm, 1971:
 $O(n \log n \log \log n)$

- Example of *log-shaving* – slightly better running-times with lower logarithmic terms.

Fürer's FFT-based algorithm, 2007: $O(n \log n 2^{O(\log^* n)})$

- More divide-and-conquer → More log-shaving
- Best known.

Approaching the conjectured
 $\Omega(n \log n)$ lower bound.

Multiplication Algorithms 9, ...

Use parallelism.

Even serial processors use some bit-level parallelism.
Divide-&-conquer & FFT algorithms easily parallelized.

Multiplication Algorithms Summary

Asymptotically-faster often →

- more complicated
- higher constant factors
- slower for typical input sizes.

Karatsuba & FFT-based algs.
practical for cryptography's
large numbers.

Good ideas lead to more
good ideas

- Algs. build upon previous algs.' ideas
- Karatsuba generalizes to Strassen's matrix multiplication, [CLRS] 28.2
- Toom-Cook & FFT generalize to polynomial multiplication

Algorithm Analysis Summary

1. State problem.
2. Characterize the input size.
3. State algorithm.
 - Often difficult, of course
4. Prove algorithm correctness.
 - Necessary!
5. Determine its resource usage.
 - Often via recurrence equations
 - Allows comparison of algorithms
 - Can decide which algorithm suitable for given application
 - Can guide the search for better algorithms

We almost missed an important detail that would have produced an incorrect analysis.