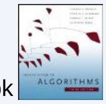
Design & Analysis of Algorithms
COMP 482 / ELEC 420

John Greiner

Pragmatics

- Prerequisites
- Textbook  — To do: [CLRS] 1-2
- Wikipedia, …
- **www.clear.rice.edu/comp482/** — Policies
- OWL-Space
- Assignments, late policy, exams — #0, Start #1

Solving Interesting Problems

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

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

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.

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 textbooks** 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.

Most algorithm textbooks & this course: mostly mutable.

Outline of Semester

- Finish course overview – extended example
- Math background – review & beyond
- Algorithms & data structures
  - Techniques, as needed:
    - Randomization
    - Probabilistic analysis
    - Amortized analysis
    - Dynamic programming
- <u>Really</u> hard problems

7

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 & their efficiency? ?

8

Multiplication Algorithm 1

Single basic machine operation, O(1) time.

? Problem with this answer? ?

Only applies to bounded-sized integers.
Instead, explicitly assume unbounded-sized integers.

9

Multiplication Algorithm 2

$$x \times y = \underbrace{x+\ldots+x}_{\text{y copies}} = \underbrace{y+\ldots+y}_{\text{x copies}}$$

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:

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

10

Multiplication Algorithm 3

Grade-school "long-hand" algorithm.

```
x=      38
y=   × 473
       114
      2660
   + 15200
     17974
```

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!

11

Multiplication Algorithm 4: Part 1

Karatsuba, 1962

Break the bit strings of x,y into halves.

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

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

	a=5			b=5		
x=45	1	0	1	1	0	1
y=28	0	1	1	1	0	0
	c=3			d=4		

xy = <u>ac</u> << n + (<u>ad</u>+<u>bc</u>) << n/2 + <u>bd</u>

Compute 4 subproducts recursively.

Divide-and-conquer

12

Multiplication Algorithm 4: Part 1

How long does this take?

Form recurrence equation:

$$T(n)=\begin{cases} k & n=1 \\ \underbrace{4\times T\left(\frac{n}{2}\right)+k\times n}_{\text{4 subproducts + additions \& shifts}} & n>1 \end{cases}$$

k = arbitrary constant to fill in for the details

Solve recurrence equation.

- How? Discussed next week.
- $T(n) = O(n^2)$ – No better than previous.

Multiplication Algorithm 4: Part 2

Previous:

xy = ac << n + (ad+bc) << n/2 + bd

	a=5			b=5		
x=45	1	0	1	1	0	1
y=28	0	1	1	1	0	0
	c=3			d=4		

Regroup (very non-obvious step!):

u = (a+b) × (c+d)
v = ac
w = bd
xy = v << n + (u-v-w) << n/2 + w

Only 3 subproducts! But more additions & shifts.

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

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

More complicated, but asymptotically faster.

Multiplication Algorithm 4: Part 2

Previous:

u = (a+b) × (c+d)   v = ac   w = bd
xy = v << n + (u-v-w) << n/2 + w

An overlooked detail:

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

Single bits.

Solution:

$a+b = a_1 << n/2 + b_1$
$c+d = c_1 << n/2 + d_1$

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

$(a+b) \times (c+d) = (a_1\, c_1) << n + (a_1\, d_1 + b_1\, c_1) << n/2 + b_1\, d_1$

O(1) time   O(n/2) time

Multiplication Algorithms 5—8

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

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

FFT-based:

Karp: $O(n \log^2 n)$

Schoenhage & Strassen, 1971: $O(n \log n \log\log n)$

Fürer, 2007: $O(n \log n\, 2^{O(\log^* n)})$

log-shaving – slightly improving logarithmic terms

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.

Summary

Asymptotically-faster

often →

more complicated,
higher constant factors, &
slower for typical input sizes.

Good ideas lead to more good ideas

Karatsuba generalizes to Strassen's matrix multiplication [CLRS] 4.2 & Toom-Cook.

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
   - Compare algorithms
   - Decide which algorithm suitable for given application
   - Guide the search for better algorithms

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