CS 361
Algorithms & Data Structures
Lecture 1

Prof. Tom Hayes
University of New Mexico
8-21-2012

Who we are

Prof. Tom Hayes, hayes@cs.unm.edu
Office: FEC 149
Hours: TuW 2:00-2:50 and by appt
Phone: 277-9328 (deprecated)

TA: Matt Peterson, mpeterson@unm.edu
Office: FEC 126
Hours: MW 4:00-4:50 and by appt

Skill Objectives

- Learn how to learn
- How to think about a problem
- How to think about an algorithm
- Careful reasoning, reading, and writing
- Know lots of algorithms
  - (and their diverse uses)

What about data structures?

- Just an algorithm(s) in a box
- Reminds us to keep information wisely
- Alternative viewpoint:
  - Algorithms are just ways to modify data structures

Learn how to learn

- Reading assignments
  - Textbook: *Algorithm Design* by Kleinberg & Tardos
  - Read them **before** class
- Ask lots of questions.
  - answer them yourself first, but not last
  - Never stop asking and answering
- Make full use of professor, TA, study groups

Thinking about problems

- What is the input?
  - Think of as many examples as you can.
  - What are extreme/boundary cases?
- What is the output?
  - Work examples by hand.
  - Think about your process
- How is success (correctness) measured?
- How is efficiency measured?

Algorithm descriptions

- Algorithms can be given in
  - one phrase or sentence
  - one paragraph
  - high-level pseudocode
  - low-level pseudocode
  - actual code
- Each level is essential!

Understanding an algorithm

- First, understand the problem
- Describe algorithm at ≥2 levels of detail
- Does the algorithm work? Why? Prove it!
- How long does it take? Why? Prove it!
  - Input scaling? How big can it go?
- Can it be done faster?
  - Yes: do it.
  - No: Why not? Prove it!

Careful reasoning

- How can we prove the algorithm works?
  - Run unit tests? No (but do it anyway)
  - Try all possible inputs? No, impossible
  - A carefully justified argument? Yes!
- What is the standard of proof?
  - A jury of your peers. (or bosses/profs)
  - Read and re-read the textbook. Emulate its clarity and style.

Before we start...

- All students: email me, hayes@unm.edu
  Tell me: your name, email (that you will check), recent CS and math courses, registered or not?, major/department, year/grad/undergrad. Can you come to the office hours? Are you a graduating senior?

First Assignments

Reading: Chapter 1, by Tuesday.
Sections 2.1-2.4, by the following Tuesday.

Written Assignment 1: due Thurs, Sept 6

Exercises 1.1, 1.2, 1.3, 1.4, 1.6

Assignments

Reading assignments: weekly. It is important to do these before class.

Written assignments: mostly exercises from the textbook. These are very important.

Programming assignments: in Java. Detailed instructions, together with unit tests, will be provided. Grading will be **mostly** automated, so ensure that your code passes all available tests (we hold some back)

What is Good Coding?

What is Good Coding?

Ultimately, it's whatever works for you!

Scalability

Imagine: you write a function foo to find the n'th term in this sequence:

0, 1, 1, 2, 3, 5, 8, 13, ...

```
int foo(int n) {
   if (n<=1) return n; //else
   return foo(n-1)+foo(n-2);
 }
```

Scalability

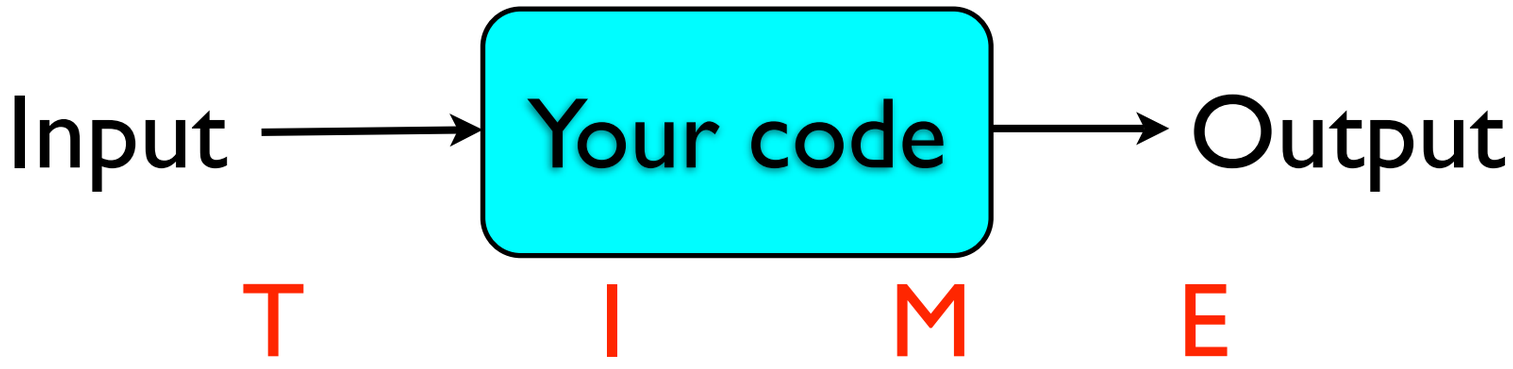
Imagine: you write a function foo to find the n'th term in this sequence:

0, 1, 1, 2, 3, 5, 8, 13, ...

Now, you've tested it up to n=20, and it works great! But, in production code, you need your solution to scale up to n=100. Will it do so? How much will it cost? (in extra time, memory, servers, ...)

Scalability

- The real question:



- as size(input) grows,
  - is output correct?
  - how fast does time grow? i.e. what is the *function* T(n)? (T=time, n=input size)

Scalability

Imagine: you write a function foo to find the n'th term in this sequence:

0, 1, 1, 2, 3, 5, 8, 13, ...

foo(10) = 55

foo(20) = 6765

foo(30) = 832040

foo(40) = 102334155

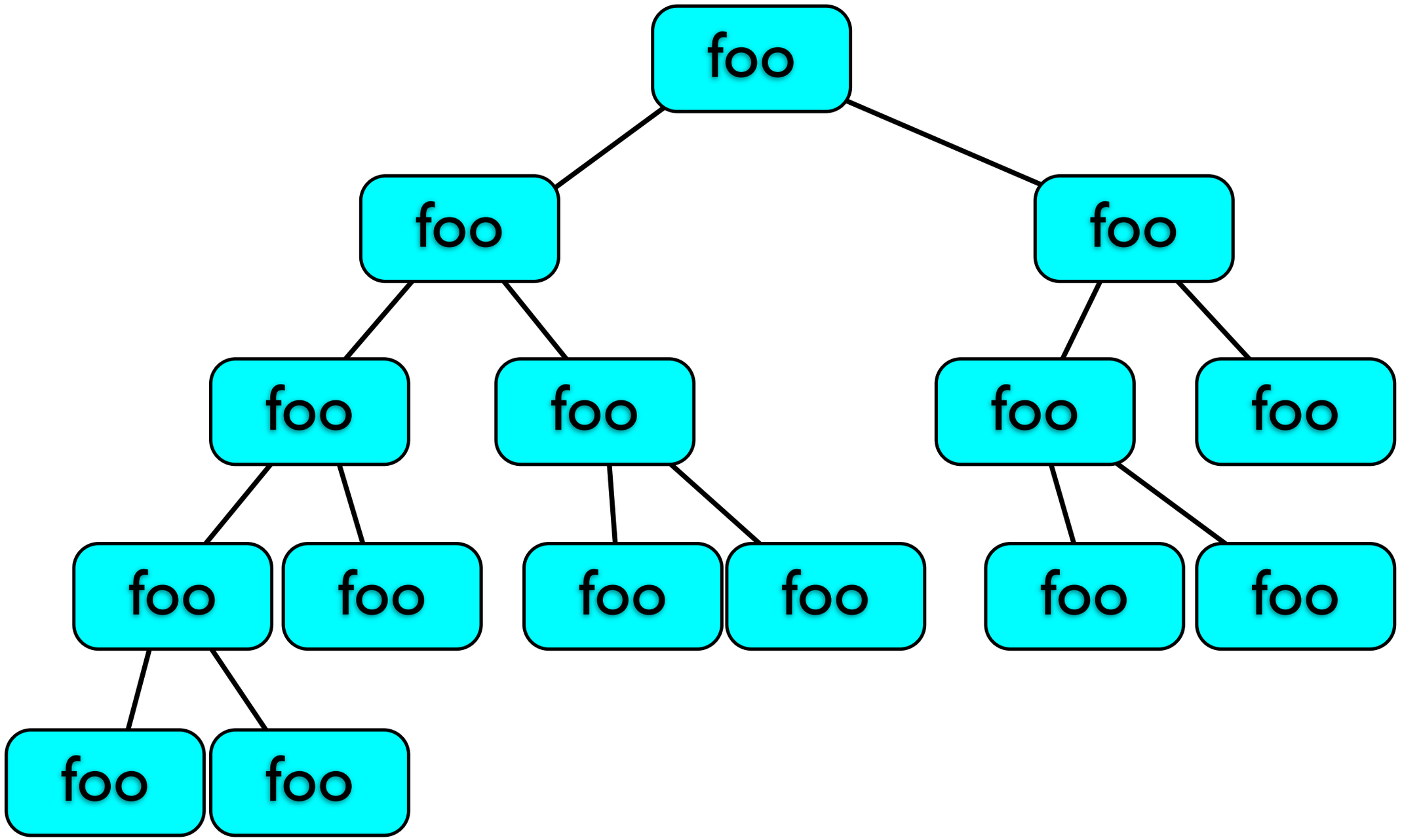
foo(47) = -1323752223 (mod 2^32)

What's wrong

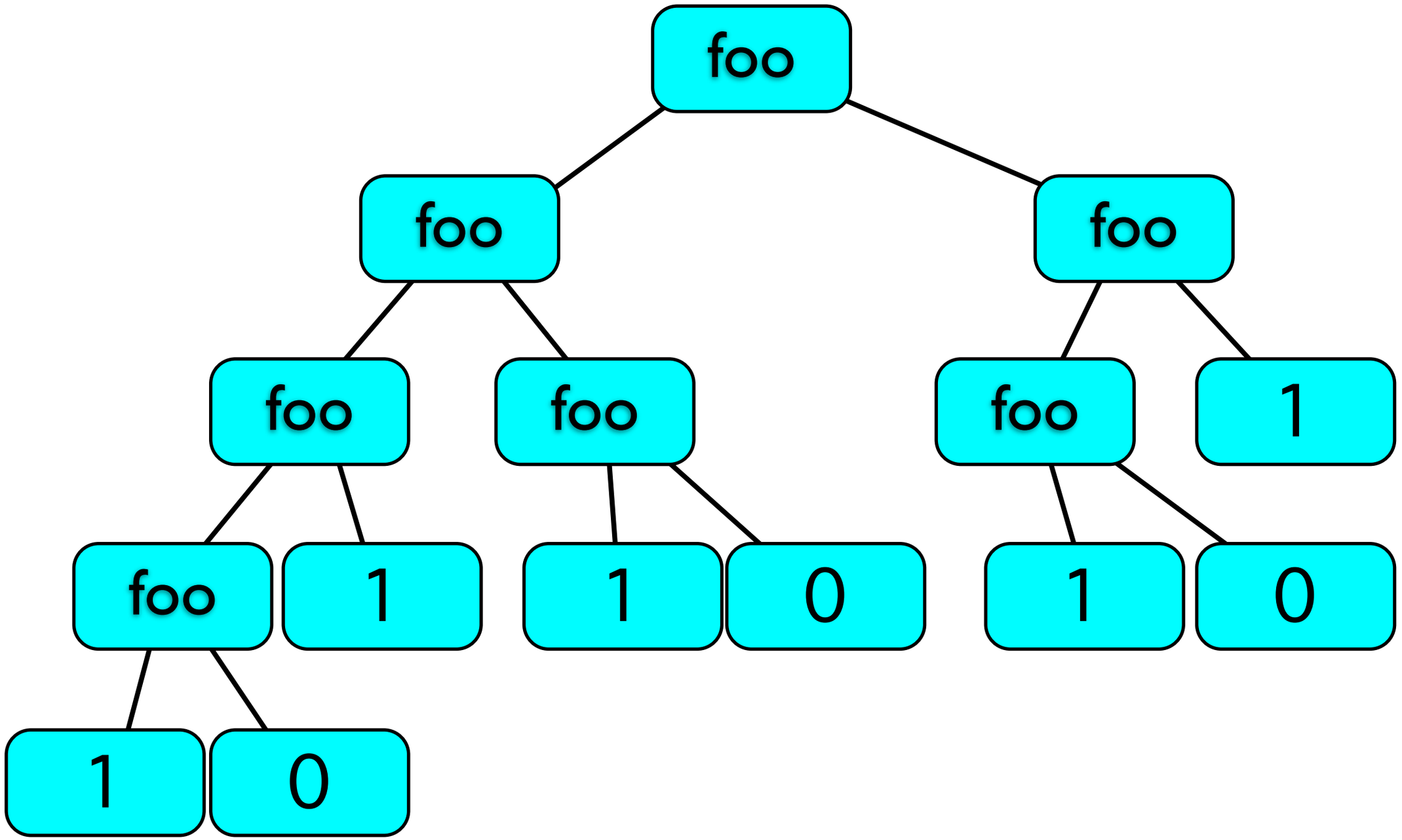
```
int foo(int n) {
   if (n<=1) return n; //else
   return foo(n-1)+foo(n-2);
}
```

- Values quickly exceed max_int
- T(n) is proportional to foo(n), which grows fast. Why? Unroll the recursion.

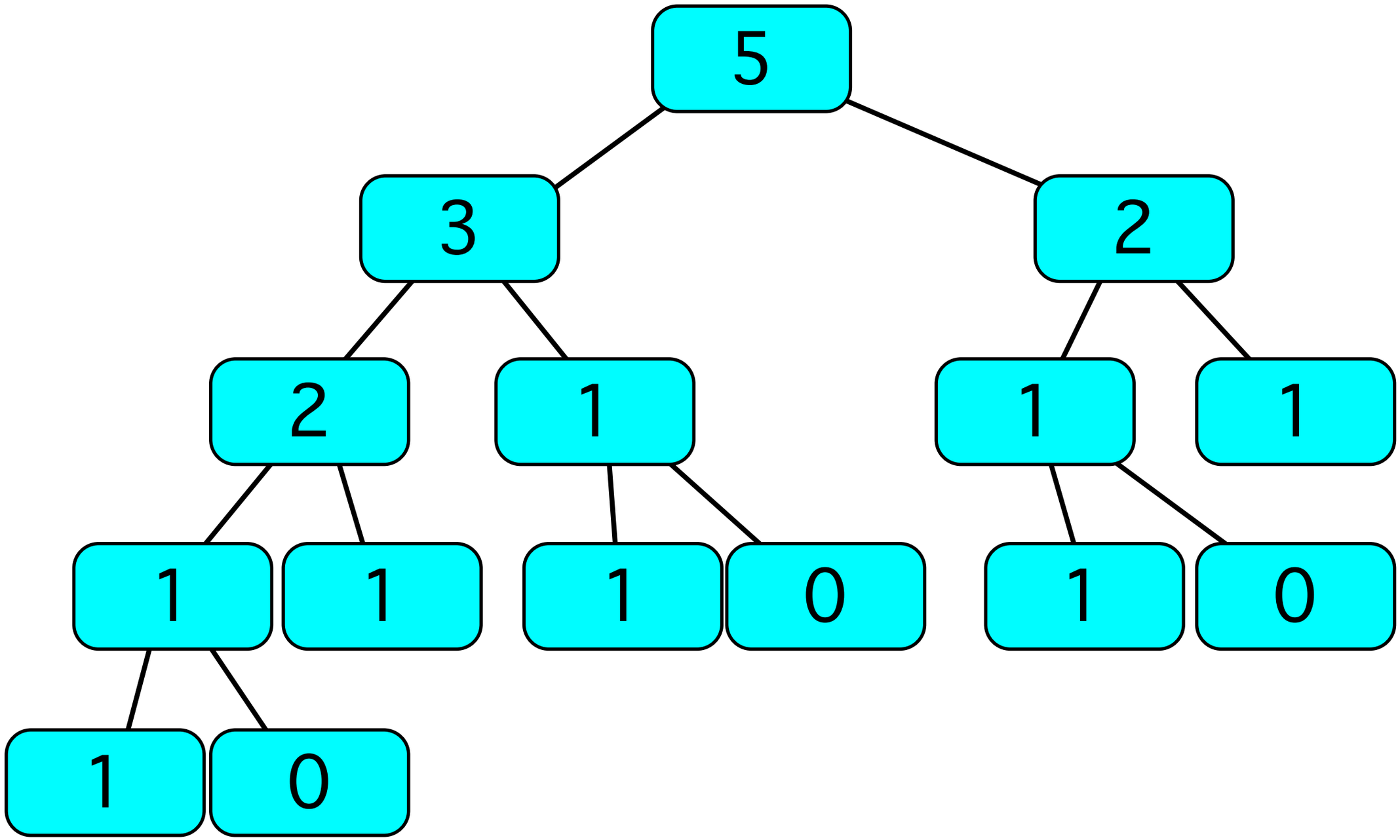
The recursion, unrolled



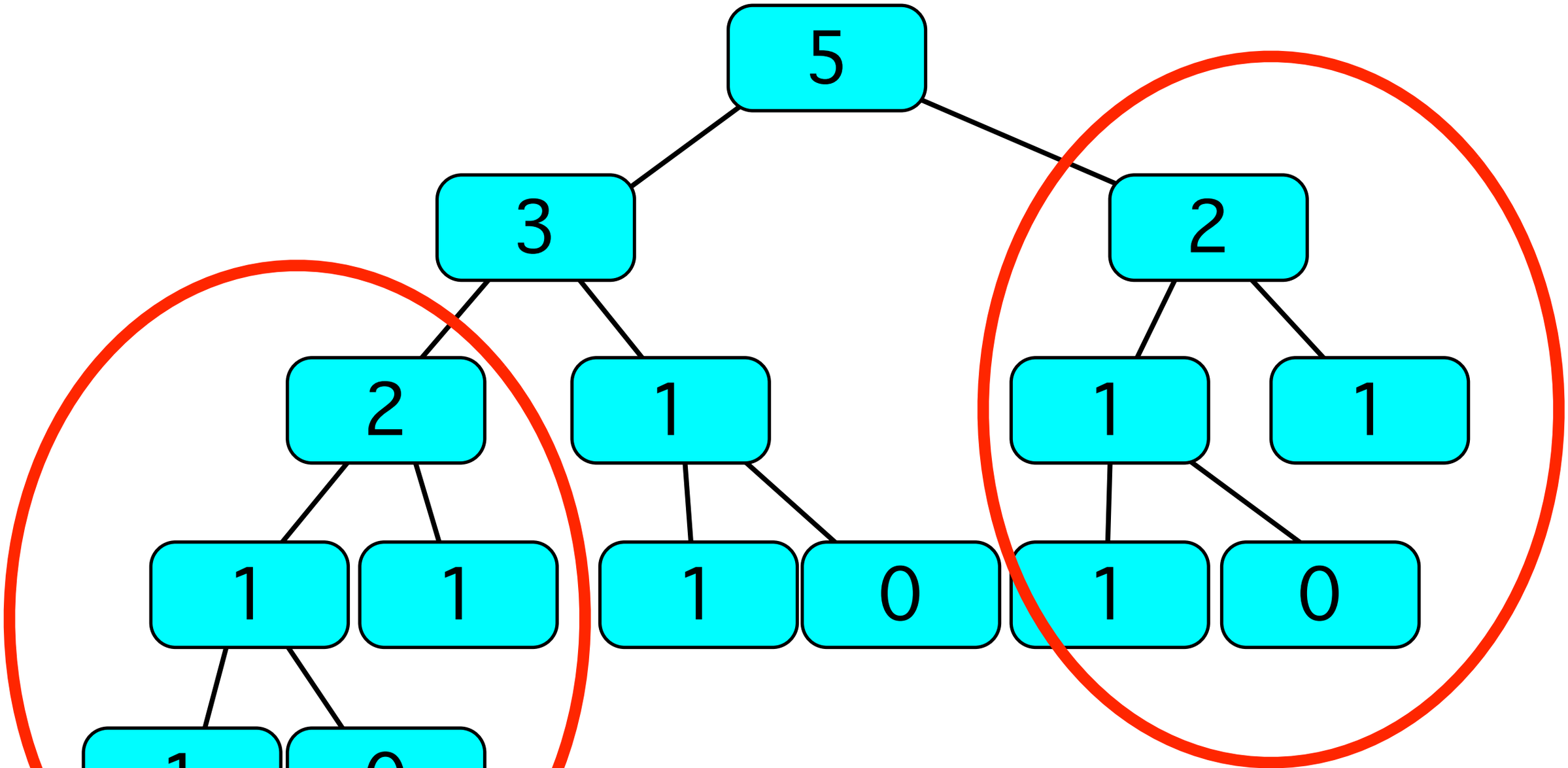
The recursion, unrolled



The recursion, unrolled

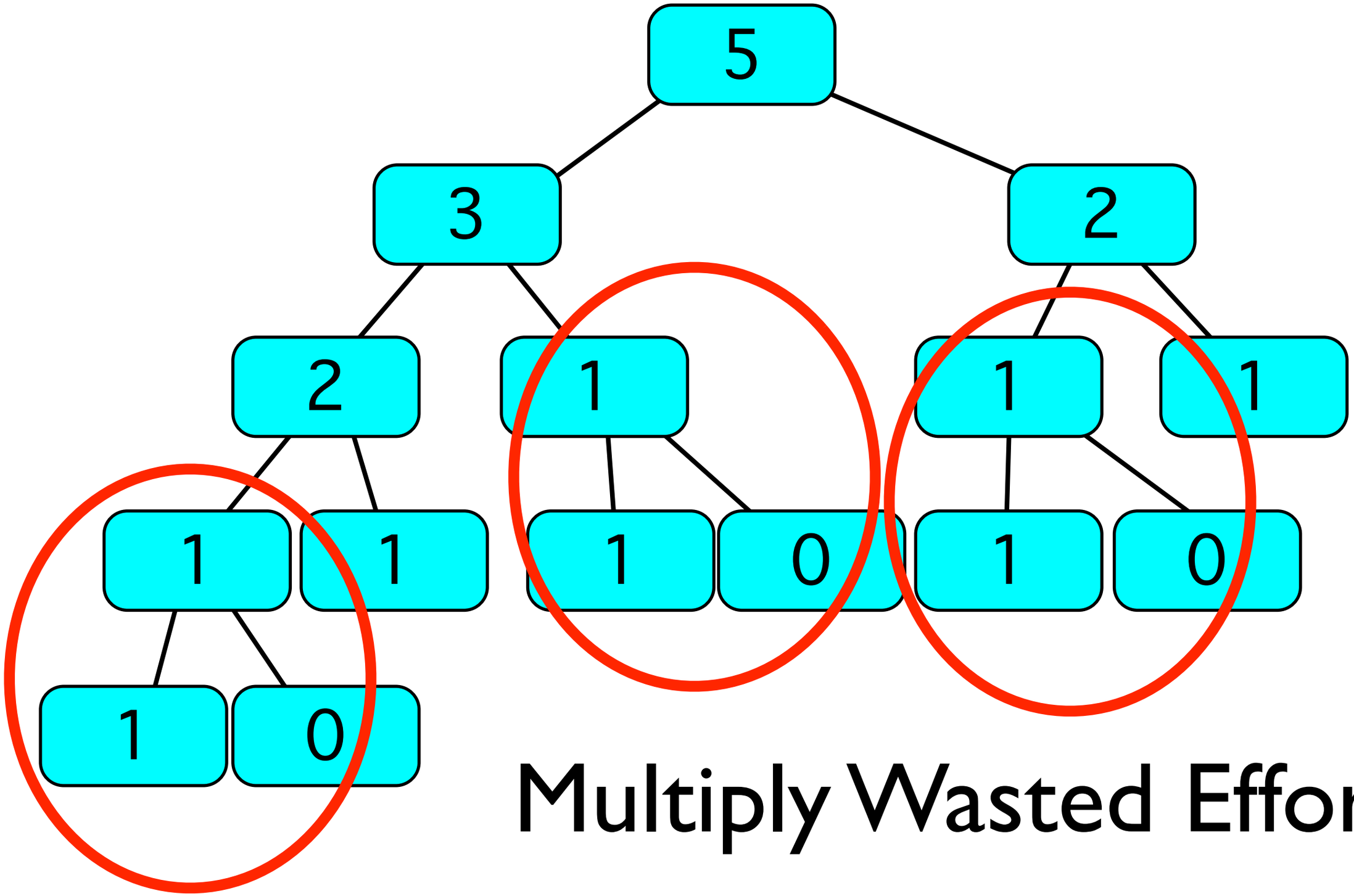


The recursion, unrolled



Duplication of Effort!

The recursion, unrolled



Multiply Wasted Effort!

The recursion, unrolled

- A lower bound on run-time: one clock cycle per node in this tree.
- A lower bound on this: one clock cycle per leaf of this tree.
- A lower bound on this: just count the leaves that have a value of ONE.
- There are foo(n) of these. Why? foo(n) is the sum of these ONEs. Thus $T(n) \geq foo(n)$

Fix: use memory!

```
int foo(int n) {
  static int answer[1000] = {0,1};
  static int top=1;
  if (n>top) {
    answer[n] = foo(n-1)+foo(n-2);
    top = n;
  }
return answer[n];
}
```

(Also, use BigInteger to avoid overflow!)

“Memoizing”

- For proper functions (no side effects)
- Use static array to keep a record of previously computed values.
- Only compute new values when needed--don’t forget to store them!
- Just look up previously computed values

That example was in C

```
int foo(int n) {
  static int answer[1000] = {0,1};
  static int top=1;
  if (n>top) {
    answer[n] = foo(n-1)+foo(n-2);
    top = n;
  }
return answer[n];
}
```

(Also, use BigInteger to avoid overflow!)

That example was in C

Danger!

```
int foo(int n) {
   static int answer[1000] = {0,1};
   static int top=1;
   if (n>top) {
      answer[n] = foo(n-1)+foo(n-2);
      top = n;
   }
return answer[n];
}
```

Annoying

(Also, use BigInteger to avoid overflow!)

Here it is in Java

```
ArrayList<BigInteger> answer=
        new ArrayList<BigInteger>();
answer.add(BigInteger.ZERO);
answer.add(BigInteger.ONE);
int foo(int n) {
  if (n>A.size())
    A.add(foo(n-1).add(foo(n-2)));
  return A.get(n);
  }
```

Win: Safety, Cleaner Code

Why? Better Data Structures!

Data Structures

BigInteger: won't silently overflow. Will throw exception only if you run out of memory.

ArrayList<BigInteger>: dynamically resized array. No need to guess the size or explicitly track it.

No C++-style operator overloading ("syntactic sugar") (oh, well!)

Sorting

Input: Sequence of things that can be ordered

21, 3, 8, 6, 59 -40, 0

Bob, Carol, David, Alice, Aaron

Output: The input, but actually in order

-40, 0, 3, 6 ,8, 21, 59

Aaron, Alice, Bob, Carol, David

Comparable

A built-in Java Interface (with generic type) that supports the notion of Total Ordering.

(look at the API)

Why use this?

Abstraction and Code Re-use

embodies a Fundamental Concept

See also: Comparator

Relations

Let S be a set.

Binary Relation on S:

(a) particular set of pairs (x,y) where x and y are elements of S.

(b) If (x,y) in relation R, write "x R y"

"Total Order" Relation: transitive, antisymmetric, total. Often denoted "<"

Order Relations

Let S be a set.

Let < be a Binary Relation on S:

Transitive: If $a < b$ and $b < c$ then $a < c$

Antisymmetric: At most one of ($a<b$), ($b<a$) can be true.

We say the order relation is TOTAL if for every $a \neq b$, $a<b$ or $b<a$. Otherwise, partial.

Order Relations

Example of a partial order:

S = {strings}. a "is a prefix of" b.

Example: tom is a prefix of tomato

mat, oat are not prefixes of tomato.

Transitive: check.

Antisymmetric: check.

Total? No: tomato, potato incomparable

Order Relations

Examples of total orders:

S = {strings}. Dictionary order (aka lexicographical order).

an < angel < apricot < baa < baal < turkey

S = {real numbers}. "less than"

Any subset of the above S with the same relation

Sorting Algorithms

Examples (3 of many):

MergeSort (split in half, recursively sort each half, then merge results)

QuickSort (split into low and high "halves" using a "pivot element", recursively sort each. No merge needed)

BubbleSort (keep swapping neighboring pairs that are out of order)

Sorting: Performance

Run-time (in clock cycles) is extrinsic to these algorithms.

Intrinsic: count the number of basic operations done.

These include: comparing list elements, comparing loop variable to end cond, managing loop variables, swapping elts, data structure internal costs, subroutine calls

Sorting: Performance

Run-time (in clock cycles) is extrinsic to these algorithms.

Intrinsic: count the number of basic operations done.

Focus: just comparisons of list items

Let $Cost_A(x)$ = # comparisons done by sorting algorithm A on input x

Only depends on the order of the list x

Sorting: Performance

Let $Cost_A(x)$ = # comparisons done by sorting algorithm A on input x

Only depends on the order of the list x

Still not good enough--I want to know how many steps to sort a list of length 88
Answer: it depends

"Worst-case analysis": Let $Cost_A(n)$ = max # comparisons by A, over all x of size n

Sorting: Performance

Let $Cost_A(x)$ = # comparisons done by sorting algorithm A on input x

Only depends on the order of the list x

Still not good enough--I want to know how many steps to sort a list of length 88
Answer: it depends

"Worst-case analysis": Let $Cost_A(n)$ = max # comparisons by A, over all x of size n

Hurray! We can graph it!

Comparing Graphs

Suppose I have two functions:

f(n) = max #ops for Alg A over inputsize n

g(n) = max #ops for Alg B over inputsize n

(go to whiteboard)

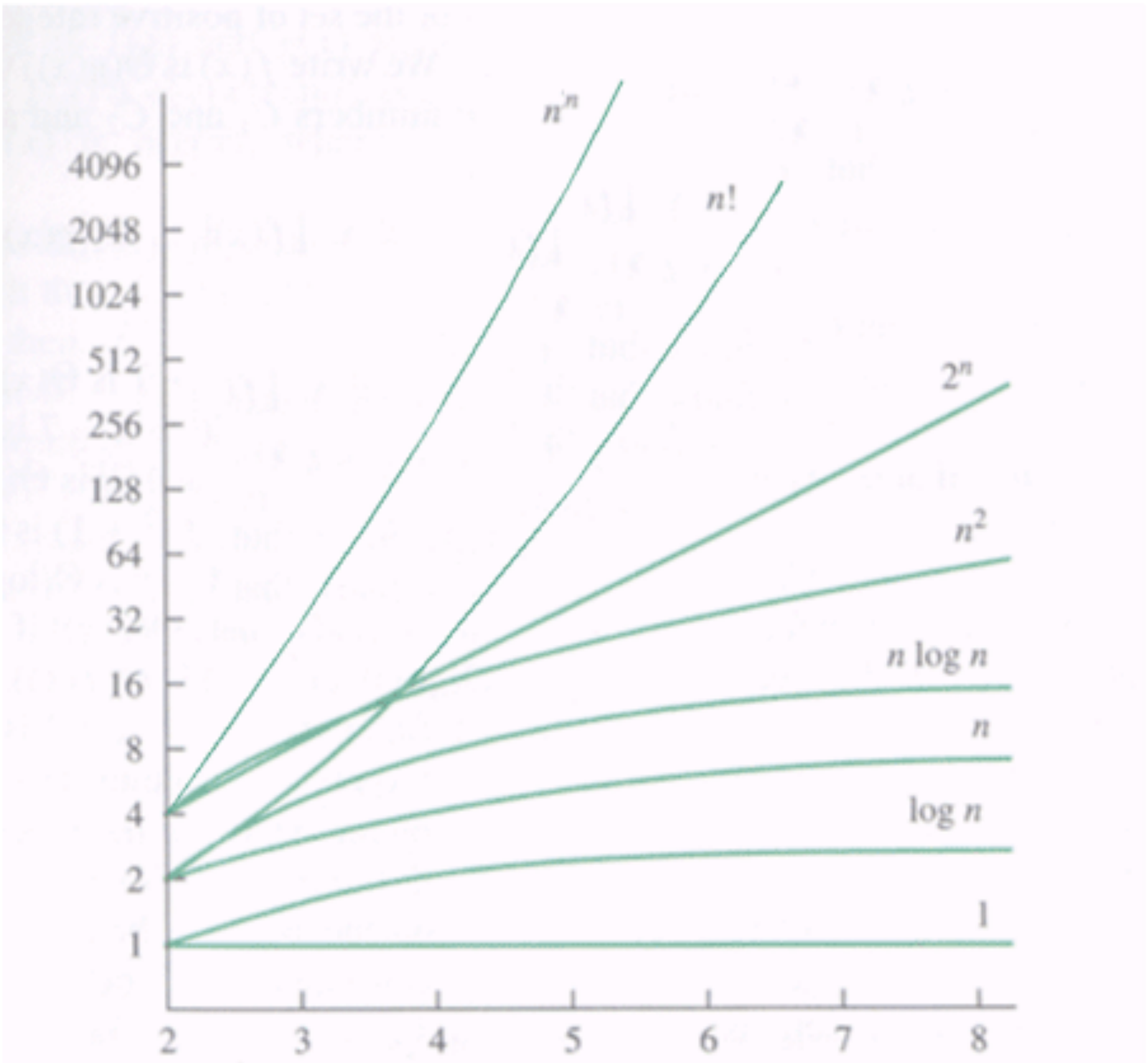
How do we compare them? Which alg has a better guarantee?

Scalability: don't focus on small n. What is the trend for large n?

Fin

In class, we stopped after the previous slide.

There are a few more slides beyond, that we will go through at the beginning of Lecture 2.


n^n
$n!$
2^n
n^2
$n \log n$
n
$\log n$
1
4096
2048
1024
512
256
128
64
32
16
8
4
2
1
2
3
4
5
6
7
8

little-oh notation

We say f = o(g) "f is little-oh of g" if

f(n)/g(n) ---> 0 as n grows to infinity.

Said another way: for every $\varepsilon > 0$, there exists an n_0 such that for all $n \geq n_0$, $f(n) \leq \varepsilon\, g(n)$

little-oh notation

We say f = o(g) "f is little-oh of g" if

f(n)/g(n) ---> 0 as n grows to infinity.

Said another way: for every $\varepsilon > 0$, there exists an n_0 such that for all $n \geq n_0$, f (n) $\leq \varepsilon$ g(n)

little-oh is a partial order relation

little-oh notation

We say f = o(g) "f is little-oh of g" if

f(n)/g(n) ---> 0 as n grows to infinity.

Said another way: for every $\varepsilon > 0$, there exists an n_0 such that for all $n \geq n_0$, f(n) $\leq \varepsilon$ g(n)

little-oh is a partial order relation

Note: It's a letter o, not a zero!

Properties of little-oh

If $f = o(g)$ and $g = o(h)$ then $f = o(h)$.

If $f = o(g)$ and $f' = o(g')$ then the product $f \cdot f' = o(g \cdot g')$. Also, the sum $f + f' = o(g + g')$.

Why? (prove on whiteboard)

Don't tell students: perfect QUIZ question!

Warning! Take heed!

We say f = o(g) "f is little-oh of g" if

f(n)/g(n) ---> 0 as n grows to infinity.

The notation "=" above is a "deliberate abuse." You cannot combine it with properties of equality. For instance,

$n^2 = o(n^4)$ and $n^3 = o(n^4)$ but $n^2 \neq n^3$

$n^2 = o(n^3)$ and $n^2 = o(n^4)$ but $o(n^3) \neq o(n^4)$

Also, never write: o(g) = f

Asymptotic analysis

$T_A(n)$ = worst-case running time for alg A on inputs of size n. We don't know a formula for this function.

Goal: Find an actual formula F(n) such that $T_A(n) < F(n)$.

Asymptotic analysis: we're satisfied if result holds for all $n>n_0$ (for some fixed n_0)

Back to Sorting

$T_B(n)$ = worst-case #comparisons for BubbleSort on list of length n.

Claim: $T_B(n) < n^2$

Why? Claim: we only compare any two elements at most once. Since there are (n choose 2) = n (n-1) / 2 < n^2 pairs of elements, this bounds the total number of comparisons.

Why? Each swap only changes the relative

Back to Sorting

Claim: we only compare any two elements at most once.

Why? Each swap only changes the relative positions of the two elements we just compared. So, after each comparison, we put that pair in the right relative order, and later swaps will never undo this.

Reminders: To Do!

- All students: email me, hayes@unm.edu Tell me: your name, email (that you will check), recent CS and math courses, registered or not?, major/department, year/grad/undergrad, can attend office hours? graduating senior?
- Start on the reading and written assignments!
- I will be out of town next week, so come to this week's office hours!