I just wrote the most beautiful code of my life.

They casually handed me an impossible problem. In 48 hours and 200 lines, I solved it.

Academia:

My god ... this will mean a half-dozen papers, a thesis or two, and a paragraph in every textbook on queuing theory!

Business:

You got the program to stop jamming up? Great. While you're fixing stuff, can you get Outlook to sync with our new phones?
From last lab

Example of *data abstraction*

- Makes your code easier to read
- Reduces errors
- Makes your code easier to change
From last lab

Example of *data abstraction*

- Makes your code easier to read
- Reduces errors
- Makes your code easier to change

Next language will have some features that make data abstraction easier.

Sometimes called *structs* or *records* in programming languages.
Searching a binary search tree

(define tree-search
  (lambda (T k)…))

Strategy:
• Compare $k$ to key at root of tree
• If $k$ is less, recursively search left subtree
• If $k$ is more, recursively search right subtree

How to analyze?
Not in terms of size of input because this procedure does not have to look at the whole input.
Instead, analyze by depth of subtree.
Depth of tree is 3

Depth of node Is 1

Subtree has depth 2

Depth of node is 3
Depth of tree is 3

Depth of node is 1

Subtree has depth 2

Subtree has depth 1

Depth of node is 3
Let $f(d)$ be the worst-case time for tree-search on trees of depth $d$.

\[
f(0) \leq a \\
f(d) \leq b + f(d - 1)
\]
for $d > 0$

Solution:

\[
f(d) \leq a + bd
\]

How deep is a BST?
How deep is a BST with $n$ items?

Let $g(n)$ be the worst-case depth of a BST with $n$ items

$$
g(1) = 0$$

$$
g(n) \leq 1 + g(n/2) \quad \text{for } n > 1$$

Solution:

$$
g(n) \leq \log_2 n$$

Claim: For $n \geq 1$, $g(n) \leq \log_2 n$

Proof: By induction on $n$.

Base case: $n=1$. $g(0) = 0 = \log_2 1$.

Induction step: $n > 1$. $g(n) \leq 1 + g([ (n - 1)/2 ])$. By induction hypothesis,

$$
g([ (n - 1)/2 ]) \leq \log_2 [(n - 1)/2] \leq \log_2 n/2 \leq \log_2 n - 1$$

So $g(n) \leq 1 + g([ (n - 1)/2 ]) \leq 1 + \log_2 n - 1 = \log_2 n$ QED
Bottom-line: searching a BST with $n$ items takes $O(\log n)$ time.

Think of $\log n$ as “small but not constant.”

In terms of big $O$,
the function $\log n$ is $O(n)$, $O(n^{1/2})$, $O(n^c)$ for any $c > 0$.
Lexical scope, first look

Body of a lambda expression can use a top-level variable.

> (define increment 7)
> ((lambda (x) (+ x increment)) 10)
17

> (define add-increment
   (lambda (x) (+ x increment)))
> (add-increment 10)
17

Body of a lambda expression can use a formal argument.

> (define add-to-all
   (lambda (increment L)
     (map (lambda (x) (+ x increment)) L)))
> (add-to-all 7 '(10 20 30))
(17 27 37)

Variables available: those in lexical scope when lambda expression evaluated
Body of a lambda expression can use a formal argument.

> (define add-to-all
   (lambda (increment L)
     (map (lambda (x) (+ x increment)) L)))
> (add-to-all 7 '(10 20 30))
(17 27 37)

Variables available: those in lexical scope when lambda expression evaluated

One procedure can return a second procedure whose body uses as a variable a formal argument of first procedure.

> (define adder
   (lambda (increment)
     (lambda (x) (+ x increment))))
> (map (adder 7) '(10 20 30))
(17 27 37)
#lang racket
(define increment 7)
((lambda (x) (+ x increment)) 10)
(define add-increment
  (lambda (x) (+ x increment)))
(define add-to-all
  (lambda (increment L)
    (map (lambda (x) (+ x increment)) L)))

Welcome to DrRacket, version 7.0 [3m].
Language: racket, with debugging [custom]; memory limit: 128 MB.
17
> (define adder
  (lambda (increment)
    (lambda (x) (+ x increment))))
> (map (adder 7) '(10 20 30))
(17 27 37)
#lang racket
(define increment 7)
((lambda (x) (+ x increment)) 10)
(define add-increment
  (lambda (x) (+ x increment)))

(define add-to-all
  (lambda (increment L)
    (map (lambda (x) (+ x increment)) L)))

Welcome to DrRacket, version 7.0 [3m].
Language: racket, with debugging [custom]; memory limit: 128 MB.
17
> (define adder
    (lambda (increment)
      (lambda (x) (+ x increment))))
> (map (adder 7) '(10 20 30))
(17 27 37)
**Enumeration**

Generating a set of all combinatorial objects of a certain kind.

**Example:** permutations (orderings)

Consider the list \((a \ b \ c)\)

Reordering the elements, can get also
\[(a \ c \ b), (b \ a \ c), (b \ c \ a), (c \ a \ b), (c \ b \ a)\]

Total number of permutations (including original), there are 3!

We will write a **procedure** permutations to generate a list of all permutations of a given list of data objects.

\[
\text{(permutations ~(a b c))} \rightarrow
\]

\[
((a \ b \ c) \ (a \ c \ b) \ (b \ a \ c) \ (b \ c \ a) \ (c \ a \ b) \ (c \ b \ a))
\]

**“Quiz” (but in pairs):** Write a recursion diagram, and then come up with a strategy (not code).