CIS 313:
Intermediate Data Structure

first slide

Programs = Algorithms + Data Structures
(by Niklaus Wirth)

- From the book
  - Algorithm: any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.
  - Data structure: a way to store and organize data in order to facilitate access and modifications.

themes

- computational complexity, start to measure it
- simple data structures (mostly review)
- tree based structures
  - binary trees
  - binary heaps, binomial heaps
  - self adjusting trees: AVL, Red-Black
  - (2,4) trees, B-trees
- sorting, order statistics, voting

First algorithm

find the maximum number in an array

Input: a sequence of numbers a_1, a_2, …, a_n

Output: the maximum number in the input sequence

Algorithm:

```
max = a_1
for i = 2 to n:
    if a_i > max:
        max = a_i
return max
```

How long does this take?

Maybe: *n* variable assignments, *n*-1 comparisons, *n*-2 increments, one return?

how do we talk about algorithm speed?

- use functions of the size of the input *n* (typically the number of input numbers/items in this class), i.e., T(*n*)
- apply asymptotic notation for these functions
- it ignores constants and only focuses on the highest-order term
    - why? machine independence, constants not important *asymptotically*
    - asymptotically = "in the long run or in the limit"
- see description and definitions in text (section 3.1, pp 43-52)
- O, Ω, Θ, o, ω

Time spent at 1,000,000 operations per second:

input size

algorithm speed

	10	20	30	40	50	60	...	100
n	10^{-5} seconds	$2\cdot 10^{-5}$ seconds	$3\cdot 10^{-5}$ seconds	$4\cdot 10^{-5}$ seconds	$5\cdot 10^{-5}$ seconds	$6\cdot 10^{-5}$ seconds		10^{-4} seconds
n^2	10^{-4} seconds	$4\cdot 10^{-4}$ seconds	$9\cdot 10^{-4}$ seconds	$1.6\cdot 10^{-3}$ seconds	$2.5\cdot 10^{-3}$ seconds	$3.6\cdot 10^{-3}$ seconds		.01 second
n^3	10^{-3} seconds	$8\cdot 10^{-3}$ seconds	$2.7\cdot 10^{-3}$ seconds	$6.4\cdot 10^{-2}$ seconds	.125 second	.216 second		1 second
n^{10}	2.7 hours	118 days	18 years	333 years	3,103 years	19,213 years		31,775 centuries
2^n	10^{-3} seconds	1 second	17 minutes	12 days	35.7 years	36,634 years		$4\cdot 10^{14}$ centuries
3^n	.06 second	58 minutes	6.5 years	3863 centuries	$2\cdot 10^{8}$ centuries	$1.3\cdot 10^{13}$ centuries		$1.6\cdot 10^{32}$ centuries
$n!$	3.6 seconds	773 centuries	$8\cdot 10^{16}$ centuries	$2.6\cdot 10^{32}$ centuries	$9.7\cdot 10^{48}$ centuries	$2.6\cdot 10^{66}$ centuries		$3\cdot 10^{142}$ centuries
$2^{2^\wedge n}$	$>10^{292}$ centuries	$>10^{315637}$ centuries	ouch! →					

big-Oh formally

f(n) =O(g(n)) if and only if (iff)

$$\exists\, c > 0\; \exists\, N\; \forall\, n \geq N \quad 0 \leq f(n) \leq c \cdot g(n)$$

- *c* is the dropped constant
- *N* is the crossover point so that …
- … if *n* is big enough *f* is bounded above by *c*g*
- the growth rate of *g* bounds the growth rate of *f* from above

example: let f(n) = $3n^3 + 5n^2 + n + 17$

some true statements:

- f(n) = $O(n^3)$
- f(n) = $O(n^4)$
- f(n) = $O(17\ n^3)$
- f(n) = $3n^3 + O(n^2)$

Big Omega and Theta

$f(n) = \Omega\big(g(n)\big)$ iff
$\exists\, c > 0\ \exists\, N\ \forall\, n \geq N\ \ f(n) \geq c \cdot g(n) \geq 0$

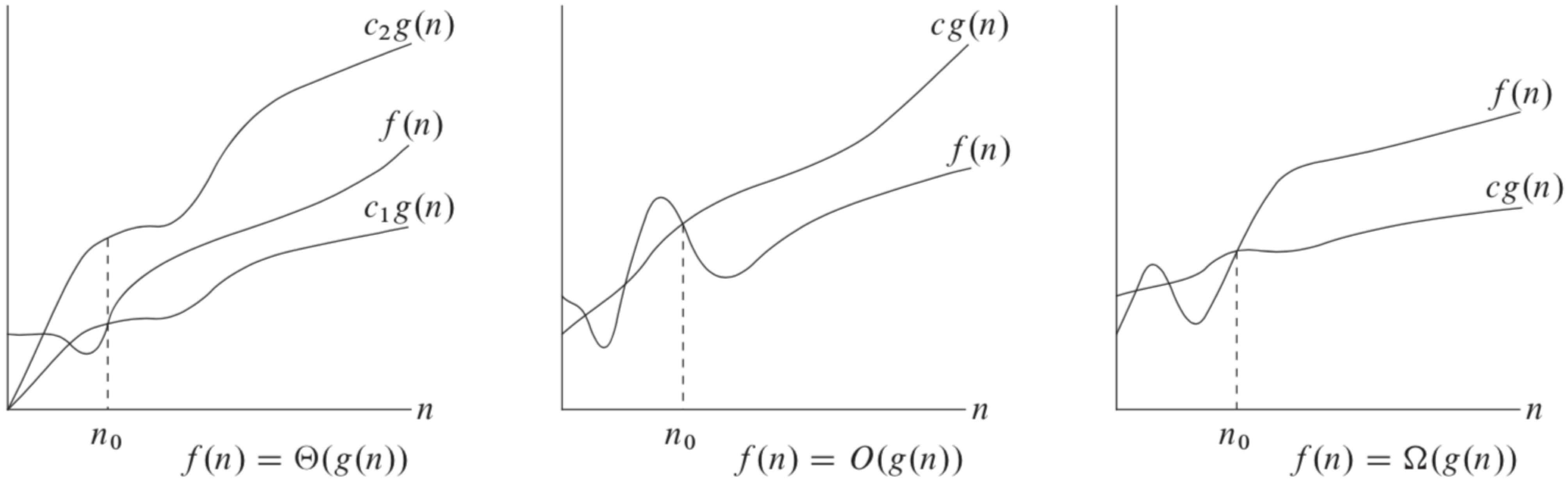
thus, the growth rate of *g* is less than or equal to the growth rate of *f* (ignoring the constant)

now we can say (f(n) = 3n^3 + 5n^2 + n + 17)

- $f(n) = \Omega(n^3)$
- $f(n) = \Omega(n^2)$
- $f(n) = \Theta(n^3)$
- $f(n) = 3 \cdot n^3 + \Theta(n^2)$

$f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

- here *f* and *g* have the <u>same</u> growth rate
- sort of like saying $A \leq B$ and $A \geq B$ implies that $A = B$



c₂g(n)
f(n)
c₁g(n)
n
n₀
f(n) = Θ(g(n))
cg(n)
f(n)
n
n₀
f(n) = O(g(n))
f(n)
cg(n)
n
n₀
f(n) = Ω(g(n))

little-oh and little-omega

$f(n) = o(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

or

$$\forall\, c > 0\ \exists\, N\ \forall\, n \geq N\ 0 \ \leq\ f(n) \leq c \cdot g(n)$$

in other words, the growth rate of f is *strictly less* than that of g

$f(n) = \omega(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

or

$$\forall\, c > 0\ \exists\, N\ \forall\, n \geq N\ \ f(n) \geq c \cdot g(n) \geq 0$$

the growth rate of f is *strictly greater* than that of g

examples:

- $f(n) = o(n^4)$
- $f(n) = \omega(n^2)$
- $f(n) = 3 \cdot n^3 + o(n^3)$
- $\frac{1}{n} = o(1)$

some properties

-Transitivity:

$f(n) = \alpha(g(n))$ and $g(n) = \alpha(h(n))$ imply $f(n) = \alpha(h(n))$ ($\alpha \in \{O, \Omega, \Theta, o, \omega\}$)

- Reflexivity:

$f(n) = \alpha(f(n))$ ($\alpha \in \{O, \Omega, \Theta\}$)

- Symmetry:

$f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$

- Transpose Symmetry:

$f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$

$f(n) = o(g(n))$ iff $g(n) = \omega(f(n))$

common functions

- n^k , where k is a constant (polynomial)
- 2^n, 3^n , c^n (exponential)
- $\log_2 n$, $\log_c n$, ln n (logarithmic – usually *log* n implies base 2)
  - fact: $\log_2 n = O(\log_c n)$ (why?)
- O(n *log* n) (also poly, but very common)
- n! (factorial)
- $2^{(\log n)^2}$ (super-poly, sub-exponential) (ok, not so common)

other functions

- factorials: $n! = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1$
- Stirling's Approximation: $n! = \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \cdot (1 + \Theta\left(\frac{1}{n}\right))$
- importantly $\log n! = \Theta(n \cdot \log n)$
- binomial coefficients
- Fibonacci sequence: $F_0 = 0, F_1 = 1, F_{k+2} = F_{k+1} + F_k$
- (Fibonacci used for AVL trees)

more examples

$10 \log n + \log\log n$ is $O(\log n)$? $O(n)$? $O(n^{0.0000001})$? $\Omega(\log n)$? $O((\log n)^{0.5})$? $\Omega((\log n)^{0.5})$

$2^{3^{2000}}$ is $O(1)$? $\Omega(1)$? $2^{3^{2000}}\, n$ is $O(n)$?

$2/n$ is $O(1/n)$? $O(1/\sqrt{n})$? $O(1/n^{1.7})$? $O(1)$?

$$f(n) = \begin{cases} 0.1\, n \text{ if } n \text{ is odd} \\ 3\, n^2 \text{ if } n \text{ is even} \end{cases}$$ is $O(n)$? $O(n^{1.5})$? $O(n^2)$? $\Omega(n)$? $\Omega(n^{1.5})$ $\Omega(n^2)$

Exercise

Order the following by growth rate (big-Theta). Start on your own:

n

$n^2 – 4n$

$n^2 + n (\log n)^3$

$n^{5/2} + n^{3/2} + 100 \log n$

$n + \log n$

$(\log n)(n + n^2)$

$n^2 \log n + n (\log n)^3$

$2^{\log n}$

$2^n \log n$

$1/n$

$1/(n \log n)$

$n^{1/2} + n \log n$

$n + n \log n$

$(\log n)^3 + (\log n)^2 + \log n$

$n^2 \log n + n (\log n)^3$

$2^{n \log n}$

reading for previous material

- chapter 3
- appendix A.1

loop invariants

- “simple” method to prove correctness of a loop structure
- follows induction
- three phases: *initialization* (base case),
- *invariance maintenance* (induction), and
- *termination*

- look at pp 18-20 of text for more discussion
- while there, look at pp 20-22 for description of pseudo-code

general structure of argument

code:

<init>
while γ
  do $\mathcal{L}$

invariant: α
a true/false statement about the variables of the code

initialization: show that α is true after the <init> phase of the code has been executed

maintenance: show that if $\alpha \wedge \gamma$ is true, then α will be true after one execution of the loop body $\mathcal{L}$

termination: the loop finishes when γ is false, so argue that $\neg\gamma \wedge \alpha$ is the desired outcome

example

```
input: integer n>0
output: n(n+1)/2

--initialization
int s=0
int k=0

--loop
while k < n+1 do
    s = s+k
    k = k+1

--end
return s
```

γ: k < n+1

α:
- $0 \leq k \leq n+1$
- s = k(k-1)/2

example

```
input: integer n>0
output: integer k, array b of k bits

--initialization
int k=0
int t=n
array b=[] of bit

--loop
while t>0 do
    b[k] = (t mod 2)
    k = k+1
    t = t div 2

--end
return k, b
```

γ: t>0

α:

- $t \geq 0$
- Let $m = \sum_{i=0}^{k-1} b[i] \cdot 2^i$ be the number represented by b in base 2. Then $n = 2^k \cdot t + m$

notice:

- initialization is easy
- termination also easy
- see handout (posted on class site) for full discussion

example

Compute the n-th Fibonacci number