MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001 Structure and Interpretation of Computer Programs
Spring, 2007

Recitation 4, Friday February 16

Order of Growth Notes Dr. Kimberle Koile

Order of Growth: For a process with an input of size n, we want to characterize how much of some resource is required by the process as the input becomes larger.

Let n be a parameter that measures the size of the problem solved by the process
R(n) be the amount of resources needed for a problem of size n

R(n) has order of growth $\Theta(f(n))$ if there are positive constants k_1 and k_2 independent of n such that:

$k_1 f(n) \leq R(n) \leq k_2 f(n)$ for any sufficiently large value of n

R(n) is typically measured in terms of space (max size of expression) and time (number of steps) requirements. We will measure the size of an expression by number of deferred operations.

Typical Orders of Growth

$\Theta(1)$ (constant): The resource requirements do not change with the size of the problem. All of our linear iterative processes use constant space (e.g., iterative version of fact).

$\Theta(n)$ (linear): The resource requirements grow linearly with the size of the problem. (Multiplying the size of the problem multiplies the resource use by the same factor.) All of our linear iterative processes use a linear number of steps (e.g., iterative fact). All of our linear recursive processes use linear space and number of steps (e.g., recursive fact).

$\Theta(b^n)$ (exponential): The resource requirements grow exponentially with the size of the problem. (Incrementing the size of the problem multiplies the resource use by a constant factor.) Recursive fib requires an exponential number of steps (though linear space).

$\Theta(\log n)$ (logarithmic): The resource requirements grow logarithmically with the size of the problem. (Multiplying the size of the problem adds a constant amount to the resource use.) Fast-expt is logarithmic in both number of steps and space.

$\Theta(n^m)$ (power law): The resource requirements grow as a power of the size of the problem. (Multiplying the size of the problem by some factor multiplies the resource use by a power of that factor.) Linear growth is a special case of this (m = 1). Another common case is quadratic growth, $\Theta(n^2)$. The prime-testing procedure in the problems today is iterative and is an example of the power law.

Order of Growth Notes (cont'd)

Time: # of operations

Space: # of <u>deferred</u> operations

Examples

fact

operations determined by n = linear

```
(define (fact n)
 (if (= n 1)1(* n (fact (- n 1))))))
```

1 deferred op = linear in space

time	$\Theta(n)$
space	$\Theta(n)$

```
(fact 4)
(* 4 (fact 3))
(* 4 (* 3 (fact 2)))
(* 4 (* 3 (* 2 (fact 1))))
(* 4 (* 3 (* 2 1)))
(* 4 (* 3 2))
(* 4 6)
24
```

n used in 2 ways: as counter + as arg for multiply op

```
(define (ifact n)
   (iter 1 1 n))

(define (ifact-iter product counter max-count)
   (if (> counter max-count)
          product
         (ifact-iter (* counter product)
                (+ counter 1)
                max-count)))
```

time	$\Theta(n)$
space	$\Theta(1)$

ops determined by n (aka counter) = linear time

no deferred ops = constant space

```
(ifact 4)
(ifact-iter 1 1 4)
(ifact-iter 1 2 4)
(ifact-iter 2 3 4)
(ifact-iter 6 4 4)
(ifact-iter 24 5 4)
24
```

expt

```
(define (expt b n))
  (if (- n 0)
       1
       (* b (expt b (- n 1)))))
```

time	$\Theta(n)$
space	$\Theta(n)$

```
(define (iexpt b n)
    (expt-iter b n 1))

(define (expt-iter b counter product)
    (if (= counter 0)
        product
        (expt-iter b
                (- counter 1)
                (* b product))))
```

time	$\Theta(n)$
space	$\Theta(1)$

fast-expt

Take advantage of: $b^n = (b^{n/2})^2$ if n is even, $b^n = b*b^{n-1}$ if n is odd.

e.g. $b^2 = b*b$

$b^4 = b^2 * b^2$

$b^8 = b^4 * b^4$

```
(define (fast-expt b n))
   (cond ((=  n 0) 1)
         ((even? n) (square (fast-expt b (/ n 2))))
         (else (* b (fast-expt b (- n 1))))))
```

1 deferred op; # halved with each call

steps halved with each call = log n time

this step only done once so doesn't contribute

time	$\Theta(\log n)$
space	$\Theta(\log n)$

fib

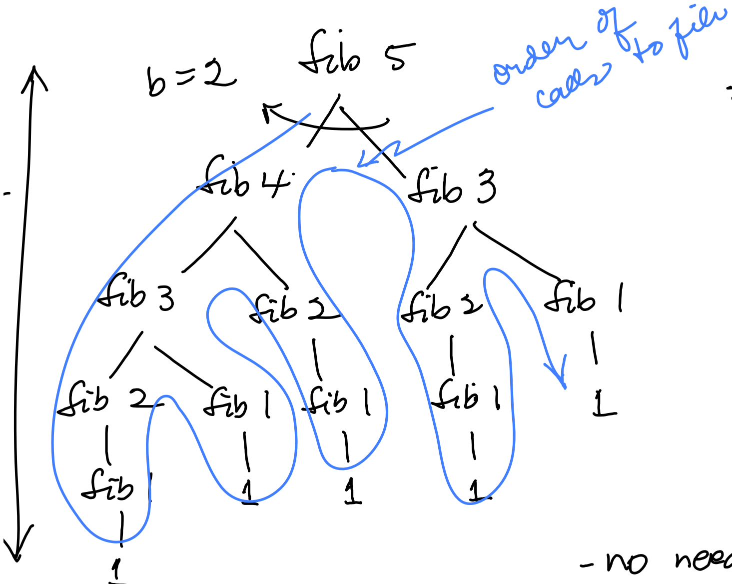
```
(define (fib n)
   (cond ((= n 0) 0)
         ((= n 1) 1)
         (else (+ (fib (- n 1))
                  (fib (- n 2))))))
```

each call adds 2 new calls = 2^n time *

1 deferred op = linear space

time	$\Theta(2^n)$
space	$\Theta(n)$

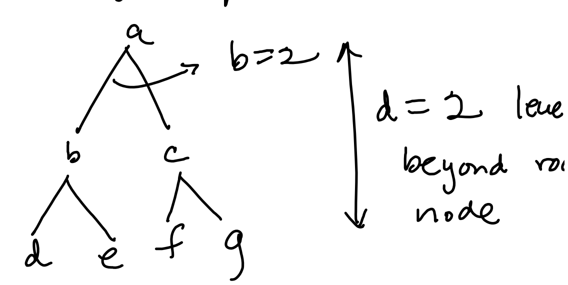
* 2^n = total # nodes in tree that represents calls:



nodes in a tree = b^d

where b = branching factor

d = depth



- no need to worry about unevenness of nodes at bottom of tree; want order of growth

ib :

time : # ops = # nodes = $b^d = 2^n$

space : 1 deferred op = linear (depth of tree because that's the