6.001 Recitation 4: Orders of Growth

RI/Substitute: Gerald Dalley, dalleyg@mit.edu
16 Feb 2007

Announcements / Notes

- No classes Monday. Tuesday is a virtual Monday. If you normally attend a Tuesday tutorial, try to attend any tutorial, but stick with your TA if at all possible.
- Project 1 is due on 2 March 2007. It's new! It's fun! It's cryptic!
- InstaQuiz discussion

Apocrypha

Kings, wheat, chessboards, orders of growth, and 18,446,744,073,709,551,615.

Definitions

Theta (Θ) notation:

$$f(n) = \Theta(g(n)) \rightarrow k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n), \text{for } n > n_0$$

Big-O notation:

$$f(n) = O(g(n)) \rightarrow f(n) \leq k \cdot g(n), \text{for } n > n_0$$

Adversarial approach: For you to show that $f(n) = \Theta(g(n))$, you pick k_1, k_2, and n_0, then I (the adversary) try to pick an n which doesn't satisfy $k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$.

Time order of growth: how many primitive operations are evaluated?

Space order of growth: maximum number of pending operations.

Implications

Ignore constants. Ignore lower order terms. For a sum, take the larger term. For a product, multiply the two terms. Orders of growth are concerned with how the effort scales up as the size of the problem increases, rather than an exact measure of the cost.

Typical Orders of Growth

- $\Theta(1)$ - Constant growth. Simple, non-looping, non-decomposable operations have constant growth.
- $\Theta(\log n)$ - Logarithmic growth. At each iteration, the problem size is scaled down by a constant amount: `(recur (/ n c))`.
- $\Theta(n)$ - Linear growth. At each iteration, the problem size is decremented by a constant amount: `(recur (- n c))`.
- $\Theta(n \log n)$ - Nifty growth. Nice recursive solution to normally $\Theta(n^2)$ problem.
- $\Theta(n^2)$ - Quadratic growth. Computing correspondence between a set of n things, or doing something of cost n to all n things both result in quadratic growth.
- $\Theta(2^n)$ - Exponential growth. Really bad. Searching all possibilities usually results in exponential growth.
  `(+ (recur (- n c1)) (recur (- n c2)))`.

What's n?

Order of growth is *always* in terms of the size of the problem. Without stating what the problem is, and what is considered primitive (what is being counted as a "unit of work" or "unit of space"), the order of growth doesn't have any meaning.

Problems

1. Give order notation for the following:

   (a) $5n^2 + n$ $\boxed{\Theta\left(n^2\right)}$

   (b) $\sqrt{n} + n$ $\boxed{\Theta\left(n\right)}$

   (c) $3^n n^2$ $\boxed{\Theta\left(3^n n^2\right)}$

2. Consider the following implementation of factorial:

```
(define (fact n)
  (if (= n 0)
      1
      (* n (fact (- n 1)))))
```

   Show the steps in the substitution model for (`fact5`). Only write out the steps which introduce a new recursive call or are a base case.

```
(fact 5)
((lambda (n) (if (= n 0) 1 (* n (fact (- n 1))))) 5) ; not required for answer
(if (= 5 0) 1 (* 5 (fact (- 5 1)))) ; not required for answer
(if #f 1 (* 5 (fact (- 5 1)))) ; not required for answer
(* 5 (fact (- 5 1))) ; not required for answer
(* 5 (fact 4))
(* 5 (* 4 (fact 3)))
(* 5 (* 4 (* 3 (fact 2))))
(* 5 (* 4 (* 3 (* 2 (fact 0)))))
(* 5 (* 4 (* 3 (* 2 1))))
(* 5 (* 4 (* 3 2)))
(* 5 (* 4 6))
(* 5 24)
120
```

   Running time? $\boxed{\Theta\left(n\right)}$ Space? $\boxed{\Theta\left(n\right)}$

3. Consider the following approximation to the constant $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + ...$

```
(define (find-e n)
  (if (= n 0)
      1.
      (+ (/ (fact n)) (find-e (- n 1)))))
```

   Running time? $\boxed{\Theta\left(n^2\right)}$ Space? $\boxed{\Theta\left(n\right)}$

4. Assume you have a procedure `(divisible? n x)` which returns `#t` if `n` is divisible by `x`. It runs in $O(n)$ time and $O(1)$ space. Write a procedure `prime?` which takes a number and returns `#t` if it's prime and `#f` otherwise. You'll want to use a helper procedure.

```
; Assume n is positive
(define (prime? n)
  (define (helper curr n)
    (cond ((>= curr n) #t)
          ((divisible? n curr) #f)
          (else (helper (+ 1 curr) n))))
  (helper 2 n))
```

```
; more clever given below...
(define (prime-fast? n)
  (define (helper curr)
    (cond ((> (* curr curr) n) #t)
          ((divisible? n curr) #f)
          (else (helper (+ 1 curr)))))
  (helper 2))
; Note: we could have checked (> curr (sqrt n)) instead
```

Running time? slow: $\Theta\left(n^2\right)$, clever: $\Theta\left(n\sqrt{n}\right)$ Space? both versions: $\Theta\left(1\right)$

InstaQuiz

Name:

1. Write a procedure that computes the number of decimal digits in it's input. Do not use logs.
   (num-digits 102) → 3

```
; Assumes n is non-negative
(define (num-digits n)
  (if (= n 0)
      0
      (+ 1 (num-digits (quotient n 10)))))

; Theta(n) time, Theta(n) space
```

2. Write a procedure that will multiply two numbers together, but the only arithmetic operation allowed is addition (*i.e.* multiplication through repeated addition). In addition, your procedure should be iterative, not recursive.
   (slow-mul 3 4) → 12

```
; Assumes a,b are non-negative
(define (slow-mul a b)
  (mul-helper a b 0))

(define (mul-helper a b total)
  (if (= a 0)
      total
      (mul-helper (- a 1) b (+ total b)))) ; or (+ a -1) if picky

; Theta(n) time, Theta(1) space
```

3. On Wednesday, we have a bonus recitation (since there's no lecture on Tuesday). By default, we'll keep diving into orders-of-growth questions. Is there anything else that you'd like included in Wednesday's recitation?