Lecture 5: Efficiency

Summer 2006
CS 61A
Instructor: Kevin Lin

Administrative stuff

- My office hours and TA office hours have been set -- see website.
- Use nova.cs.berkeley.edu to login remotely. Alternate servers are star.cs.berkeley.edu and solar.cs.berkeley.edu.
- Homework 1A and 1B due on Wednesday.
- Project 1 due on Thursday.
- No class tomorrow, July 4th.
- Please fill out the short course survey.

Plan

- Topics for today and this week:
  Scheme potpourri
  Efficiency
  Iteration vs. recursion
  Programming methodology
- Next week:
  Data abstraction
  Hierarchical data
- The week after next week, and beyond:
  Representing abstract data
  Object-oriented programming
  Meta-circular evaluator …
  Logic programming …

Scheme potpourri

```
; What gets evaluated and what doesn't?
(define temp (+ 4 5))
(define pi 3.1415926)
(define f (lambda (x) (+ x x)))

; What gets evaluated and what doesn't?
(define (foo) (+ 4 5))
(define (f x) (+ x x))
```


Scheme potpourri

```
; Some special forms we've seen so far:
quote
define
if
cond
lambda
let
and
or

; Can we ever define special forms ourselves?
; No. (Actually we can, but for the purposes
; of this class the answer is no.)
```


Scheme potpourri

What are some of the advantages of being able to use LAMBDAs in a programming language?

In a functional language, does it matter what order we do things in? That is, if we do things in a different order, will we get a different result?

No, it doesn't matter! Therefore, programs written in a functional language are *highly parallelizable.*

Scheme potpourri

Something cool about LAMBDAs is that in principle they're all that we need!

That is, we could theoretically do everything we can do on a computer using only LAMBDAs.

Extra For Experts in HW 1B is an illustration of this (doing recursion using only LAMBDAs and without using any DEFINEs).

Efficiency

If the argument sentence has N numbers in it, how many multiplications do we perform? How many calls to `se` do we do?

```
(define (square x) (* x x))

(define (square-sent sent)
  (if (empty? sent)
      '()
      (se (square (first sent))
          (square-sent (bf sent)))))
```

We perform N multiplications, and we call `se` N times.

Efficiency

If there are N numbers in the argument sentence, then how many numerical comparisons do we do (in the worst case)?

```
(define (sort sent)
  (if (empty? sent)
      '()
      (insert (first sent)
              (sort (bf sent)))))

(define (insert num sent)
  (cond ((empty? sent) (se num sent))
        ((< num (first sent))
           (se num sent))
        (else
           (se (first sent)
               (insert num (bf sent))))))
```


Efficiency

The number of comparisons is …
0 + 1 + 2 + … + (N-1)

This simplifies to (1/2)N(N-1).

Efficiency

Why are we concerned with the number of primitive operations performed? Shouldn't we be concerned about the *time* that it takes to run our procedures?

The actual *time* that a procedure takes is dependent on the hardware. We want to be able to talk about the efficiency of our *procedures*. We want to be able to compare the efficiencies of different *procedures* without making any assumptions on hardware.

An efficient *procedure* should be efficient no matter what hardware you're using.

Efficiency

Back to the sorting procedure a few slides back …

The number of comparisons is equal to (1/2)N(N-1).

For large N, this is roughly $(1/2)N^2$.

The constant factor of 1/2 isn't important, since we don't know what we're halving. That is, we don't know exactly how long it takes to do one comparison.

The running time of the sort procedure is *proportional* to N^2. How can we formalize this notion?

Efficiency

We formalize this using Θ (Big Theta).

We say that the running time of the sort function is $\Theta(N^2)$ and the running time of the square-sent function is Θ(N).

Sometimes we say "order of growth in time" rather than "running time".

A function f(x) is Θ(g(x)) if:
There exist constants K_1, K_2, and N such that for all x > N, $K_1*|g(x)| < |f(x)| < K_2*|g(x)|$.

13

Efficiency

Properties of Θ:

If f(x) is asymptotically greater than g(x), then Θ(f(x)+g(x)) = Θ(f(x)).

If C is a constant, then Θ(C*f(x)) = Θ(f(x)).

If C is a constant, then Θ(f(x)+C) = Θ(f(x)).

We only care about LARGE values of input.

14

Efficiency

```
(define (foo n)
  (if (= n 0)
      1
      (+ (helper n) (foo (- n 1)))))

(define (helper x)
  (if (= x 0)
      1
      (+ 1 (helper (- x 1)))))
```

What is the order of growth in time of foo?

15

Iterative vs. recursive processes

```
(define (count sent)
  (if (empty? sent)
      0
      (+ 1 (count (bf sent)))))

(count '(what yeah okay))
(+ 1 (count '(yeah okay)))
(+ 1 (+ 1 (count '(okay))))
(+ 1 (+ 1 (+ 1 (count '()))))
(+ 1 (+ 1 (+ 1 0)))
(+ 1 (+ 1 1))
(+ 1 2)
3   ; This is a recursive PROCESS.
    ; count requires Theta(n) space.
```

16

Iterative vs. recursive processes

```
(define (count2 sent)
  (define (iter wds result)
    (if (empty? wds)
        result
        (iter (bf wds) (+ result 1))))
  (iter sent 0))

(count2 '(what yeah okay))
(iter '(what yeah okay) 0)
(iter '(yeah okay) 1)
(iter '(okay) 2)
(iter '() 3)
3   ; This is an iterative PROCESS.
; count2 requires Theta(1) (constant) space.
```

17