MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring Semester, 1999

Recitation – Wednesday, February 10

1. More Special Forms

The special form **begin** has the following form. It evaluates each one of the expressions in turn. The value of the begin expression is the value of the last expression.

```
(begin <expr1> <expr2> ... <exprN>)
```

Example: `(begin (+ 5 3) (- x 5) (* 9 9))`
Wait a second... Why do we need this?

The special form **cond** has the following form. Conditional expressions are evaluated as follows. **<pred1>** is evaluated, and if that value is not **#f**, then the value of the cond clause is the value of **<expr1>**. If **<pred1>** evaluates to **#f**, then **<pred2>** is evaluated, etc.. If all of the predicates are **#f**, then the else expression **<exprE>** is evaluated and returned.

```
(cond (<pred1> <expr1>)
      (<pred2> <expr2>)
             ...
      (<predN> <exprN>)
      (else    <exprE>))
```

```
Example:  (define (sign x)
            (cond ((> x 0) 1)
                   (< x 0) -1)
                   (else 0))
```

2. Iterative vs Recursive Processes

Consider the following functions similar to those presented in lecture.

```
(define (mul2 n m)
  (define (iter count ans)
    (if (= count 0)
        ans
        (iter (- count 1) (+ m ans))))
  (iter n 0))

(define (mul3 n m)
  (cond ((= n 0) 0)
        ((even? n)
         (double (mul3 (halve n) m)))
        (else
         (+ m (mul3 (- n 1) m)))))
```

Now write a procedure mul4 that computes **m*n** in $\theta(\log n)$ time in $\theta(1)$ space.

```
(define (mul4 m n)
  (define (mul4-iter m count ans)
    (cond ((= count 0) ans)
          ((even? count)
           (mul4-iter (double m) (halve count) ans))
          (else
           (mul4-iter m (- count 1) (+ m ans)))))
  (mul4-iter m n 0)
)
```

3. More Iterative vs Recursive Processes

Consider the following two procedures.

```
(define (count-down x by)
   (cond ((< x 0) #t)
         (else (print x)
               (count-down (- x by) by))))

(define (count-up x by)
   (cond ((< x 0) #t)
         (else (count-up (- x by) by)
               (print x))))
```

What happens for each of

```
⇒ (count-down 11 3)
 11 8 5 2
⇒ (count-up 11 3)
 2 5 8 11
```

Write a function `count-up-2` which performs the same way as `count-up`, but is an iterative process.

```
(define (count-up-2 x by)
   (define (iter val)
     (cond ((> val x) #t)
           (else (print val)
                 (iter (+ val by)))))
   (iter (modulo x by))
)
```

4. Orders of Growth

Why should we care?

Name	θ notation	n = 2	n = 10	n = 100
Constant	$\theta(1)$	1	1	1
Logarithmic	$\theta(\log n)$	1	3.33	6.66
Linear	$\theta(n)$	2	10	100
Quadratic	$\theta(n^2)$	4	100	10,000
Exponential	$\theta(2^n)$	4	1024	$\approx 1.26 \times 10^{30}$

At 1 billion operations per second (current state of the art), if you were to run an exponential time algorithm in the lab on a data set of size $n = 100$, you would be waiting for approximately 4×10^{11} centuries for the code to finish running!

Formal Definition

Let R be some resource (e.g. space or time) used by a computation, and suppose R is a function of the size n of a problem. The amount of resources consumed will be $R(n)$. We say $R(n)$ has order of growth $\Theta(f(n))$ written $R(n) = \Theta(f(n))$ if there is some constant k_1 and k_2 independent of n such that

$$k_1 f(n) \leq R(n) \leq k_2 f(n)$$

for sufficiently large n.

5. Order of Growth Examples

For the following functions R, find the simplest and slowest growing function f for which $R(n) = \Theta(f(n))$.

(a) $R(n) = 6$

$\theta(1) \quad 1 \cdot 1 \leq 6 \leq 6 \cdot 1 \quad \forall n > 0$

(b) $R(n) = n^2 + 3$

$\theta(n^2) \quad 1 \cdot n^2 \leq n^2 + 3 \leq 2 \cdot n^2 \quad \forall n > 2$

(c) $R(n) = 6n^3 + 3n^2 + 7n + 100$

$\theta(n^3) \quad 1 \cdot n^3 \leq 6n^3 + 3n^2 + 7n + 100 \leq 7 \cdot n^3 \quad \forall n > 100$

(d) $R(n) = 5 \cdot \log(n^6)$

$\theta(log(n)) \quad 1 \cdot \log(n) \leq 5 \cdot \log(n^6) \leq 40 \cdot \log(n) \quad \forall n > 1$

(e) $R(n) = 2^{3n+7}$

$\theta(8^n) \quad 1 \cdot 8^n \leq 2^{3n+7} \leq 2^8 \cdot 8^n \quad \forall n > 0$

What are the Orders of Growth (space and time) for the procedures listed below?

```
(define (fact1 n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```

Time = $\theta(n)$
Space = $\theta(n)$

```
(define (fact2 n)
   (define (helper cur k)
       (if (= k 1)
           cur
           (helper (* cur k) (- k 1))))
   (helper 1 n))
```

Time = $\theta(n)$
Space = $\theta(1)$

```
(define (fib1 n)
   (cond ((= n 0) 0)
         ((= n 1) 1)
         (else (+ (fib1 (- n 1))
                  (fib1 (- n 2))))))
```

Time = $\theta(2^n)$
Space = $\theta(n)$

```
(define (fib2 n)
    (define (fib-iter a b count)
       (if (= count 0)
           b
           (fib-iter (+ a b) a (- count 1))))
 (fib-iter 1 0 n))
```

Time = $\theta(n)$
Space = $\theta(1)$