MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring 2006

Recitation 5
More Orders of Growth: Solutions

Special Forms

1. *begin* - `(begin` *expr1 expr2 ... exprn*`)`
   First evalutate *expr1*, then *expr2*, and so on. The value of the `begin` statement is the value of the last expression in the sequence.

2. *let* - `(let ((`*name1 val1*`) (`*name2 val2*`) ... (`*namen valn*`))` *body*`)`
   Syntactic sugar for the following:
   `((lambda (`*name1 name2 ... namen*`)` *body*`)` *val1 val2 ... valn*`)`.
   Used to bind additional names inside a procedure body.

Typical Orders of Growth: Review

- $\Theta(1)$ - 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: `(call-again (/ n c))`.
- $\Theta(n)$ - Linear growth. At each iteration, the problem size is decremented by a constant amount: `(call-again (- 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.

Problems

1. ```
   (define (fact n)
     (if (= n 0)
         1
         (* n (fact (- n 1)))))
   ```
   Running time? $\Theta(n)$ Space? $\Theta(n)$
   ```

2.
```
(define (find-e n)
  (if (= n 0)
      1.
      (+ (/ (fact n)) (find-e (- n 1)))))
```
Running time? $\Theta(n^2)$ Space? $\Theta(n)$

3. 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.

```
(define (prime? p)
   (define (helper n)
      (cond ((> n (sqrt p)) #t)
            ((divisible? p n) #f)
            (else (helper (+ n 1)))))
   (helper 2))
```

Running time? $\Theta(n\sqrt{n})$ Space? $\Theta(1)$

4. Write an iterative version of `find-e`.

```
(define (find-e-iter n)
  (define (helper n s)
      (if (= n 0) s
        (helper (- n 1) (+ s (/ (fact n))))))
  (helper n 1.0))
```

Running time? $\Theta(n^2)$ Space? $\Theta(n)$

5. Write a version of `sum-by-halves` (from your problem set) that only computes the midpoint between `a` and `b` once per iteration.

```
(define (sum-by-halves a b)
  (cond ((= a b) a)
        ((= (- b a) 1) (+ b a))
        (else
          (let ((mean (floor (/ (+ a b) 2))))
            (+ (sum-by-halves a mean)
               (sum-by-halves (inc mean) b))))))
```

Running time? $\Theta(n)$ Space? $\Theta(n)$