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

Recitation 5, Friday February 23

List (+ Recursion + Orders of Growth) Problems Dr. Kimberle Koile

Fill in the code for these recursive procedures. Assume recursive processes (not iterative).

1. This procedure returns the length (i.e., number of elements) in a list.

```
(define (length lst)
  (if (null? lst)
      0
      (+ 1 (length (cdr lst))))
)
```

time = Θ(n)
space = Θ(n)
n is length of list arg

2. This procedure returns the nth element of a list, where the first element index is 0.

```
(define (list-ref lst n)
  (if (= n 0)
      (car lst)
      (list-ref (cdr lst) (- n 1)))
)
```

time = Θ(n)
space = Θ(n)
n is the element index n

3. This procedure returns #t if obj is an element of a list; #f if it is not.
(Hint: Use the procedure `equal?`.)

```
(define (member? obj lst)
  (cond ((null? lst) #f)
        ((equal? obj (car lst)) #t)
        (else (member? obj (cdr lst))))
)
```

time = Θ(n)
space = Θ(1)
n is length of list arg

4. The procedure returns a new list that has exactly one instance of each element in the original list.
(Hint: Use the procedure `member?`.) e.g., (remove-duplicates (list 1 2 1 2 3 4)) => (1 2 3 4)

```
(define (remove-duplicates lst)
  (cond ((null? lst) '())                          ; or lst, since it's empty
        ((member? (car lst) (cdr lst))
         (remove-duplicates (cdr lst)))
        (else (cons (car lst)
                    (remove-duplicates (cdr lst))))
  )                                                ; 1 pending operation
```

both are Θ(n)

time = Θ(n²) } for worst-case
space = Θ(n)
n is length of list arg