Alpha Chen
CS 3304 Comparative Languages
Professor Edwards
HW 04
September 21st, 2005

**Problem 1**: Chapter 15, Problem 6
*What does the following Scheme function do?*

```
(define (y s lis)
  (cond
    ((null? lis) '())
    ((equal? s (car lis)) lis)
    (else (y s (cdr lis)))))
```

This function returns lis if s is a top-level member of lis, and the empty set otherwise.

**Problem 2**: Chapter 15, Programming Exercise 13
*Write a Scheme function that takes a lit as a parameter and returns it with the second top-level element removed. If the given list does not have two elements, the function should return* `()`.

```
(define (remove-second-member lst)
  (if (< (length lst) 2)
    '()
    (cons (first lst) (cddr lst))))
```

Note that this is assuming that the problem, as stated, really asks for `()` to be returned on *less than* two elements, instead of "does not have". In the case that we are taking the problem statement literally, `s/< (length lst) 2/not (= (length lst) 2)/`.

**Problem 3**:
*Write your own implementation of the Scheme built-in function sublist:*

```
(sublist list start end)
```

*Returns a newly allocated list formed from the elements of list beginning at index* start *(inclusive) and ending at* end *(exclusive). You may assume that start and end are integers that satisfy:*

```
0 <= start <= end <= (length list)
```

*Name your function* `my-sublist`*, since sublist is already provided as a standard operation in Scheme (you may not use* `sublist`*,* `list-head`*, or* `list-tail` *in your solution).*

```
(define (my-sublist lst start end)
  (cond ((> start 0) (my-sublist (rest lst) (- start 1) (- end 1)))
        ((= start 0) (my-sublist (reverse lst) -1 (- end 1)))
        ((> (- (length lst) end) 0) (my-sublist (rest lst) -1 (+ end 1)))
        (else (reverse lst))))
```

**Problem 4**:
*Analyze the performance of your* `my-sublist` *implementation, and describe its worst-case performance using big-oh notation, in terms of the number of* `car` *or* `cdr` *operations performed (including those required by any predefined operations you use). Justify your performance analysis.*

The worst case scenario is O(n), or linear with the number of elements in the list. This is because my-sublist cannot perform any more cdrs (rests) than there are elements in the array. Worst case scenarios involve reducing the list to one element (since it acts inclusively on start and exclusively on end), so only (- (length lst) 1) rest operations can be performed.

Although reverse may use car or cdr, I am not sure if it does, or how many it uses. I believe reverse should be O(n) anyway, as it can be accomplished by using an inject (fold-right?) method. Even if intermediate student Scheme doesn't support it, I imagine that behind the scenes, it would be performed that way. But even if it is O(n), my-sublist would still only be O(2n).