Here are my implementations of adjoin, union, and intersection:

```
(define (define (adjoin x s) (cons x s))
The adjoin function is clearly constant.
(define (union s t)
  (define (iter result a b)
    (cond
     ((and (null? a) (null? b)) result)
     ((null? a) (iter result b nil)) ; set a is done, now work on set b
     ((not (member? (car a) result)) ; if not already in the resulting set
      (iter (cons (car a) result) (cdr a) b))
     (else
      (iter result (cdr a) b))
    )
  )
  (iter nil s t))
  ) Since the resulting set has no duplicates, scanning the result takes theta(n) time. We do this
  for theta(n^2) items in the sets a and b (n unique elements, each having O(n) duplicates). So,
  overall, we have theta(n^3).
(define (intersection s t)
  (define (iter result a b)
    (cond
     ((null? a) result)
     ((and (not (member? (car a) result)) (member? (car a) b))
      (iter (cons (car a) result) (cdr a) b))
     (else
      (iter result (cdr a) b))
    )
  )
  (iter nil s t))
  ) Intersection takes quartic time, since each of the theta(n^2) elements in a requires a scan of
  the theta(n^2) elements in b.

However, for intersection, I can do better. I could remove the duplicates in both sets by unioning
  each with the empty set. Then I could run intersection. That would be theta(n^3) for the unioning
  and an additional theta(n^2) for the intersection, yielding an overall time of theta(n^3).
```