```
(define (list-len l)
    (if (null? l)
    0
    (+ 1 (list-len (cdr l)))
    )
    ;; #f TODO: return something other than FALSE
)

(define (inc-list n)
    (if (= 1 n)
        (list 1)
        (if (= 0 n)
            (list)
        (append (inc-list (- n 1)) (list n))
    )
    )

     ;; TODO: return something other than FALSE
    )

(define (rev-list l)
    (if(null? l)
    (list)
    (if(null? (cdr l))
        l
        (append(rev-list (cdr l))(list (car l)))
    )
    )
     ;; TODO: return something other than FALSE
    )

(define (my-map f l)
        (if(null? l)
            (list)
            (if(null? (cdr l))
                (list (apply f (list (car l))))

            (append (list (apply f (list (car l)))) (my-map f (cdr l)))
            )
        )
    )

(define (merge-sort l)
    ;; Split a list into two halves, returned in a pair. You may uncomment this.

    (define (split l)
        (define (split-rec pair)
            (let ((front (car pair)) (back (cdr pair)))
                (if (>= (length front) (length back))
                    pair
                    (split-rec (cons (append front (list (car back))) (cdr
back))))))
        (split-rec (cons (list (car l)) (cdr l))))
    (define (merge list-1 list-2)

        (if (null? list-1)
            list-2
            (if (null? list-2)
                list-1
```

```
                (if (< (car list-1) (car list-2))
                    (cons (car list-1) (merge (cdr list-1) list-2))
                    (cons (car list-2) (merge (cdr list-2) list-1))
                )
            )
        )
     ;; TODO: return something other than FALSE
    )
    (if (null? l)
        '()
        (if (null? (cdr l))
            l
            (let ((split-l (split l)))
                (let ((list-1 (car split-l)) (list-2 (cdr split-l)))
                    (merge (merge-sort list-1) (merge-sort list-2)))
            )
        )
    )
)
```