```
; Bee Cha
; Lab 8
; This is a merge sort example program using a
; functional language, scheme. You will use the divide
; and conquer algorithm on the list. But instead of cutting
; the list in half, the list will be divided by every other
; odd element and every other even element. I have
; two functions to grab the odd elements and even elements
; and it will put them into another list. Once you get
; the list down to singletons, it will call merge on
; the singletons. But before merging, it checks and compares
; to see if which element is less. The lesser value goes in
; front of the other.

;------This is how you define a function in scheme--------

(define (square x) (* x x))


;---------------------Length of a list---------------------
; If the list is empty (NULL) then we should return 0
; Else if it's not empty, then we should increment
; length by 1 and do a recursive call on the tail of
; the list by using (cdr (LIST) )
;
; (length (4 2 5) )
; (1 + (length (2 5) ))
; (1 + 1 + (length (5) ))
; (1 + 1 + 1 + (length () ))
; (1 + 1 + 1 + 0)
; Value = 3
;
; Not sure if this function is needed

(define (length n)
   ( cond  ( (null? n) 0 )
      (else  (+ (length (cdr n) ) 1) )
   )
)

;------------------Spliting the list---------------------
; The original way of splitting a list is to split it down
; the middle, but SCHEME doesn't have a function to do that.
; Rather than splitting the list in half down the middle,
; splitting the list into every odd and even element into
; two separate lists is easier. Split List1 to be every other
; odd element, and split List2 to be every other even element
;
; (split `(5 2 9 2 8 1))
; List1 = (5 9 8)
; List2 = (2 2 1)
; Thus returning the list of:
; ((5 9 8) (2 2 1))

; First to define a function to grab every other odd elements
(define (odd_e i)
   (if (null? i) `()
      (if (null? (cdr i)) (list (car i)) (cons (car i) (odd_e (cddr i)))
      )
   )
)

; Second to define a function to grab every other even elements
(define (even_e j)
   (if (null? j) `()
      (if (null? (cdr j)) `() (cons (cadr j) (even_e (cddr j)))
      )
   )
)

; Third is to split the list to have list1 with every other odd element
; and list2 with every other even element
(define (split k)
   (cons (odd_e k) (cons (even_e k) `()))
)

;----------------------------Merging Lists--------------------------
; When you have two lists, List1 andd List2, you will merge it together
; to become a whole new List. But you have to compare each element to
; see which element is less. If it's less, then put that element first.
; To access the head of the list, you use the car function
; (car `(1 2 3)) => 1
;
; List1 = (5 4 1)
; List2 = (0 2 9)
;
; Merge List1 List2
; (0 1 2 4 5 9)

(define (merge L R)
   (if (null? L) R        ; If List1 is empty, then just return List2
      (if (null? R) L     ; If List2 is empty, then just return List1

         ; Merge-Sort will call the split
         (if (< (car L) (car R)) (cons (car L) (merge (cdr L) R))
                           (cons (car R) (merge (cdr R) L))
         )
      )
```

```
    )
)

;---------------------------Merge Sort--------------------------------
; Merge sort will split the list recursively down to its single element
; and then it will call the merge function to merge the element until
; it forms a new list with the sorted elements
;
; List = ( 7 8 5 6 4 3 2 10)
; Split:
;   (7 5 4 2) | (8 6 3 10)
; Split:
;    (7 4) | (5 2) | (8 3) | (6 10)
; Split:
;   (7) | (4) | (5) | (2) | (8) | (3) | (6) | (10)
; Merge:
;   (4 7) | (2 5) | (3 8) | (6 10)
; Merge:
;   (2 4 5 7) | (3 6 8 10)
; Merge:
;   (2 3 4 5 6 7 8 10)
; Done:

(define (mergesort L)
    (if (null? L) L                  ; Empty List case
        (if (null? (cdr L)) L        ; Singleton case
            (
                merge (mergesort (car (split L))) (mergesort (cadr (split L)))
            )
        )
    )
)
```

Sample Run:

```
1 ]=> (odd_e`(9 3 5 2 7 1 0 4 6))

;Value 11: (9 5 7 0 6)

1 ]=> (even_e `(9 3 5 2 7 1 0 4 6))

;Value 12: (3 2 1 4)

1 ]=> (split `(9 3 5 2 7 1 0 4 6))

;Value 13: ((9 5 7 0 6) (3 2 1 4))

1 ]=> (merge `(1 3 5) `(2 4 6))

;Value 14: (1 2 3 4 5 6)

1 ]=> (mergesort `(9 3 5 2 7 1 0 4 6))

;Value 15: (0 1 2 3 4 5 6 7 9)
```