CMSC 10500-1: Homework 5

(due on Friday July 9th)

Merge Sort

In the class we saw a way to sort a list of numbers. The basic idea there was to remove one element of the list, sort the rest of the list and insert this element into the resulting list. For obvious reasons this method is called in **Insertion Sort**. In this exercise we will see another method to sort a list of numbers, called **Merge Sort**.

This is a typical case of a **Divide and Conquer algorithm**, and consists of the following phases:

- **Base Case:** The list has at most 1 element.
- **Divide:** Split the list (approximately) evenly into two lists.
- **Conquer:** Recursively sort the two lists.
- **Combine:** Merge the two sorted lists into one sorted list.

1. **(6 pts)** To implement the divide phase, write scheme functions `first-k` and `rest-k` which consume a list and a number `k` and produce a list. `first-k` produces a list containing the first `k` items of the list, and `rest-k` contains all but the first `k` items of the list. If the list has at most `k` items then `rest-k` returns an empty list.

```
> (first-k (cons 1 (cons 3 (cons 2 empty))) 2)
(cons 1 (cons 3 empty))
> (rest-k (cons 1 (cons 3 (cons 2 empty))) 2)
(cons 2 empty)
```

2. **(10 pts)** To implement the combine phase, write a scheme function `merge` which consumes two lists of numbers, which are independently sorted (in increasing order) and produces one sorted list containing all the numbers in both the lists.

```
> (merge (cons 1 (cons 4 (cons 6 empty)))
         (cons 2 (cons 3 (cons 6 (cons 8 empty))))  )
(cons 1 (cons 2 (cons 3 (cons 4 (cons 6 (cons 6 (cons 8 empty)))))))
```

3. **(4 pts)** Finally implement the function `merge-sort` which consumes a list of numbers `alon` and sorts them. Be sure to split your list (approximately) evenly into two lists. You may use the in-built scheme function `length` which returns the length of a list.

```
> (merge-sort (cons 1 (cons 6 (cons 4
                (cons 2 (cons 3 (cons 8 (cons 6 empty)))))))
(cons 1 (cons 2 (cons 3 (cons 4 (cons 6 (cons 6 (cons 8 empty)))))))
```

Here is skeleton code to get you started....

```
;; first-k: list number -> list
;;   return list of the first k elements
(define (first-k lst k)
   .....)

;; rest-k: list number -> list
;;   return all but the first k elements
(define (rest-k lst k)
    .....)

;; merge: list-of-numbers list-of-numbers -> list-of-numbers
;;    consume two sorted lists and output one sorted list
;;    containing the numbers in both the lists
(define (merge lon1 lon2)
    .....)

;; merge-sort : list-of-numbers -> list-of-numbers
;;   sort the list of numbers in increasing order
(define (merge-sort alon)
    .....)
```