List Tree

Hannah Zhang

1. Write the function first-n that takes two inputs, a list, lyst, and a number, n. It returns a list containing the first n elements of lyst.

```
> (first-n '(A B C D E) 3)
(A B C)
```

```
; if the number is zero, return empty list
; if not, add the first num and do recursion
(define (first-n lyst n)
  (if (= n 0)
      '()
      (cons (car lyst) (first-n (cdr lyst) (- n 1)))))
```

2. Write the function part1 which takes a list as its input and returns a list containing the first half of the elements of the input list. If the input list has an odd number of elements, the extra element should NOT be part of part1. Make sure to be very careful with an input of the empty list.

```
> (part1 '(1 2 3 4 5 6))
(1 2 3)
> (part1 '(1 2 3 4 5))    ; the 3 will go into part2
(1 2)
> (part1 '())
()
```

```
; uses floor, length, first-n
(define (part1 lyst)
  (if (empty? lyst)
      '()
      (first-n lyst (floor (/ (length lyst) 2)))))
```

3. Write the function rest which takes a list as its input and returns list containing those elements in the list that would not be returned by part1. Again, make sure your function handles empty lists properly.

```
> (rest '(1 2 3 4 5 6))
(4 5 6)
> (rest '(1 2 3 4 5))    ; the 3 will go into rest
(3 4 5)
> (rest '())
()

; uses repeated
(define (rest lyst)
  (if (empty? lyst)
      '()
      ((repeated cdr (floor (/ (length lyst) 2))) lyst)))
```

4. Write the function middle-datum which takes a list as its input and returns the middle element of the list. If the list contains an even number of elements, the middle element will be second of the two in the middle. You may assume that middle-datum will only be called with lists that contain at least one element.

```
> (middle-datum '(1 2 3))
2
> (middle-datum '(1 2 3 4))
3

; uses rest from above
(define (middle-datum lyst)
  (car (rest lyst)))
```

5. Write the function part2 that takes a list as its input and returns the list excluding middle-datum and part1.

```
> (part2 '(1 2 3))          ; middle-datum is 2
(3)
> (part2 '(1 2 3 4))        ; middle-datum is 3
(4)
> (part2 '(1 2 3 4 5))      ; middle-datum is 3
(4 5)
```

```
> (part2 '(1 2 3 4 5 6))  ; middle-datum is 4
(5 6)

; everything but the first part and middle → cdr of rest
(define (part2 lyst)
  (if (empty? lyst)
      '()
      (cdr (rest lyst))))
```

6. Write the function list->tree that takes a list of increasing numbers as its input and returns a **weight-balanced** binary search tree containing those numbers.

```
; returns procedure if not printed
; makes a bintree my using other functions on the lyst
(define (list->tree lyst)
  (if (empty? lyst)
      tet
      (make-bintree (middle-datum lyst)
                    (list->tree (part1 lyst)) (list->tree (part2
lyst)))))


(define binary-tree (list->tree '(1 2 3 4 5 6 7 8)))
→ when testing, the numbers are correct
```