Quiz 3b

1. (3 points) Use big-Θ notation to describe the runtime of the following procedures:
   1. The procedure keep, which takes a predicate and a sentence and returns a new sentence containing the elements that satisfy the predicate. Assume the predicate has runtime Θ(1).

> (keep even? '(1 2 3 4))

(2 4)

Θ(*n*)\_\_ (1 point all or nothing)

* 1. A procedure called every-word-pair, which takes a sentence and returns a new sentence containing every possible way to concatenate an earlier word with a later one.

> (every-word-pair '(head crab master)

(headcrab headmaster crabmaster)

Θ(*n*2)\_\_ (1 point all or nothing)

* 1. The procedure larger-power-of-3, as defined below.

(define (larger-power-of-3 n)

(larger-power-helper n 1))

(define (larger-power-helper n i)

(if (> i n)

i

(larger-power-helper n (\* i 3))))

> (larger-power-of-3 8)

9

> (larger-power-of-3 10)

27

Θ(log *n*)

Θ(log3 *n*) is also correct. 1 point all or nothing.

1. (3 points) Identify what kind of process each procedure generates.
   1. What kind of process does count-letters generate?

Circle one: linear recursive tree-recursive iterative

**Iterative. 1 point all or nothing.**

(define (count-letters sent)

(count-helper sent 0))

(define (count-helper sent letter-count)

(if (empty? sent)

letter-count

(count-helper (bf sent)

(+ letter-count

(count (first sent))))))

* 1. What kind of process does remove-letter generate?

Circle one: linear recursive tree-recursive iterative

**Iterative. 1 point all or nothing.**

(define (remove-letter letter wd)

(cond ((empty? wd) (bf letter))

((eq? (first wd) (first letter))

(remove-letter letter (bf wd)))

(else

(remove-letter (word letter (first wd))

(bf wd)))))

* 1. What kind of process does remove-first-word generate?

Circle one: linear recursive tree-recursive iterative

**Linear recursive. 1 point all or nothing.**

(define (remove-first-word sent)

(remove-helper sent (first sent)))

(define (remove-helper sent wd)

(cond ((empty? sent) '())

((eq? (first sent) wd)

(remove-helper (bf sent) wd))

(else

(se (first sent) (remove-helper (bf sent) wd)))))

1. (4 points) Write the following procedures so that they generate **iterative processes**. You will receive no credit for writing a procedure that generates a recursive process. Hint: You may want to do most of the computation in a helper procedure that takes one more argument than the original procedure.
   1. Write a procedure called square-every, which squares every number in a sentence of numbers:

> (square-every '(1 2 3))

(1 4 9)

(define (square-every sent)

(square-helper sent '()))

(define (square-helper sent result)

(if (empty? sent)

result

(square-helper (bf sent)

(se result

(\* (first sent) (first sent))))))

2 points if correct; 1 point if structurally correct but has a few issues.

* 1. Write a procedure called count-word, which counts how many times a certain word occurs in a sentence:

> (count-word 'you '(you are what you eat))

2

(define (count-word wd sent)

(count-helper wd sent 0))

(define (count-helper wd sent total)

(cond ((empty? sent)

total)

((equal? (first sent) wd)

(count-helper wd (bf sent) (+ total 1)))

(else

(count-helper wd (bf sent) total))))

2 points if correct; 1 point if structurally correct but has a few issues.