RECURSION, ITERATION, AND EFFICIENCY 3

COMPUTER SCIENCE 61AS

Basics of Recursion and Iteration

1. What is a tail-recursive call?

2. Why do we say certain procedures generate recursive and iterative processes when recursion is used either way?

3. What is the primary reason to use iteration instead of recursion?

4. What is a disadvantage of using iteration?

5. Identify whether each procedure will generate a recursive or an iterative process.

   a.
```
(define (fac n)
    (if (= n 1)
        1
        (* n (fac (- n 1)))))
```

b.
```
(define (foo a b)
    (if (< a 0)
        b
        (foo (- a 1) (+ b 1))))
```

c.
```
(define (bar n)
    (define (loop i)
      (if (= i 0)
          nil
          (se (loop (quotient i 2)) (remainder i 2))))
    (loop n))
```

Practice with Recursion/Iteration

1. Determine whether each procedure generates a recursive/iterative process. If the procedure generates a recursive process, rewrite it so it generates an iterative one and vice versa for iterative processes.

   a.
```
(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))))))
```

   b.
```
(define (remove-letter letter wd)
  (cond ((empty? wd) "")
        ((eq? (first wd) letter) (remove-letter letter (bf wd)))
        (else (word (first wd) (remove-letter letter (bf wd))))))
```

Basics of Orders of Growth

1. Rank the orders of growth from slowest to fastest: $\theta(n), \theta(1), \theta(n^2), \theta(\log n)$

2. If we know that a procedure that has an input of size n runs in $\theta(n^2)$ time, is it possible to determine how long it will take to finish on a given input? Why/Why not?

3. Will a procedure that runs in $\theta(n^2)$ always run slower than a procedure that runs in $\theta(n)$? Why/Why not?

Practice with Orders of Growth

1. Using big-theta notation, classify the order of growth for both time and memory used by each of the procedures below.

   a.
   ```
   (define (min-sent sent)
     (if (empty? (bf sent))
         (first sent)
         (min (first sent) (min-sent (bf sent)))))
   ```

   b.
   ```
   (define (foo n)
     (define (loop num)
       (if (= num 0)
           ()
           (se (loop (quotient num 2)) (remainder num 2))))
     (loop n))
   ```

   c. A procedure called `all-pair-sums`, which takes a sentence of numbers and returns a sentence of all the sums of every possible pair of numbers in the sentence. Only give the order of growth for time.
   ```
   (all-pair-sums '(1 2 3)) => (3 4 5) ;; (1+2 1+3 2+3)
   (all-pair-sums '(2 4 5 6)) => (6 7 8 9 10 11)
   ```