# RECURSION 0.3

COMPUTER SCIENCE 61AS

## The Basics of Recursion

1. What are three things you find in every recursive function?

2. (a) What does the mathematical function factorial do? Write out your answer in either math or words.

   (b) Below is a Racket function that computes factorial. What is its domain and range? Identify the three things from question 1.

```
(define (factorial n)
    (if (<= n 1)
        1
        (* n (factorial (- n 1)))))
```

   (c) Write out the recursive calls made when we do `(factorial 4)`.

3. Below is a Racket function that computes the $n^{th}$ Fibonnaci number.

   (a) What is its domain and range? Also, identify the three things from question 1.

```
(define (fib n)
    (cond ((= n 0) 0)
          ((= n 1) 1)
          (else (+ (fib (- n 1)) (fib (- n 2))))))
```

   (b) Write out the recursive calls made when we do `(fib 4)`. (This will look like an upside-down tree).

## Practice with Recursion

1. What's wrong with the following code? Identify all of the things and fix them.

```
;; Multiplies two numbers together without using *
(define (multiply x y)
    (if (= x 0)
        1
        (+ y (multiply x y))))
```

2. We're going to write a function `square-sent` that takes in a sentence of numbers and squares each element.

   (a) What should our base case be? (Hint: when dealing with a numerical input, our base case is usually when the input is a 0 or 1. What might the correspondance be for when a sentence is the input?)

   (b) Write the function `square`.

   (c) If the first number in our sentence is a 3, what do we want the first number in our output to be?

   (d) Let's generalize this a little more. If our sentence is the variable `sent`, what do we want the first number in our output to be?

   (e) Assuming we now know how to figure out the first number in our output, what should the rest of the numbers be? (Remember: we have this really nifty function `square-sent` that computes the squares of a sentence of numbers...)

   (f) Using the parts you've come up with so far, write the entire `square-sent` function.

3. Write a procedure `merge`, which accepts two sorted sentences as input, and merges them together to produce a sorted sentence as output. (If you're having trouble with this problem, try breaking it down into steps like the question above!)

```
-> (merge '(2 2 8 19) '(1 5 6 7 10))
(1 2 2 5 6 7 8 10 19)
```

4. Write `foobar`. This procedure outputs a sentence of `foo`'s and `bar`'s, in the sequence `foo, bar, bar, foo, bar, bar, foo, ...`. The formal parameter indicates up to which term to output. (Remember, computer scientists start counting from 0!) The `remainder` function may be useful.

```
-> (foobar 4)
(foo bar bar foo bar)

-> (foobar 2)
(foo bar bar)
```