CS 61A, Summer 2006 July 3, 2006
HOMEWORK ASSIGNMENT 2A
DUE WEDNESDAY JULY 12, 2006 AT 11:59 PM

1. Exercises 1.11, 1.16, 1.35, 1.37, 1.38.

2. A "perfect number" is defined as a number equal to the sum of all its proper factors (factors that are strictly less than the number itself). For example, the first perfect number is 6, because its proper factors are 1, 2, and 3, and 1+2+3=6. The second perfect number is 28, because its proper factors are 1, 2, 4, 7, and 14, and 1+2+4+7+14=28. What is the third perfect number? Write a procedure `(next-perf n)` that tests numbers starting with `n` and continuing with `n+1`, `n+2`, etc. until a perfect number is found. Then you can evaluate `(next-perf 29)` to solve the problem. Hint: you'll need a `sum-of-factors` subprocedure.

Note: If you run this program when the system is heavily loaded, it may take quite a while to compute the answer! If you're getting impatient with your program, try tracing helper procedures to make sure your program is on track, or start by computing `(next-perf 1)` and see if you get 6.

3. What are the orders of growth in time and in space of the following procedures? Also, which procedures generate recursive processes and which generate iterative processes? Give a brief (about one sentence) justification for each of your answers. Assume that each procedure takes a positive integer n as argument.

```
(define (f1 n)
  (define (helper x y)
    (if (= x 0)
        y
        (helper (- x 1) (+ x y))))
  (helper n 0))

(define (f2 n)
  (if (< n 100)
      1
      (+ (f2 (- n 1)) (f2 (- n 1)))))

(define (f3 n)
  (if (< n 100)
      1
      (let ((temp (f3 (- n 2))))
        (+ temp temp))))

(define (f4 n)
  (define (helper x)
    (if (= x n)
        x
        (helper (+ x 1))))
  (helper 0))
```

Extra For Experts on next page ...

Extra For Experts. This is a **totally optional exercise**, worth no course credit whatsoever, intended only for people who have finished the assignment and are looking for something fun and/or intellectually challenging to do. You will never be expected to know anything about "extra for experts" problems in this course.

1. The partitions of a positive integer are the different ways to break the integer into pieces. The number 5 has seven partitions:

```
5 (one piece)
4, 1 (two pieces)
3, 2 (two pieces)
3, 1, 1 (three pieces)
2, 2, 1 (three pieces)
2, 1, 1, 1 (four pieces)
1, 1, 1, 1, 1 (five pieces)
```

The order of the pieces doesn't matter, so the partition 2, 3 is the same as the partition 3, 2 and thus isn't counted twice. 0 has one partition.

Write a procedure `number-of-partitions` that computes the number of partitions of its nonnegative integer argument.

2. Compare the `number-of-partitions` procedure with the `count-change` procedure by completing the following statement:

```
Counting partitions is like making change, where the coins are ...
```

3. (Much harder!) Now write it to generate an iterative process; every recursive call must be a tail call.