Recursion

Factorial

[Equation 2] $n! = 1 \text{ if } n == 0$

[Equation 3] $n! = n * (n - 1)! \text{ if } n > 0$

GCD

$gcd(a,b)$	is	b	if a divided by b has a remainder of zero
$gcd(a,b)$	is	$gcd(b,a \% b)$	otherwise

Fibonacci

$fib(n)$	is	0	if n is zero
$fib(n)$	is	1	if n is one
$fib(n)$	is	$fib(n - 1) + fib(n - 2)$	otherwise

function f

function g

```
f(a) is g(head(a),tail(a))
g(b,c) is b if c is the empty list
g(b,c) is
     g(head(c),tail(c)) if GT(head(c),b)
g(b,c) is g(b,tail(c)) otherwise
```

if

else if

else

"is" = return statement

Fossilized Pattern

```
int
countList(Node *items)
    {
if (n == 0)
    return 0;
else
    return 1 + countList(items); //should be tail(items)
}
```

Problem never gets smaller

Bottomless Pattern

```
int
div2(int n)
     {
if (n == 0)
     return 0;
else
     return 1 + div2(n - 2);
}
```

Never/only sometimes reaches base case

8. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
int sum(int a,int b)
    {
    if (a == b)
        return a;
    else
        return a + sum(a,b-1);
    }
```

(a) Yes

(b) No

18. What pattern does the following function implement?

```
Node *f(Node *a)
    {
    return g(head(a),tail(a));
    }
Node *g(Node *e,Node *b)
    {
    if (b == 0)
        return e;
    else if (isLT(head(b),e))
        return g(head(b),tail(b));
    else
        return g(e,tail(b));
    }
```

(a) the filter pattern

(b) the extreme pattern

(c) the filtered-accumulate pattern

(d) the accumulate pattern