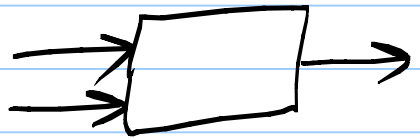
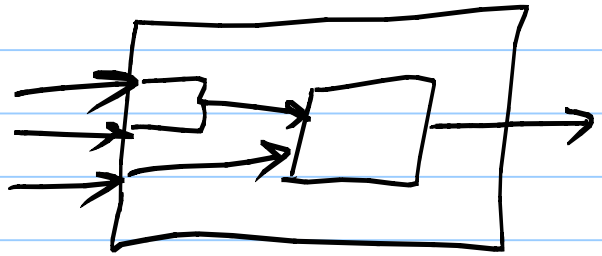
Recursion

Note Title 9/10/2007

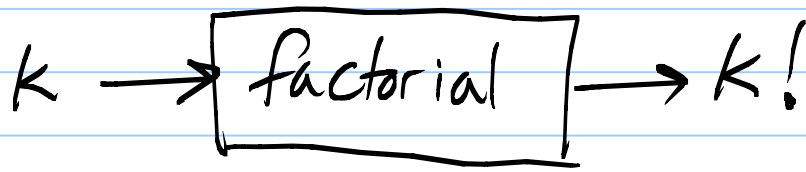
Method - procedural abstraction



Reduction — solve a problem by reducing it to another problem



Recursion — reduction of a problem to a (smaller) instance of the same problem



$\rightarrow 0! = 1$

$k \cdot \underbrace{k-1 \cdot k-2 \cdot k-3 \cdots 1}$

$k \cdot (k-1)!$

```
int fact(int k) {
    if (k == 0)
        return 1;
    else
        return k * fact(k-1);
}
```

Vote

Yes	No
20	>30

Test cases:

fact(0)

⇒ 1 ✓

fact(1)

1 * fact(0)

1 * 1

⇒ 1

```
int fact(int k) {
    if (k == 0)
        return 1;
    else
        return k * fact(k-1);
}
```

Base case — when you know the answer immediately

recursive call

combining step

fact(3)

evaluate using the substitution model

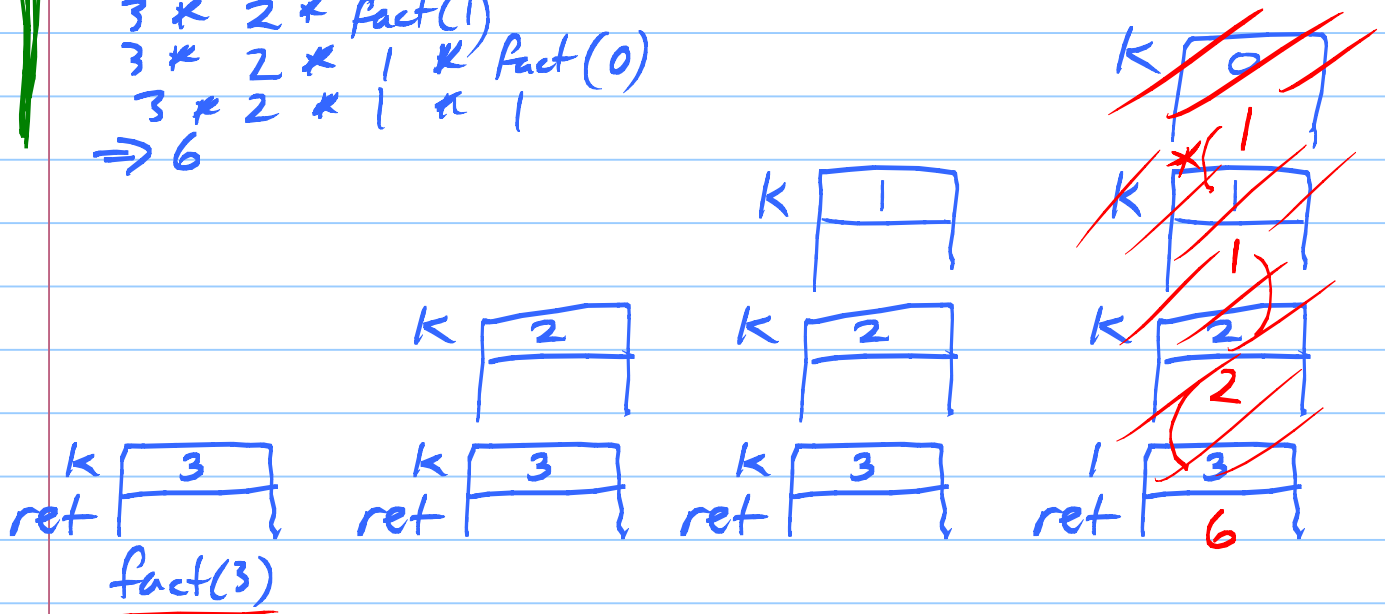
3 * fact(2)

3 * 2 * fact(1)

3 * 2 * 1 * fact(0)

3 * 2 * 1 * 1

⇒ 6



defining a method means you have to think about the algorithm — how it's going to work

```
int fact(int k) {
    if (k == 0)
        return 1;
    else
        return k * fact(k-1);
}
```

Use the abstraction!

⇓

Trust the recursion!

When you call a method you don't think about how it's going to work.

Assume
$a >= 0$

`--x` subtracts 1 from x
`++x` adds 1 to x

a + b

rewrite:

`--a + ++b`

$(a - 1) + (b + 1)$

$a + b$

```
3 + 5
2 + 6
1 + 7
0 + 8
```

```
int add(int a, int b) {
    if (a == 0)
        return b;
    else
        return add(--a, ++b);
}
```

base case

recursive call

NO COMBINING STEP

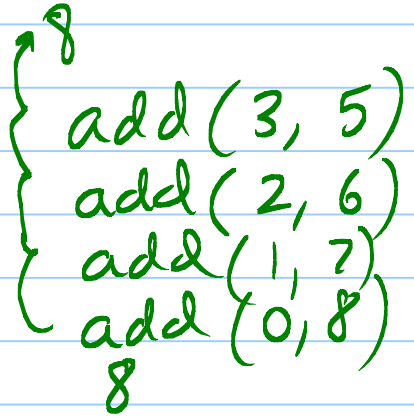
TAIL RECURSION

```
int add(int a, int b) {
    if (a == 0)
        return b;
    else
        return add(--a, ++b);
}
```

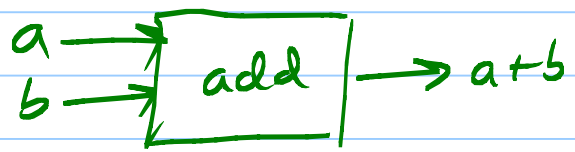
base case

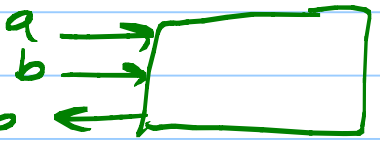
NO COMBINING STEP

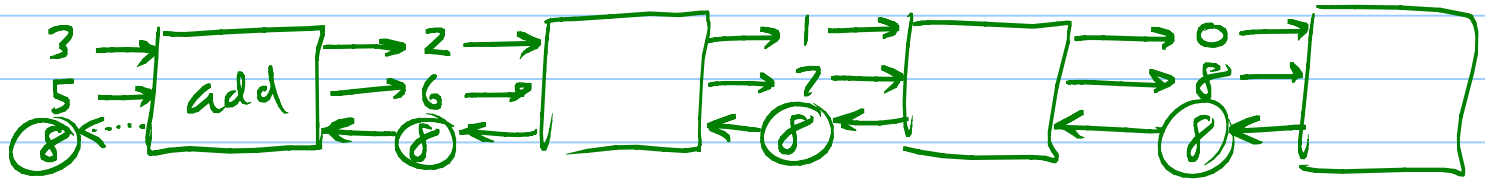
recursive call



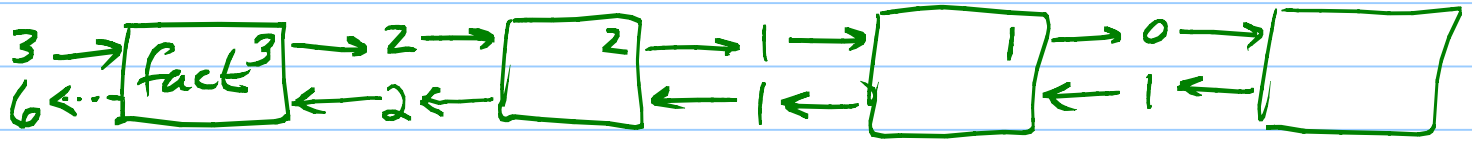
Dataflow Model — as a way to think about recursion







tail recursive



not tail recursive

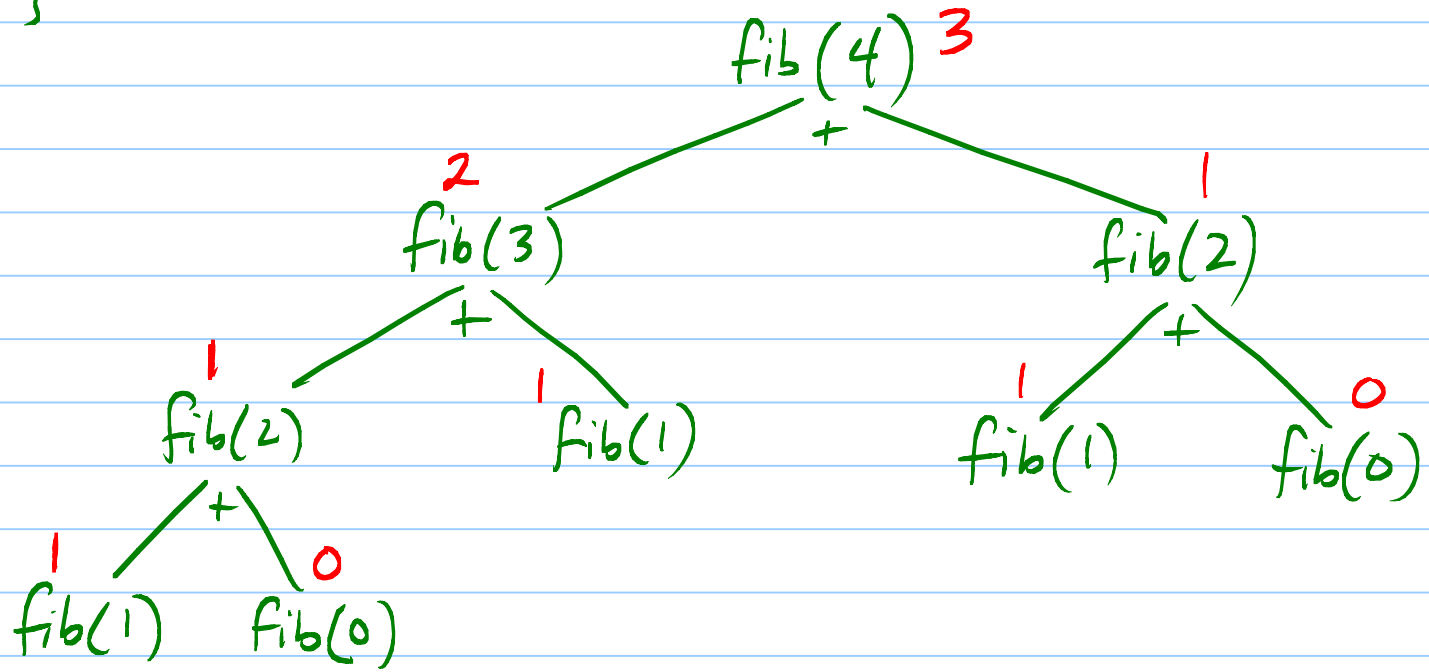
Fibonacci series

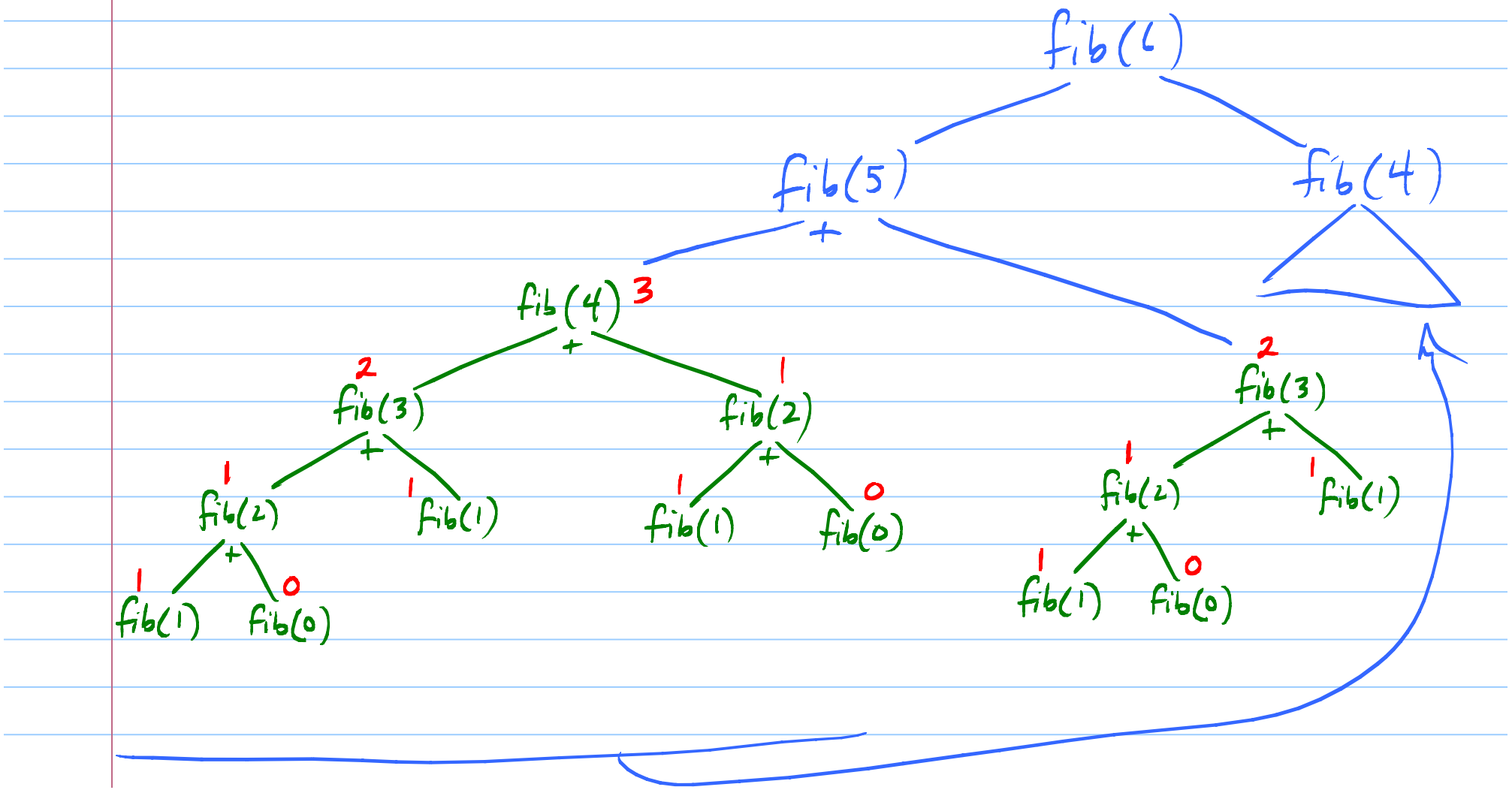
n	0	1	2	3	4	5	6	7	...
fib(n)	0	1	1	2	3	5	8	13	

$$fib(n) = \begin{cases} 0 & \text{if } n \text{ is } 0 \\ 1 & \text{if } n \text{ is } 1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

```
int fib(int n) {   //assume n >= 0
    if (n <= 1)
        return n;
    else
        return fib(n-1) + fib(n-2);
}
```

```
int fib(int n) {   // assume n >= 0
    if (n <= 1)
        return n;
    else
        return fib(n-1) + fib(n-2);
}
```




fib(6)
fib(5)
fib(4)
+
fib(4)
3
+
2
fib(3)
+
1
fib(2)
+
2
fib(3)
+
1
fib(2)
+
1
fib(1)
1
fib(1)
+
0
fib(0)
1
fib(1)
+
0
fib(0)
1
fib(1)
+
1
fib(1)
0
fib(0)

Want: More efficient algorithm

- way you go about solving a problem
- strategy for the computation
- steps