

UNIT 5A

Recursion: Basics

15110 Principles of Computing,
Carnegie Mellon University - CORTINA

1

Recursion

- A recursive operation is an operation that is defined in terms of itself.



<http://fusionanomaly.net/recursion.jpg>



Sierpinski's
Gasket

15110 Principles of Computing,
Carnegie Mellon University - CORTINA

2

Recursive Definitions

- Every recursive definition includes two parts:
 - Base case (non-recursive)
A simple case that can be done without solving the same problem again.
 - Recursive case(s)
One or more cases that are “simpler” versions of the original problem.
 - By “simpler”, we sometimes mean “smaller” or “shorter” or “closer to the base case”.

GCD

```
def gcd2(x, y)
  if y == 0 then
    return x
  else
    return gcd2(y, x % y)
  end
end
```

} base case

} recursive case
(a “simpler”
version of
the same
problem)

Factorial

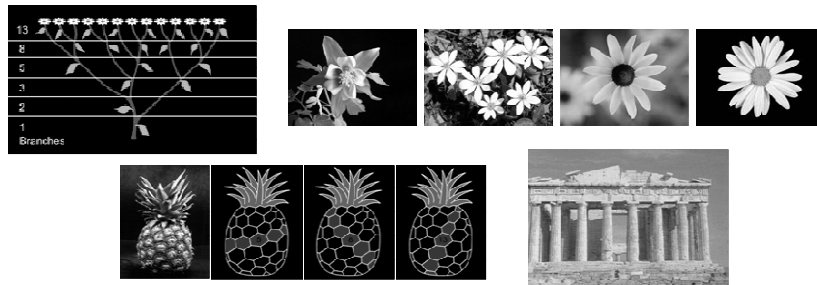
- Definition: $n! = n(n-1)(n-2)\dots(2)(1)$
- Since $(n-1)(n-2)\dots(2)(1) = (n-1)!$
 - $n! = n(n-1)!$, for $n > 0$
 - $n! = 1$ for $n = 0$ (base case)
- Example:
$$\begin{array}{r} 4! = 4(3!) \\ \quad 3! = 3(2!) \\ \quad \quad 2! = 2(1!) \\ \quad \quad \quad 1! = 1(0!) = 1(1) = 1 \end{array} \qquad \begin{array}{r} = 4(6) = 24 \\ = 3(2) = 6 \\ = 2(1) = 2 \end{array}$$

Factorial in Ruby

```
def factorial(n)
  if n == 0 then
    return 1
  else
    return n * factorial(n-1)
  end
end
```

Fibonacci Numbers

- A sequence of numbers such that each number is the sum of the previous two numbers in the sequence, starting the sequence with 0 and 1.
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, etc.



15110 Principles of Computing,
Carnegie Mellon University - CORTINA

7

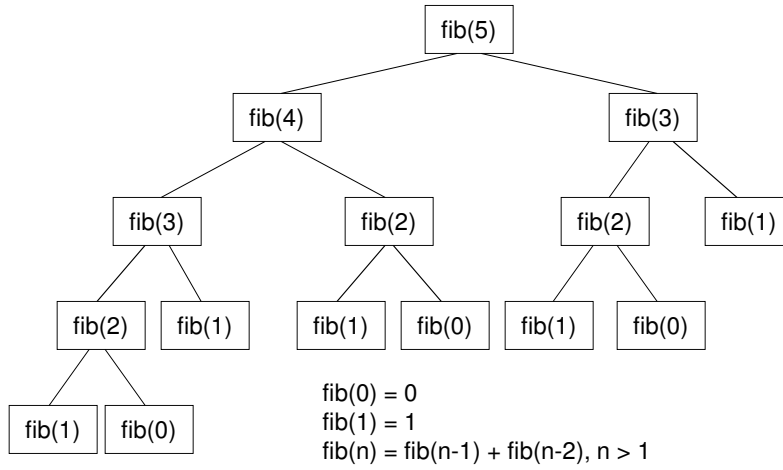
Recursive Definition

- Let $\text{fib}(n)$ = the n^{th} Fibonacci number, $n \geq 0$
 - $\text{fib}(0) = 0$ (base case)
 - $\text{fib}(1) = 1$ (base case)
 - $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$, $n > 1$

15110 Principles of Computing,
Carnegie Mellon University - CORTINA

8

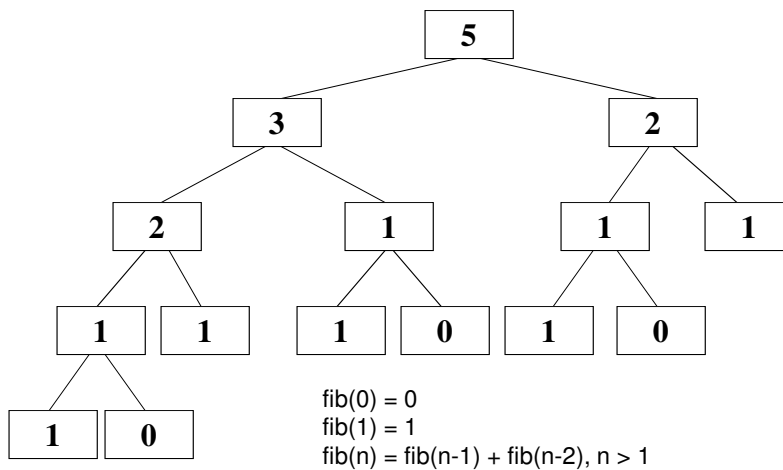
Recursive Definition



15110 Principles of Computing,
Carnegie Mellon University - CORTINA

9

Recursive Definition



15110 Principles of Computing,
Carnegie Mellon University - CORTINA

10

Fibonacci Numbers in Ruby

```
def fib(n)
  if n == 0 || n == 1 then
    return n
  else
    return fib(n-1) + fib(n-2)
  end
end
```

logical operator OR:
at least 1 condition
must be true

In irb:

```
for i in 0..30 do puts fib(i) end
```

Why does it take longer to print each subsequent value?

Computing the sum of a list

```
def sumlist(list)
  n = list.length
  if n == 0 then
    return 0
  else
    return list[0] + sumlist(list[1..n-1])
  end
end
```

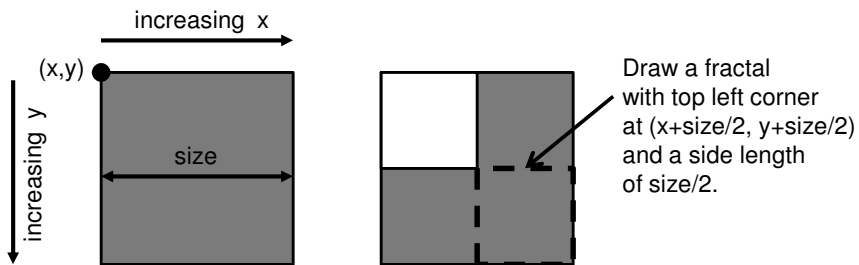
Base case:
The sum of an empty list is 0.

Recursive case:
The sum of a list is the first element +
the sum of the rest of the list.

Simple Fractal

To draw a fractal with top-left corner (x,y) and a side length of size:

- Draw a white square with top-left corner (x,y) and a side length of $size/2$.
- Draw another fractal with top-left corner $(x+size/2, y+size/2)$ and a side length of $size/2$. [recursive step]



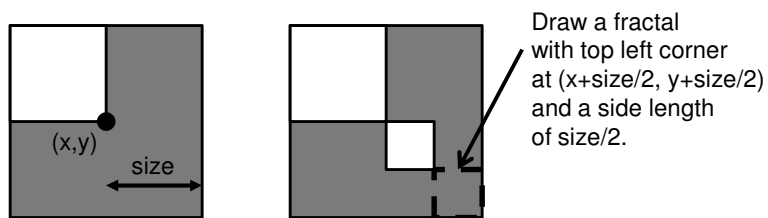
15110 Principles of Computing,
Carnegie Mellon University - CORTINA

13

Simple Fractal

To draw a fractal with top-left corner (x,y) and a side length of size:

- Draw a white square with top-left corner (x,y) and a side length of $size/2$.
- Draw another fractal with top-left corner $(x+size/2, y+size/2)$ and a side length of $size/2$. [recursive step]



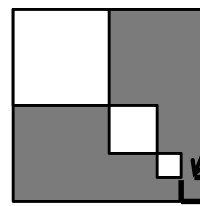
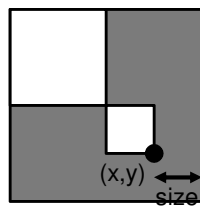
15110 Principles of Computing,
Carnegie Mellon University - CORTINA

14

Simple Fractal

To draw a fractal with top-left corner (x,y) and a side length of size:

- Draw a white square with top-left corner (x,y) and a side length of $size/2$.
- Draw another fractal with top-left corner $(x+size/2, y+size/2)$ and a side length of $size/2$. [recursive step]



Draw a fractal with top left corner at $(x+size/2, y+size/2)$ and a side length of $size/2$.

15110 Principles of Computing,
Carnegie Mellon University - CORTINA

15

Simple Fractal in Ruby

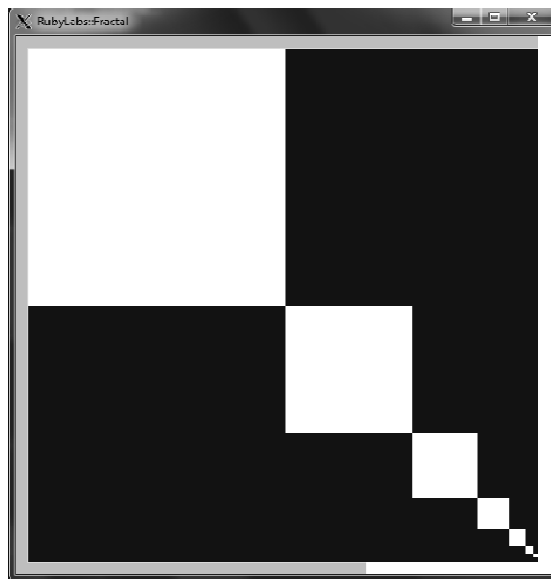
(not all code shown)

```
def fractal(x, y, size)
  return if size < 2          # base case
  draw_square(x, y, size/2)
  fractal(x+size/2, y+size/2, size/2)
end
```

```
def draw_fractal()
  # initial top-left (x,y) and size
  fractal(0, 0, 512)
end
```

15110 Principles of Computing,
Carnegie Mellon University - CORTINA

16

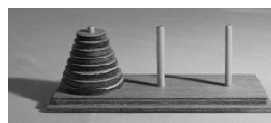


15110 Principles of Computing,
Carnegie Mellon University - CORTINA

17

Towers of Hanoi

- A puzzle invented by French mathematician Edouard Lucas in 1883.
- At a temple far away, priests were led to a courtyard with three pegs and 64 discs stacked on one peg in size order.
 - Priests are only allowed to move one disc at a time from one peg to another.
 - Priests may not put a larger disc on top of a smaller disc at any time.
- The goal of the priests was to move all 64 discs from the leftmost peg to the rightmost peg.
- According to the story, the world would end when the priests finished their work.



Towers of Hanoi
with 8 discs.

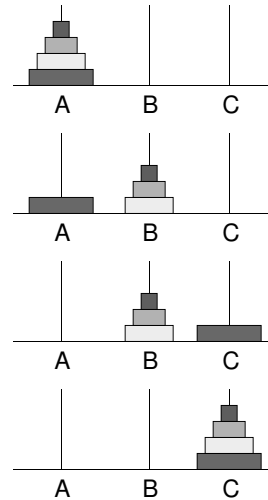
15110 Principles of Computing,
Carnegie Mellon University - CORTINA

18

Towers of Hanoi

Problem: Move n discs
from peg A to peg C using peg B.

1. Move $n-1$ discs from peg A to peg B using peg C. (recursive step)
2. Move 1 disc from peg A to peg C.
3. Move $n-1$ discs from peg B to C using peg A. (recursive step)



15110 Principles of Computing,
Carnegie Mellon University - CORTINA

19

Towers of Hanoi in Ruby

```
def towers(n, from_peg, to_peg, using_peg)
  if n >= 1 then
    towers(n-1, from_peg, using_peg, to_peg)
    puts "Move disc from " + from_peg
      + " to " + to_peg
    towers(n-1, using_peg, to_peg, from_peg)
  end
end
```

In irb: `towers(4, "A", "C", "B")`

How many moves do the priests need to move 64 discs?

15110 Principles of Computing,
Carnegie Mellon University - CORTINA

20