UNIT 5A
Recursion: Basics

Recursion

• A “recursive” function is one that calls itself.
• Infinite loop? Not necessarily.
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”. 

Example: Factorial

• \( n! = n \times (n-1) \times (n-2) \times \cdots \times 1 \)

• \( 2! = 2 \times 1 \)
• \( 3! = 3 \times 2 \times 1 \)
• \( 4! = 4 \times 3 \times 2 \times 1 \)
• So \( 4! = 4 \times 3! \)
• And \( 3! = 3 \times 2! \)
• What is the base case? \( 0! = 1 \)
How Recursion Works

Base case

4! = 4(3!) = 4(6) = 24
3! = 3(2!) = 3(2) = 6
2! = 2(1!) = 2(1) = 2
1! = 1 (0!) = 1(1) = 1

make smaller instances of the same problem

build up the result

Factorial in Ruby (Recursive)

```ruby
def factorial (n)
    if n == 0  % base case
        return 1
    else  % recursive case
        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.

Fibonacci Numbers in Nature

• 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, etc.
• Number of branches on a tree.
• Number of petals on a flower.
• Number of spirals on a pineapple.
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)
  - $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$, $n > 1$

Recursive Fibonacci in Ruby

```ruby
def fib(n)
  if n == 0 or n == 1
    return n
  else
    return fib(n-1) + fib(n-2)
  end
end
```
Recursive Definition

Recursive vs. Iterative Solutions

- For every recursive function, there is an equivalent iterative solution.
- For every iterative function, there is an equivalent recursive solution.
- But some problems are easier to solve one way than the other way.
Factorial Function (Iterative)

```python
def factorial (n):
    result = 1
    for i in 1..n do
        result = result * i
    end
    return result
end
```

Iterative Fibonacci

```python
def fib(n):
    x = 0
    next_x = 1
    for i in 1..n do
        x, next_x = next_x, x+next_x
    end
    return x
end
```

Much faster than the recursive version. Why?
Recursive sum of a list

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

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.

Geometric Recursion (Fractals)

- A recursive operation performed on successively smaller regions.

Sierpinski's Triangle

http://fusionanomaly.net/recursion.jpg
Sierpinski’s Triangle

Sierpinski’s Carpet
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]
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]

Simple Fractal in Ruby

(not all code shown)

```ruby
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
```
Next Lecture

- Binary search: Apply the technique of recursion in doing search
- Analyze its time complexity
Famous Puzzle of “Towers of Hanoi”

**EXTRA RECURSION EXAMPLE**

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

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)

Towers of Hanoi in Ruby

```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?