Algorithms and Recursion
Chapter 14

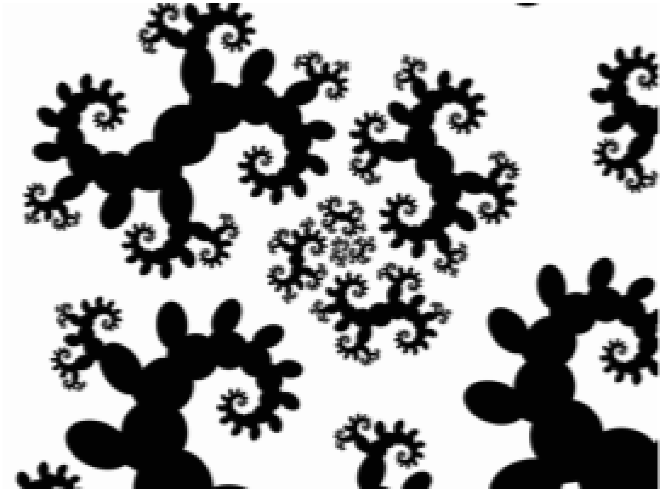
Course: CPSC 1150
Instructor: Dr. Bita Shadgar

Lecture 21

Learning Outcomes

- Define recursive algorithms
- Trace a recursive algorithm

What do these images have in common?



$$\gcd(p, q) = \begin{cases} p & \text{if } q = 0 \\ \gcd(q,\ p \,\%\, q) & \text{otherwise} \end{cases}$$

$$\begin{aligned} \gcd(4032, 1272) &= \gcd(1272, 216) \\ &= \gcd(216, 192) \\ &= \gcd(192, 24) \\ &= \gcd(24, 0) \\ &= 24. \end{aligned}$$

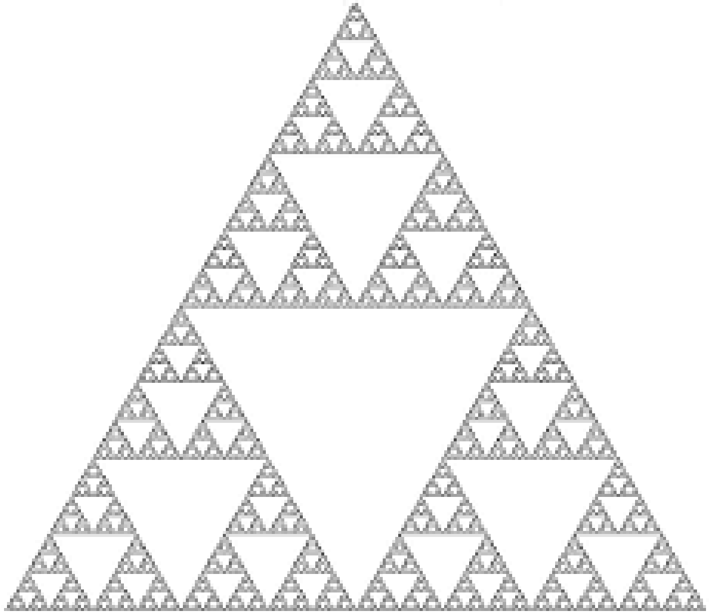
What is recursion?

Definition

Recursion is a technique “where the solution to a problem depends on solutions to smaller instances of the same problem.”

Recursive problem solving

– Recursion can be used to solve a problem, P, by decomposing P into simple problem, S, and a ‘smaller’ copy of the original problem, P. In some sense, P = S + P

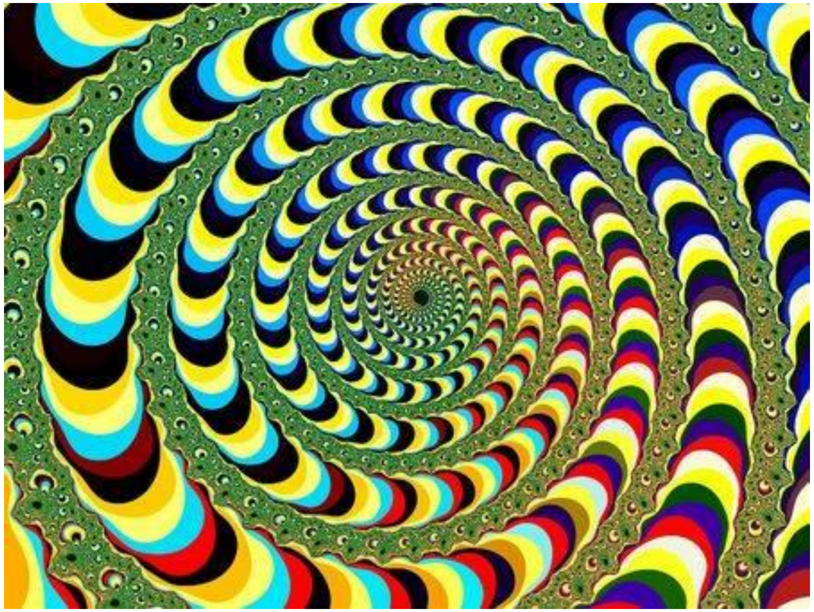


Example

Recall: n! = $\prod_{i=1}^{n} i$ = 1 × 2 × · · · × (n − 1) × n.
To compute n! recursively, can decompose it into n × (n − 1)!

Base cases

– To use recursion in programming, we write a recursive method
  - a method that calls itself



– **Question:** If a method keeps calling itself, won't this just go on forever?

– A recursive method needs base cases

– For some value(s) of the input, the method must not call itself

– All invocations of the method must change the size of the problem, coming closer to a base case

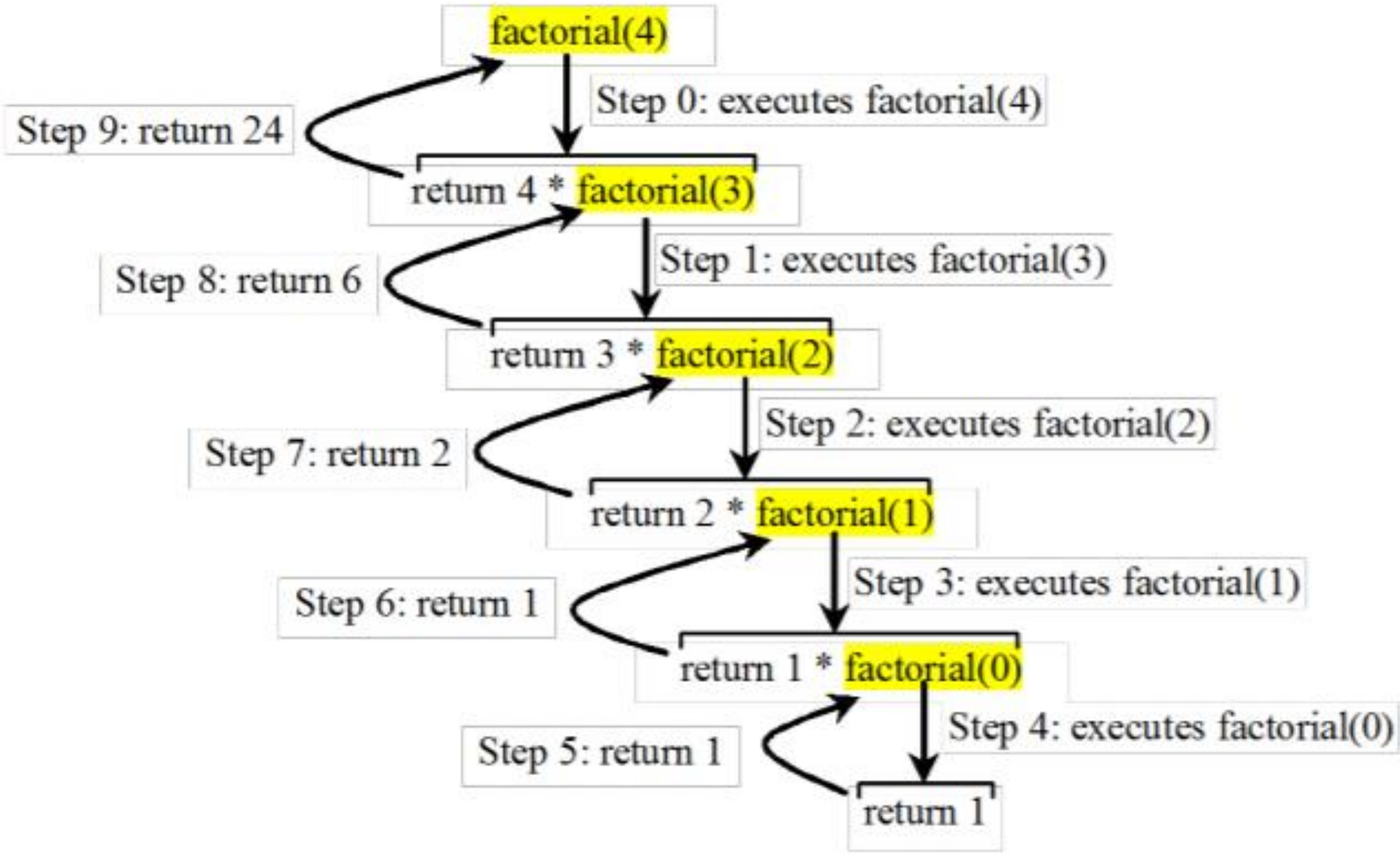
Factorial example

– **Question:** What is the base case when computing a factorial recursively?

Example

```
public class TestRecursive{
    public static void main(String []args){
        System.out.println(factorial(4));
    }
    public static int factorial(int n){
        if (n==0) //base case
            return 1;
        return n * factorial(n-1);
    }
}
```

Tracing recursion



Helper method to call recursive method

- Sometimes it's useful to have a method which sets up recursion, but is not actually recursive itself
- This higher level method will then invoke a recursive method
- The recursive method usually has the same name as the other method, but extra parameters to aid the recursion

Example

See RecursivePalindrome and RecursiveBinarySearch.

Efficiency of recursion

- Each time a recursive method invokes itself, a new activation record is created on the stack
  - This can take up a lot of **memory**, especially if a method calls itself several times, like in the Sierpinski triangle example
  - To avoid this, it is often better to use a loop than a recursive method
- A recursive program has to ‘pop’ all the way back up after hitting the base case
  - This can take a lot of **time**
  - Can usually rewrite a recursive method to be tail-recursive
    - Allows some compilers to optimize for speed by returning a value directly from the base case

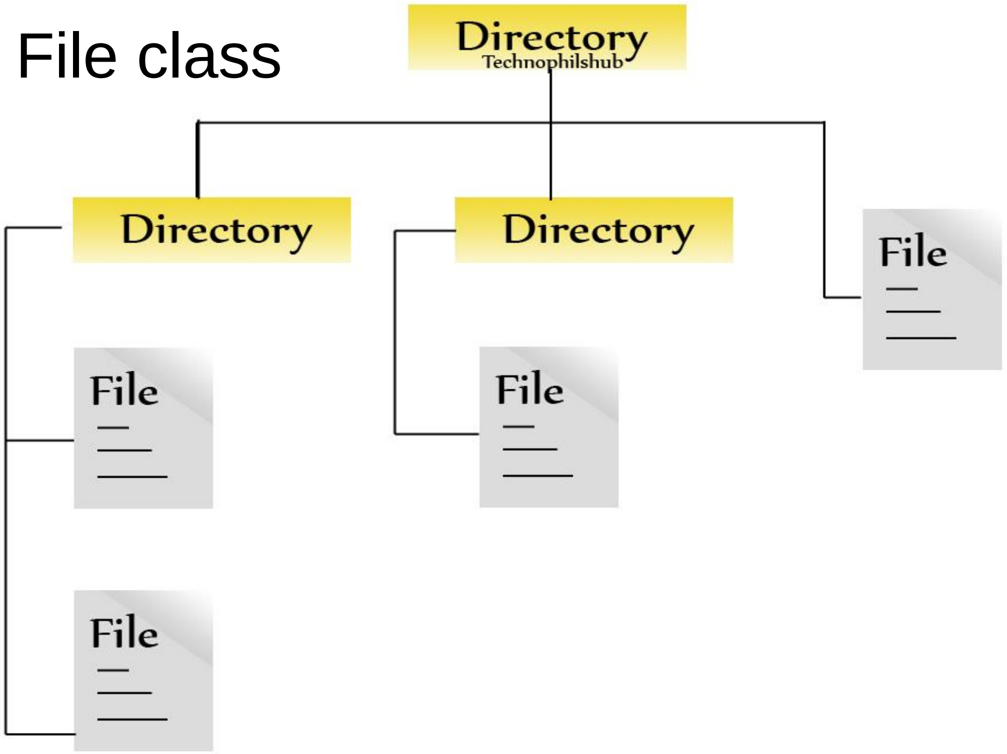
Why recursive methods?

Recursive methods are efficient for solving problems with recursive structures.

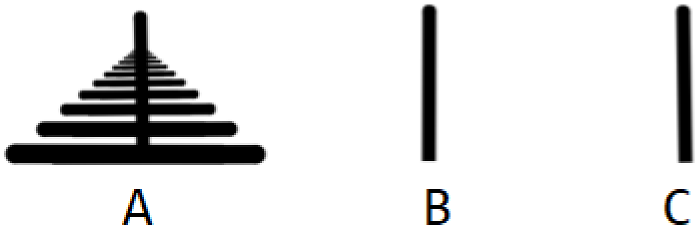
– Example – Finding the directory size

Some useful methods in File class

- isFile()
- isDirectory()
- length()
- listFiles()



Tower of Hanoi



Problem

Follow the rules to move all the disks to the second tower.

- You must move one disk at a time, from one tower to another.
- You may only move a disk if it is at the top of a tower.
- You may never place a disk on a smaller disk.

– **Question:** How many moves does it take with just 3 disks?

– **Question:** Can you think of a recursive solution to the problem for N disks?

More fun with recursion!

Check out the following resources, if you are interested in knowing more about recursions:

– The story "Little Harmonic Labyrinth" from the book G¨odel, Escher, Bach: An Eternal Golden Braid, by Douglas Hofstadter
– The website http://recursivedrawing.com/ – that's where I made a lot of the images in these slides