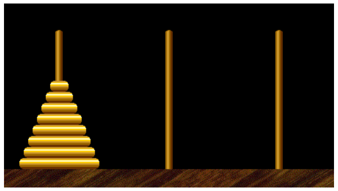
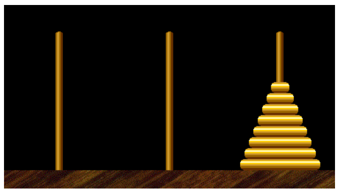
Lecture 7: Recursion



Start



Finish

Overview

What is recursion?

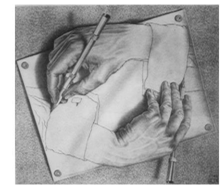
- When one function calls ITSELF directly or indirectly.

Why learn recursion?

- New mode of thinking.
- Powerful programming tool.
- Divide-and-conquer paradigm.

Many computations are naturally self-referential.

- A directory contains files and other directories.
- Euclid's gcd algorithm.
- Quicksort.
- Linked data structures.



Drawing Hands
M. C. Escher, 1948

Greatest Common Divisor

Find largest integer d that evenly divides into p and q.

Example:

- Suppose p = 32 and q = 24
- Integers that evenly divide both p and q: 1, 2, 4, 8
  - So d = 8 (the largest)

How would you compute gcd ?

Greatest Common Divisor

Find largest integer d that evenly divides into p and q.

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

⇐ base case

⇐ reduction step, converges to base case

gcd(4032, 1272) = gcd(1272, 216)
= gcd(216, 192)
= gcd(192, 24)
= gcd(24, 0)
= 24.

$4032 = 2^6 \times 3^2 \times 7^1$

$1272 = 2^3 \times 3^1 \times 53^1$

$\text{gcd} = 2^3 \times 3^1 = 24$

Applications.

- Simplify fractions: 1272/4032 = 53/168.
- RSA cryptosystem. stay tuned
- History of algorithms.

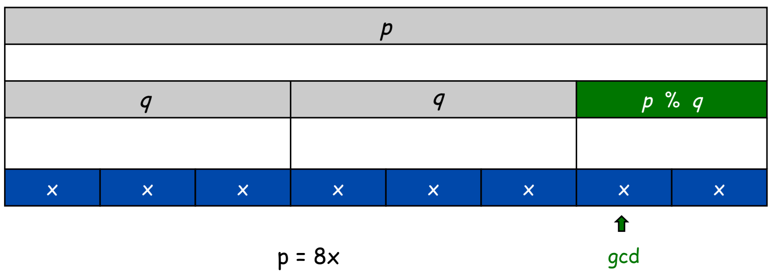
Euclid, 300 BCE

Greatest Common Divisor

Find largest integer d that evenly divides into p and q.

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

← base case
← reduction step, converges to base case



p = 8x
q = 3x
gcd(p, q) = x

Greatest Common Divisor

Find largest integer d that evenly divides into p and q.

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

← base case
← reduction step, converges to base case

Java implementation.

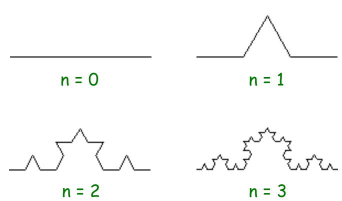
```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

← base case
← reduction step

Koch Snowflake

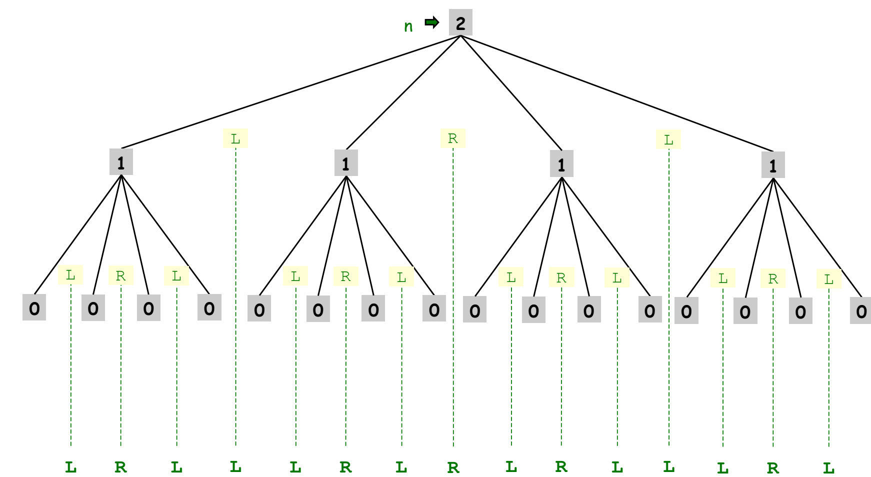
Koch curve of order n.

- Draw curve of order n-1.
- Turn 60°.
- Draw curve of order n-1.
- Turn -120°.
- Draw curve of order n-1.
- Turn 60°.
- Draw curve of order n-1.




```
public static void koch(int n, double size) {
   if (n == 0) StdDraw.goForward(size);
   else {
      koch(n-1, size);
      StdDraw.rotate(60);
      koch(n-1, size);
      StdDraw.rotate(-120);
      koch(n-1, size);
      StdDraw.rotate(60);
      koch(n-1, size);
   }
}
```

Koch Curve: Recursion Tree

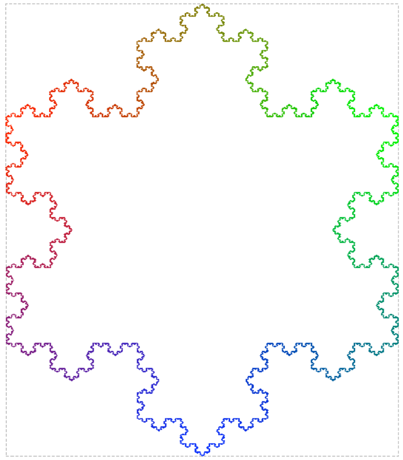


L: Rotate 60°
R: Rotate -120°

Koch Snowflake

Koch snowflake of order n.

- Draw Koch curve of order n.
- Turn -120°.
- Draw Koch curve of order n.
- Turn -120°.
- Draw Koch curve of order n.



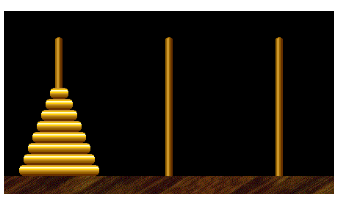
Koch Snowflake in Java

```java
public class Koch {
   public static void koch(int n, double size) { }     ⇐ just did this

   public static void main(String args[]) {
      int N = Integer.parseInt(args[0]);                  ⇐ compute parameters
      int width = 512;
      int height = (int) (2 * width / Math.sqrt(3));
      double size = width / Math.pow(3.0, N);

      StdDraw.create(width, height);
      StdDraw.go(0, width * Math.sqrt(3) / 2);
   ⇒  StdDraw.penDown();
      koch(N, size);
      StdDraw.rotate(-120);
      koch(N, size);
      StdDraw.rotate(-120);
      koch(N, size);
      StdDraw.show();                                     draw it
   }
}
```

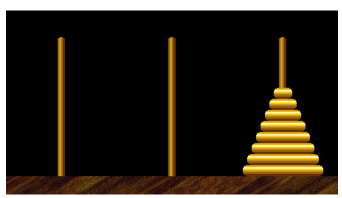
Towers of Hanoi

Move all the discs from the leftmost peg to the rightmost one.

- Only one disc may be moved at a time.
- A disc can be placed either on empty peg or on top of a larger disc.



Start

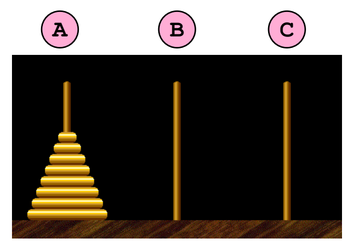


Finish

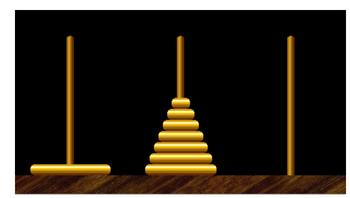
Towers of Hanoi demo

Edouard Lucas (1883)

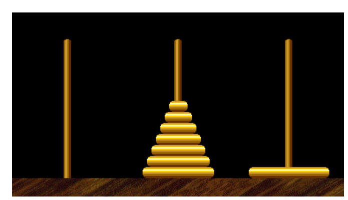
Towers of Hanoi: Recursive Solution



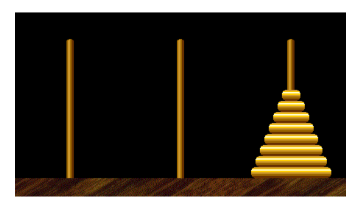
Move N-1 smallest discs to pole B.



Move largest disc to pole C.



Move N-1 smallest discs to pole C.



Towers of Hanoi Legend

Is world going to end (according to legend)?

- 40 golden discs on 3 diamond pegs.
- World ends when certain group of monks accomplish task.

Will computer algorithms help?

Towers of Hanoi: Recursive Solution

```java
public class Hanoi {

   public static void hanoi(int n, String from, String temp,
                                   String to) {
      if (n == 0) return;
      hanoi(n-1, from, to, temp);
      System.out.println("Move disc " + n +
                         " from " + from + " to " + to);
      hanoi(n-1, temp, from, to);
   }

   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      hanoi(N, "A", "B", "C");
   }
}
```

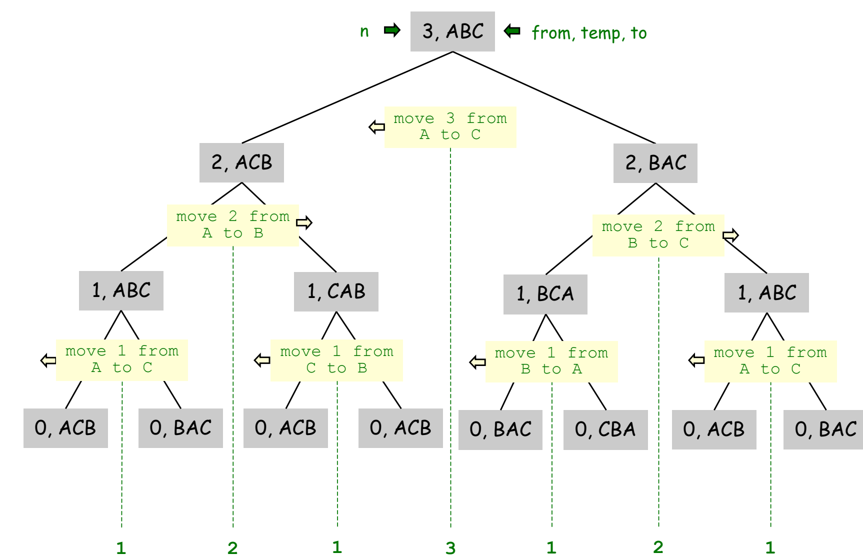
Towers of Hanoi: Recursive Solution

```
% java Hanoi 3
Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C
```

```
% java Hanoi 4
Move disc 1 from A to B
Move disc 2 from A to C
Move disc 1 from B to C
Move disc 3 from A to B
Move disc 1 from C to A
Move disc 2 from C to B
Move disc 1 from A to B
Move disc 4 from A to C
Move disc 1 from B to C
Move disc 2 from B to A
Move disc 1 from C to A
Move disc 3 from B to C
Move disc 1 from A to B
Move disc 2 from A to C
Move disc 1 from B to C
```

subdivisions of ruler

Towers of Hanoi: Recursion Tree



Properties of Towers of Hanoi Solution

Remarkable properties of recursive solution.

- Takes $2^N - 1$ steps to solve N disc problem.
- Sequence of discs is same as subdivisions of ruler.
- Smallest disc always moves in same direction.

Recursive algorithm yields non-recursive solution!

- Alternate between two moves:
  - move smallest disc to right (left) if N is even (odd)
  - make only legal move not involving smallest disc

Recursive algorithm may reveal fate of world.

- Takes 348 centuries for N = 40, assuming rate of 1 disc per second.
- Reassuring fact: ANY solution takes at least this long!

Divide-and-Conquer

Divide-and-conquer paradigm.

- Break up problem into one or more smaller subproblems of similar structure.
- Solve subproblems recursively using same method.
- Combine results to produce solution to original problem.

Historical origins.

- Julius Caesar (100 BCE - 44 BCE).
- "*Divide et impera*."
- "*Veni, vidi, vici*."

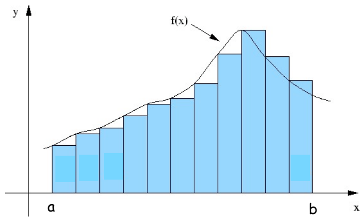
Many problems have elegant divide-and-conquer solutions.

- Adaptive quadrature.
- Sorting. quicksort (stay tuned)

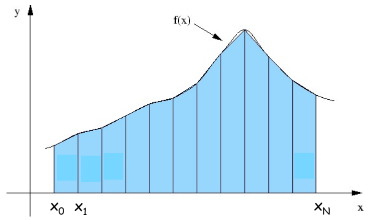
Numerical Integration

Integrate a smooth function f(x) from x = a to b.

- Quadrature: subdivide interval from a to b into tiny pieces, approximate area under curve in each piece, and compute sum.
- Rectangle rule: approximate using constant functions.
- Trapezoid rule: approximate using linear functions.



Rectangle Rule



Trapezoid Rule

$$S_N = h\left[\tfrac{1}{2}f(x_0) + f(x_1) + f(x_2) + \ldots + f(x_{N-1}) + \tfrac{1}{2}f(x_N)\right]$$

$$x_k = a + hk,\ h = \tfrac{b-a}{N}$$

Trapezoid Rule

```
public class Integration {

   static double f(double x) {
      return Math.exp(-x*x / 2) / Math.sqrt(2 * Math.PI);
   }

   static double trapezoid(double a, double b, int N) {
      double h   = (b - a) / N;
      double sum = 0.5 * h * (f(a) + f(b));
      for (int k = 1; k < N; k++)
         sum = sum + h * f(a + h*k);
      return sum;
   }

   public static void main(String[] args) {
      System.out.println(trapezoid(-3.0, 3.0, 1000));
   }

}
```

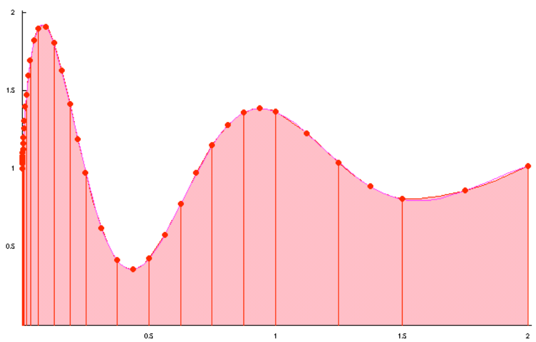
a function to integrate

number of interval subdivisions

Adaptive Quadrature

Numerical quadrature: approximate area under a curve from a to b.

- Subdividing into subintervals and approximate area in each piece.
- Trapezoid: fixed number of equally spaced subintervals.
- Adaptive quadrature: variable number of subintervals that adapt to shape of curve.



Adaptive Quadrature

To approximate area of curve from a to b:

- Approximate area from a to b using two quadrature methods.
- If nearly equal, return area.
  Otherwise
  - subdivide interval into two equal pieces
  - compute area of each piece recursively
  - return sum

```
static double adaptive(double a, double b) {
   double area  = trapezoid(a , b , 10);
   double check = trapezoid(a , b ,  5);
   if (Math.abs(area - check) > 0.00000000001) {
      double m = (a + b) / 2;
      area = adaptive(a, m) + adaptive(m, b);
   }
   return area;
}
```

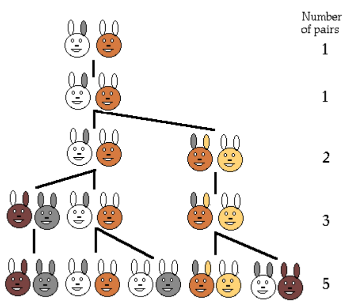
Fibonacci Numbers

Infinite series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .

- A natural for recursion.

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

Fibonacci Rabbits:



L. P. Fibonacci
(1170 - 1250)

Possible Pitfalls With Recursion

Is recursion fast?

- Yes. We produced remarkably efficient program for gcd.
- No. Can easily write remarkably inefficient programs.

Fibonacci numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

Observation: it takes a really long time to compute F(40).

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

Spectacularly inefficient Fibonacci

Possible Pitfalls With Recursion

F(39) is computed once.

F(38) is computed 2 times.

F(37) is computed 3 times.

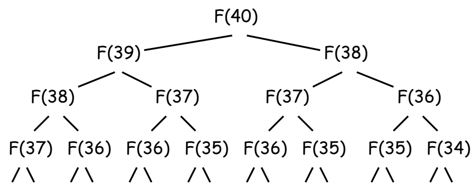
F(36) is computed 5 times.

F(35) is computed 8 times.

...

F(0) is computed 165,580,141 times.

331,160,281 function calls for F(40).




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

Spectacularly inefficient Fibonacci

Possible Pitfalls With Recursion

Recursion can take a long time if it needs to repeatedly recompute intermediate results.

- Dynamic programming solution.
  - save away intermediate results in a table
  - stay tuned: genetic sequence alignment assignment

Summary

How to write simple recursive programs?

- Base case, reduction step.
- Trace the execution of a recursive program.
- Use pictures.

Why learn recursion?

- New mode of thinking.
- Powerful programming tool.

Many problems have elegant divide-and-conquer solutions.

- Adaptive quadrature.
- Quicksort.
- Integer arithmetic for RSA cryptography.
- Polynomial multiplication for signal processing.
- Quad-tree for efficient N-body simulation.