Overview

What is recursion? When one function calls itself directly or indirectly.

Why learn recursion?
• New mode of thinking.
• Powerful programming paradigm.

Many computations are naturally self-referential.
• Binary search, mergesort, FFT, GCD.
• Linked data structures.
• A folder contains files and other folders.

Closely related to mathematical induction.

Greatest Common Divisor

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

Ex. gcd(4032, 1272) = 24.

\[
\begin{align*}
4032 &= 2^6 \times 3^2 \times 7^1 \\
1272 &= 2^3 \times 3^1 \times 53^1 \\
gcd &= 2^3 \times 3^1 = 24
\end{align*}
\]

Applications.
• Simplify fractions: 1272/4032 = 53/168.
• RSA cryptosystem.
Mathematical Induction

**Mathematical induction.** Prove a statement involving an integer N by

- **base case:** Prove it for some specific N (usually 0 or 1).
- **induction step:** Assume it to be true for all positive integers less than N, use that fact to prove it for N.

**Ex.** Sum of the first N odd integers is N^2.

**Base case:** True for N = 1.

**Induction step:**

- Let T(N) be the sum of the first N odd integers: 1 + 3 + 5 + ... + (2N - 1).
- Assume that T(N-1) = (N-1)^2.
- T(N) = T(N-1) + (2N - 1)
  = (N-1)^2 + (2N - 1)
  = N^2 - 2N + 1 + (2N - 1)
  = N^2

Greatest Common Divisor

**GCD.** Find largest integer that evenly divides into p and q.

**Euclid’s algorithm.** [Euclid 300 BCE]

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

\[
\text{gcd}(4032, 1272) = \text{gcd}(1272, 216) = \text{gcd}(216, 192) = \text{gcd}(192, 24) = \text{gcd}(24, 0) = 24.
\]

4032 = 3 * 1272 + 216

Euclid's Algorithm

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

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

**Recursive Program.** Implement a function having integer arguments by

- **base case:** Implementing it for some specific values of the arguments.
- **reduction step:** Assume the function works for smaller values of its arguments and use it to implement it for the given values.

**Ex. gcd(p, q).**

**Base case:** gcd(p, 0) = p.

**Reduction step:** gcd(p, q) = gcd(p, p-q) if p-q > 0
  = gcd(p, p-2q) if p-2q > 0
  ...
  = gcd(p, p \% q)

\[
p = 8x \\
q = 3x
\]

\[
gcd(p, q) = gcd(3x, 2x) = x
\]
Euclid’s Algorithm

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

\[ \text{gcd}(p, q) = \begin{cases} 
  p & \text{if } q = 0 \\
  \text{gcd}(q, p \mod q) & \text{otherwise}
\end{cases} \]

---

Recursive program

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

---

Recursive Graphics
H-tree of order $n$.
- Draw an H.
- Recursively draw 4 H-trees of order $n-1$, one connected to each tip.

<table>
<thead>
<tr>
<th>order 1</th>
<th>order 2</th>
<th>order 3</th>
</tr>
</thead>
</table>

Animated H-tree. Pause for 1 second after drawing each H.

Divide-and-Conquer
- Break up problem into smaller subproblems of same structure.
- Solve subproblems recursively using same method.
- Combine results to produce solution to original problem.

Many important problems succumb to divide-and-conquer.
- FFT for signal processing.
- Parsers for programming languages.
- Multigrid methods for solving PDEs.
- Quicksort and mergesort for sorting.
- Hilbert curve for domain decomposition.
- Quad-tree for efficient N-body simulation.
- Midpoint displacement method for fractional Brownian motion.
Application: Fractional Brownian Motion

Fractional Brownian Motion

Physical process which models many natural and artificial phenomenon.
- Price of stocks.
- Dispersion of ink flowing in water.
- Rugged shapes of mountains and clouds.
- Fractal landscapes and textures for computer graphics.

Simulating Brownian Motion

Midpoint displacement method.
- Maintain an interval with endpoints \((x_0, y_0)\) and \((x_1, y_1)\).
- Divide the interval in half.
- Choose \(\delta\) at random from Gaussian distribution.
- Set \(x_m = (x_0 + x_1)/2\) and \(y_m = (y_0 + y_1)/2 + \delta\).
- Recur on the left and right intervals.

Simulating Brownian Motion: Java Implementation

Midpoint displacement method.
- Maintain an interval with endpoints \((x_0, y_0)\) and \((x_1, y_1)\).
- Choose \(\delta\) at random from Gaussian distribution.
- Divide the interval in half: Set \(x_m = (x_0 + x_1)/2\) and \(y_m = (y_0 + y_1)/2 + \delta\).
- Recur on the left and right intervals.

```java
public static void curve(double x0, double y0, double x1, double y1, double var) {
    if (x1 - x0 < 0.01) {
        StdDraw.line(x0, y0, x1, y1);
        return;
    }
    double xm = (x0 + x1) / 2;
    double ym = (y0 + y1) / 2;
    ym += StdRandom.gaussian(0, Math.sqrt(var));
    curve(x0, y0, xm, ym, var/2);
    curve(xm, ym, x1, y1, var/2);
}
```

variance halves at each level; change factor to get different shapes
Plasma Cloud

Plasma cloud centered at (x, y) of size s.
- Each corner labeled with some grayscale value.
- Divide square into four quadrants.
- The grayscale of each new corner is the average of others.
  - center: average of the four corners + random displacement
  - others: average of two original corners
- Recur on the four quadrants.

Brownian Landscape


Towers of Hanoi

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.

Towers of Hanoi demo

Edouard Lucas (1883)

Towers of Hanoi Legend

Q. Is world going to end (according to legend)?
- 64 golden discs on 3 diamond pegs.
- World ends when certain group of monks accomplish task.

Q. Will computer algorithms help?

Towers of Hanoi: Recursive Solution

```java
public class TowersOfHanoi {
    public static void moves(int n, boolean left) {
        if (n == 0) return;
        moves(n - 1, !left);
        if (left) System.out.println(n + " left");
        else System.out.println(n + " right");
        moves(n - 1, !left);
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        moves(N, true);
    }
}
```

moves(n, true): move discs 1 to n one pole to the left
moves(n, false): move discs 1 to n one pole to the right
Towers of Hanoi: Recursive Solution

```java
% java TowersOfHanoi 3
1 left
2 right
1 left
1 left
2 right
1 right

% java TowersOfHanoi 4
1 right
2 left
1 right
1 right
3 right
1 right
2 left
1 right
4 left
1 right
2 left
1 right
3 right
1 right
2 left
1 right
1 right
```

Towers of Hanoi: Recursion Tree

- Every other move is smallest disc.
- Subdivisions of ruler.

Towers of Hanoi: Properties of Solution

Remarkable properties of recursive solution.
- Takes $2^n - 1$ moves to solve $n$ disc problem.
- Sequence of discs is same as subdivisions of ruler.
- Every other move involves smallest disc.

Recursive algorithm yields non-recursive solution!
- Alternate between two moves:
  - move smallest disc to right if $n$ is even
  - make only legal move not involving smallest disc

Recursive algorithm may reveal fate of world.
- Takes 585 billion years for $n = 64$ (at rate of 1 disc per second).
- Reassuring fact: any solution takes at least this long!
Fibonacci numbers. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

A Possible Pitfall With Recursion

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}

A natural for recursion?

public static long F(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return F(n-1) + F(n-2);
}

FYI (classical math):

\[ F(n) = \frac{\phi^n - (1 - \phi)^n}{\sqrt{5}} \]

\( \phi = \text{golden ratio} = 1.618 \)

Ex: F(50) = \(1.2 \times 10^{10}\)

Is this an efficient way to compute F(50)?

public static long F(int n) {
    long[] F = new long[51];
    F[0] = 0; F[1] = 1;
    if (n == 1) return 1;
    for (int i = 2; i <= 50; i++)
        F[i] = F[i-1] + F[i-2];
}

Is this an efficient way to compute F(50)?
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.

divide-and-conquer. elegant solution to many important problems.

exponential time.
• easy to specify recursive program that takes exponential time.
• don’t do it unless you plan to (and are working on a small problem).