

6. Recursion



6. Recursion

- **Foundations**
 - A classic example
 - Recursive graphics
 - Avoiding exponential waste
 - Dynamic programming

Overview

Q. What is recursion?

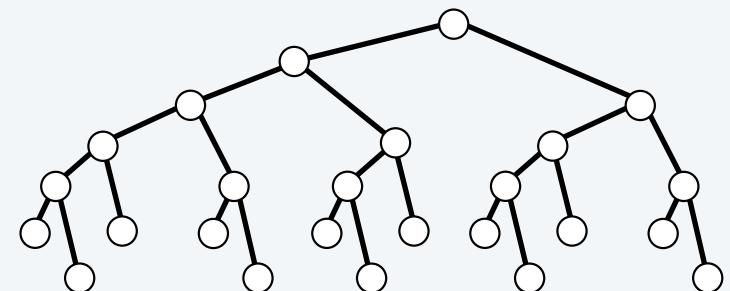
A. When something is specified in terms of *itself*.

Why learn recursion?

- Represents a new mode of thinking.
- Provides a powerful programming paradigm.
- Enables reasoning about correctness.
- Gives insight into the nature of computation.

Many computational artifacts are *naturally* self-referential.

- File system with folders containing folders.
- Fractal graphical patterns.
- Divide-and-conquer algorithms (stay tuned).



Example: Convert an integer to binary

Recursive program

To compute a function of a positive integer N

- **Base case.** Return a value for small N .
- **Reduction step.** Assuming that it works for smaller values of its argument, use the function to compute a return value for N .

```
public class Binary
{
    public static String convert(int N)
    {
        if (N == 1) return "1";
        return convert(N/2) + (N % 2);
    }
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        StdOut.println(convert(N));
    }
}
```

int 0 or 1 automatically converted to String "0" or "1"

Q. How can we be convinced that this method is correct?

A. Use *mathematical induction*.

```
% java Binary 6
110
% java Binary 37
100101
% java Binary 999999
11110100001000111111
```

Mathematical induction (quick review)

To prove a statement involving a positive integer N

- **Base case.** Prove it for some specific values of N .
- **Induction step.** Assuming that the statement is true for all positive integers less than N , use that fact to prove it for N .

Example

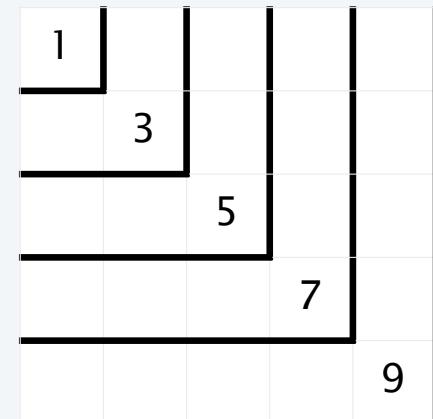
The sum of the first N odd integers is N^2 .

Base case. True for $N = 1$.

Induction step. The N th odd integer is $2N - 1$.

Let $T_N = 1 + 3 + 5 + \dots + (2N - 1)$ be the sum of the first N odd integers.

- Assume that $T_{N-1} = (N-1)^2$.
- Then $T_N = (N-1)^2 + (2N-1) = N^2$.



An alternate proof

Proving a recursive program correct

Recursion

To compute a function of N

- **Base case.** Return a value for small N .
- **Reduction step.** Assuming that it works for smaller values of its argument, use the function to compute a return value for N .

Mathematical induction

To prove a statement involving N

- **Base case.** Prove it for small N .
- **Induction step.** Assuming that the statement is true for all positive integers less than N , use that fact to prove it for N .

Recursive program

```
public static String convert(int N)
{
    if (N == 1) return "1";
    return convert(N/2) + (N % 2);
}
```

Correctness proof, by induction

convert() computes the binary representation of N

- **Base case.** Returns "1" for $N = 1$.
- **Induction step.** Assume that convert() works for $N/2$
 1. Correct to append "0" if N is even, since $N = 2(N/2)$.

$N/2$

--	--	--	--	--	--	--

 N

							0
--	--	--	--	--	--	--	---

2. Correct to append "1" if N is odd since $N = 2(N/2) + 1$.

$N/2$

--	--	--	--	--	--	--

 N

							1
--	--	--	--	--	--	--	---

Mechanics of a function call

System actions when *any* function is called

- *Save environment* (values of all variables and call location).
- *Initialize values* of argument variables.
- *Transfer control* to the function.
- *Restore environment* (and assign return value)
- *Transfer control* back to the calling code.

```
public class Binary
{
    public static String convert(int N)
    {
        if (N == 1) return "1";
        return convert(N/2) + (N % 2);
    }
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        System.out.println(convert(N));
    }
}
```

```
convert(26)
if (N == 1) return "1";
return "1101" + "0";
```

```
convert(13)
if (N == 1) return "1";
return "110" + "1";
```

```
convert(6)
if (N == 1) return "1";
return "11" + "0";
```

```
convert(3)
if (N == 1) return "1";
return "1" + "1";
```

```
convert(1)
if (N == 1) return "1";
return convert(0) + "1";
```

```
% java Convert 26
11010
```

Programming with recursion: typical bugs

Missing base case

```
public static double bad(int N)
{
    return bad(N-1) + 1.0/N;
}
```



No convergence guarantee

```
public static double bad(int N)
{
    if (N == 1) return 1.0;
    return bad(1 + N/2) + 1.0/N;
}
```



Try $N = 2$

Both lead to *infinite recursive loops* (bad news).



need to know
how to stop them
on your computer

Collatz Sequence

Collatz function of N .

- If N is 1, stop.
- If N is even, divide by 2.
- If N is odd, multiply by 3 and add 1.

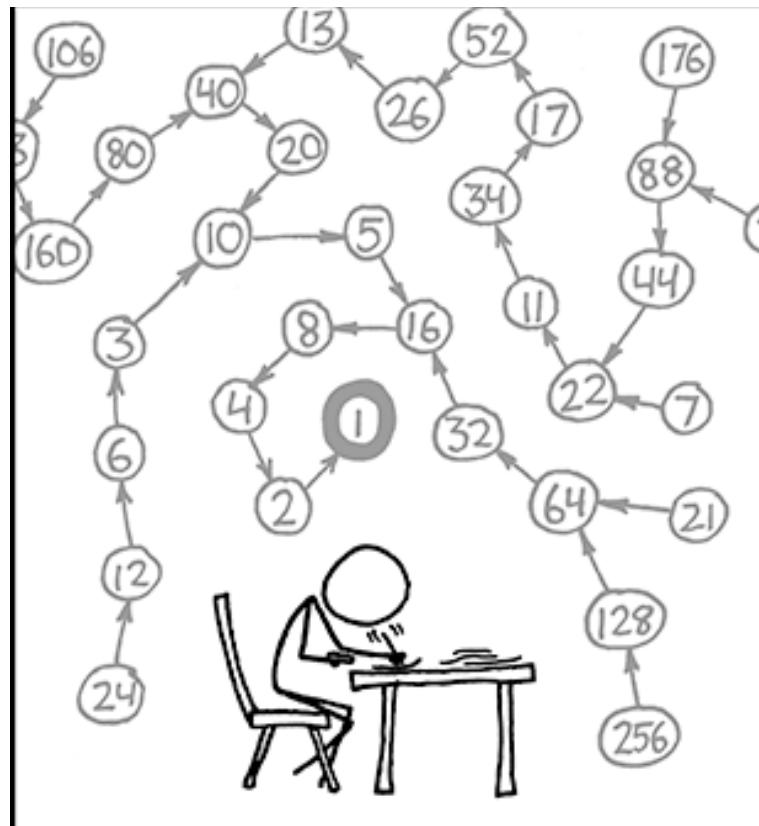
7	22	11	34	17	52	26	13	49	20	...
---	----	----	----	----	----	----	----	----	----	-----

```
public static void collatz(int N)
{
    StdOut.print(N + " ");
    if (N == 1) return;
    if (N % 2 == 0) collatz(N / 2);
    collatz(3*N + 1);
}
```

```
% java Collatz 7
7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

Amazing fact. No one knows whether or not this function terminates for all N (!)

Note. We usually ensure termination by only making recursive calls for smaller N .



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

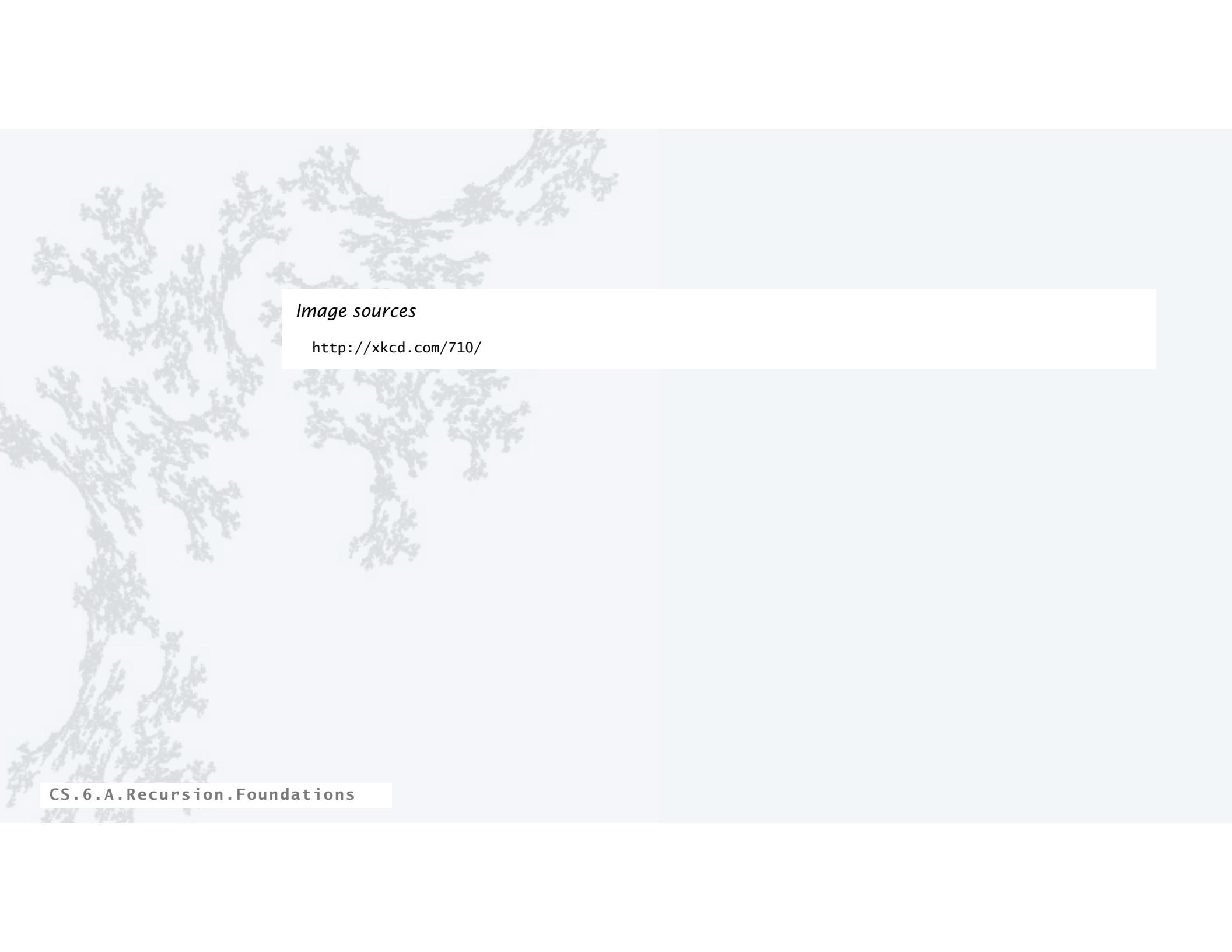


Image sources

<http://xkcd.com/710/>



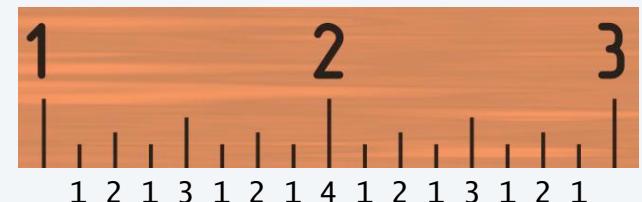
6. Recursion

- Foundations
- A classic example
- Recursive graphics
- Avoiding exponential waste
- Dynamic programming

Warmup: subdivisions of a ruler (revisited)

ruler(n): create subdivisions of a ruler to $1/2^n$ inches.

- Return one space for $n = 0$.
- Otherwise, sandwich n between two copies of ruler($n-1$).



```
public class Ruler
{
    public static String ruler(int n)
    {
        if (n == 0) return " ";
        return ruler(n-1) + n + ruler(n-1);
    }
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        StdOut.println(ruler(n));
    }
}
```

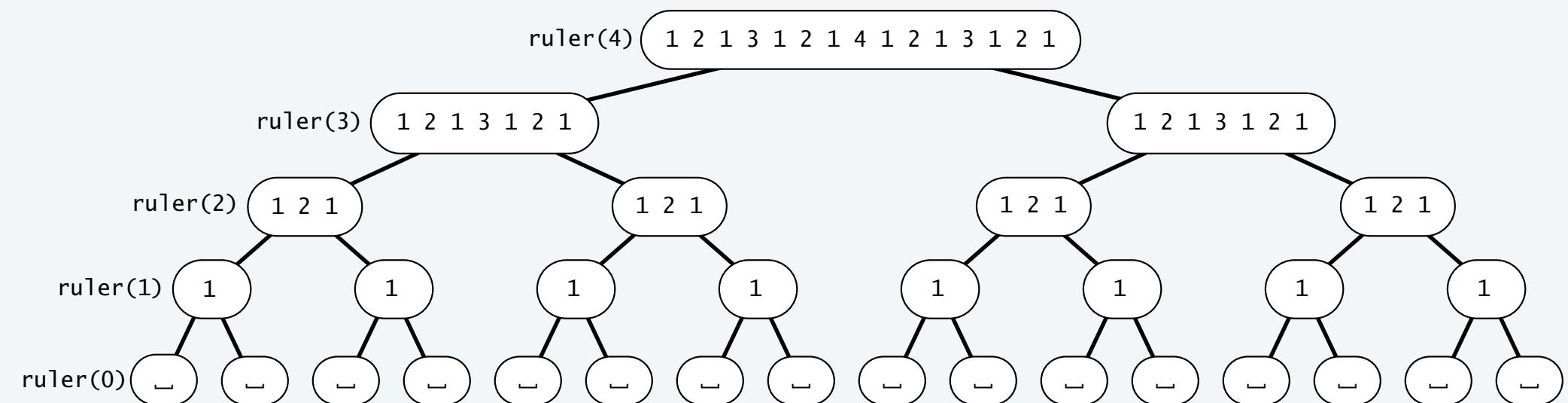
```
% java Ruler 1
1
% java Ruler 2
1 2 1
% java Ruler 3
1 2 1 3 1 2 1
% java Ruler 4
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
% java Ruler 50
Exception in thread "main"
java.lang.OutOfMemoryError:
Java heap space
```

$2^{50} - 1$ integers in output.

Tracing a recursive program

Use a *recursive call tree*

- One node for each recursive call.
- Label node with return value after children are labeled.



Towers of Hanoi puzzle

A legend of uncertain origin

- $n = 64$ discs of differing size; 3 posts; discs on one of the posts from largest to smallest.
- An ancient prophecy has commanded monks to move the discs to another post.
- When the task is completed, *the world will end*.

Rules

- Move discs one at a time.
- Never put a larger disc on a smaller disc.

Q. Generate list of instruction for monks ?

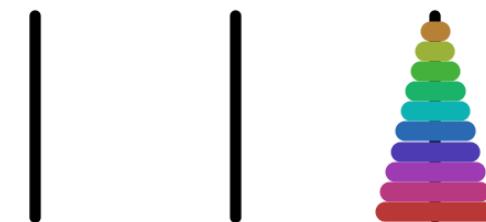
Q. When might the world end ?

$n = 10$

before



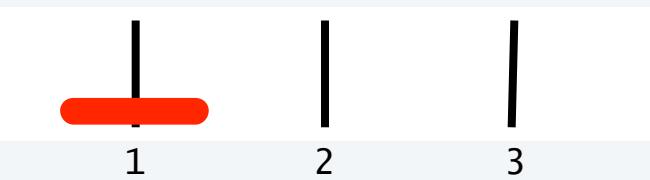
after



Towers of Hanoi

For simple instructions, use cyclic wraparound

- Move *right* means 1 to 2, 2 to 3, or 3 to 1.
- Move *left* means 1 to 3, 3 to 2, or 2 to 1.

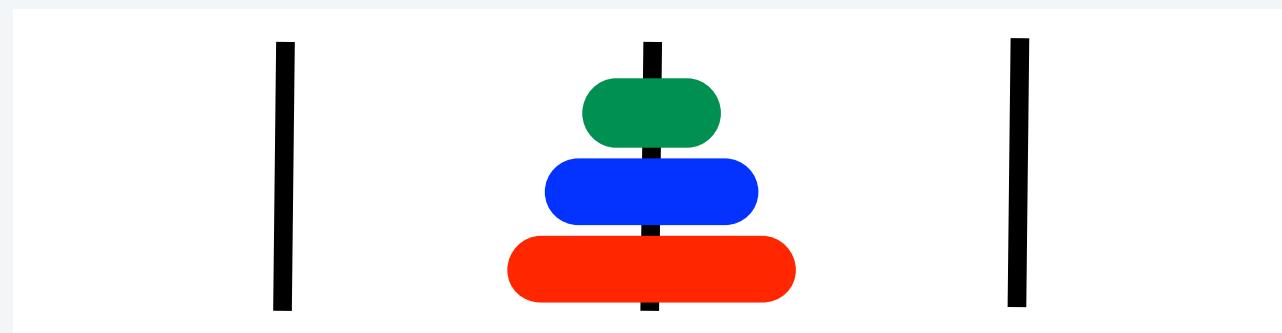


A recursive solution

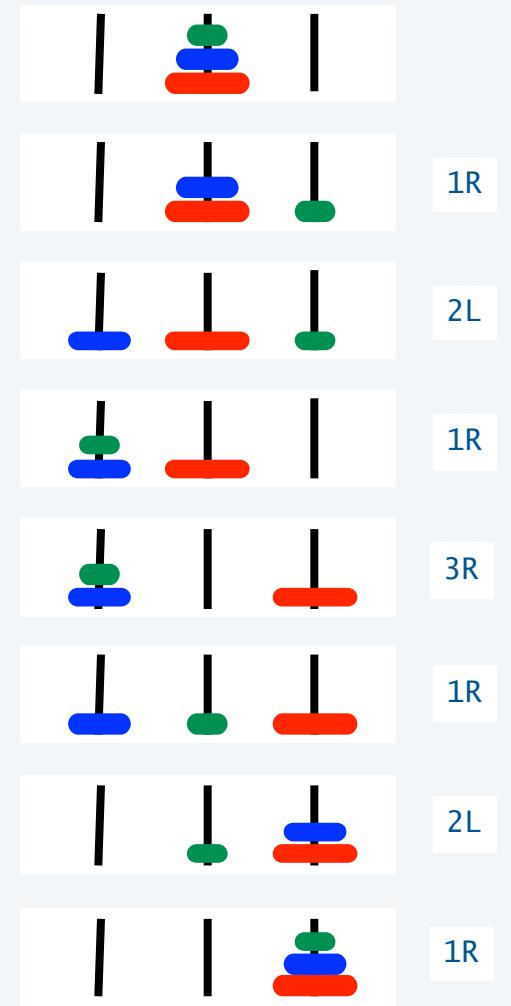
- Move $n - 1$ discs to the left (recursively).
- Move largest disc to the *right*.
- Move $n - 1$ discs to the left (recursively).



Towers of Hanoi solution ($n = 3$)



1R 2L 1R 3R 1R 2L 1R



Towers of Hanoi: recursive solution

hanoi(n): Print moves for n discs.

- Return one space for $n = 0$.
- Otherwise, set `move` to the specified move for disc n .
- Then sandwich `move` between two copies of `hanoi($n-1$)`.

```
public class Hanoi
{
    public static String hanoi(int n, boolean left)
    {
        if (n == 0) return " ";
        String move;
        if (left) move = n + "L";
        else      move = n + "R";
        return hanoi(n-1, !left) + move + hanoi(n-1, !left);
    }

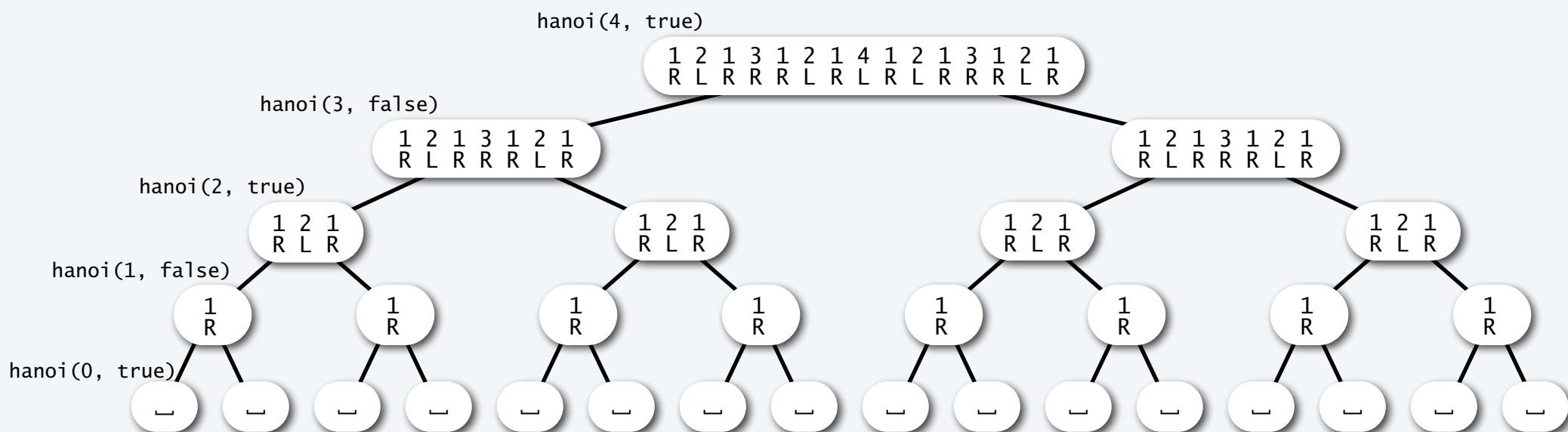
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        StdOut.println(hanoi(n, false));
    }
}
```

```
% java Hanoi 3
1R 2L 1R 3R 1R 2L 1R
```

Recursive call tree for towers of Hanoi

Structure is the *same* as for the ruler function and suggests 3 useful and easy-to-prove facts.

- Each disc always moves in the same direction.
- Moving smaller disc always alternates with a unique legal move.
- Moving n discs requires $2^n - 1$ moves.



Answers for towers of Hanoi

Q. Generate list of instructions for monks ?

A. (Long form). 1L 2R 1L 3L 1L 2R 1L 4R 1L 2R 1L 3L 1L 2R 1L 5L 1L 2R 1L 3L 1L 2R 1L 4R ...

A. (Short form). Alternate "1L" with the only legal move not involving the disc 1.
"L" or "R" depends on whether n is odd or even

Q. When might the world end ?

A. Not soon: need $2^{64} - 1$ moves.

moves per second	end of world
1	5.84 billion centuries
1 billion	5.84 centuries

Note: Recursive solution has been proven optimal.





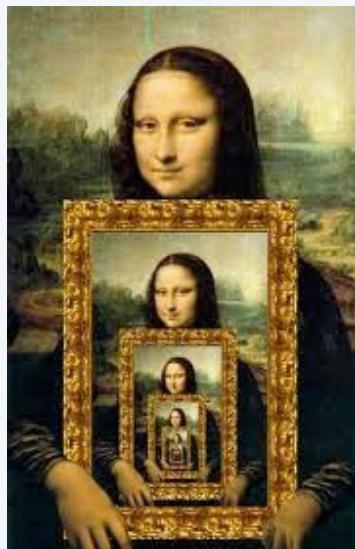
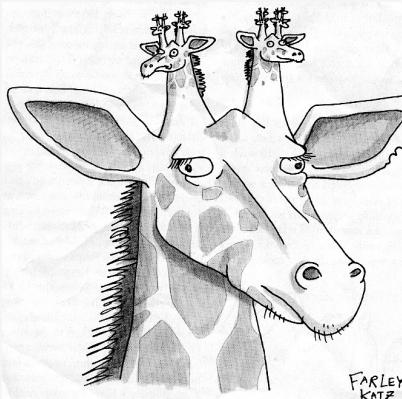
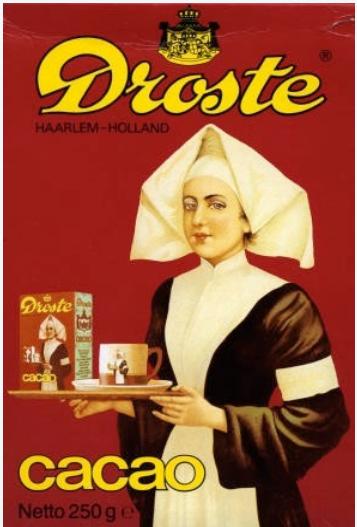
CS.6.B.Recursion.Hanoi



6. Recursion

- Foundations
- A classic example
- **Recursive graphics**
- Avoiding exponential waste
- Dynamic programming

Recursive graphics in the wild



Black, White and Read All Over Over

Divine and Devotee Meet Across Hinge

WASHINGTON — For mother-teacher, Sallie Washington (ASPA), She'll be riding as a role model. Keep it moving, though, in shape. Everyone knows that a prayer to St. Bernadette is good for the soul, but it's also good for the body. **ILLINOIS** — Cotter (ASPA) is here to stay. She's never seen so well. She's been a part of the team, for die-hard and strange problems. **MISSOURI** — The new coach of the girls' soccer team — she's the Virgo. Despite her lack of experience, she's going after victory, and probably adding another trophy to the school's collection. **MISSOURI** — The new coach of the girls' soccer team — she's the Virgo. Despite her lack of experience, she's going after victory, and probably adding another trophy to the school's collection. **MISSOURI** — The new coach of the girls' soccer team — she's the Virgo. Despite her lack of experience, she's going after victory, and probably adding another trophy to the school's collection. **MISSOURI** — The new coach of the girls' soccer team — she's the Virgo. Despite her lack of experience, she's going after victory, and probably adding another trophy to the school's collection.

Two small black and white photographs showing close-up views of hair or skin texture.

Prayers and Portraits:
Unveiling the
Netherlandish
Devotional Tradition

gradually as you
know how diptych

Sitow, left, are reprinted in an exhibition at the

National Gallery of Art in Washington through Feb. 4.

A close-up detail from a painting showing a person's torso and hands. The person is wearing a patterned garment and a dark belt. Their hands are clasped in front of them. The background is dark and textured.

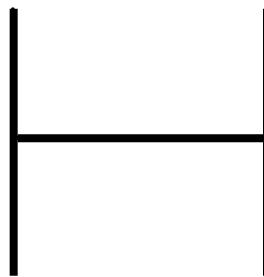
Left, Ghibelline, Berlin; National Gallery of Art, Washington

"Hello, World" of recursive graphics: H-trees

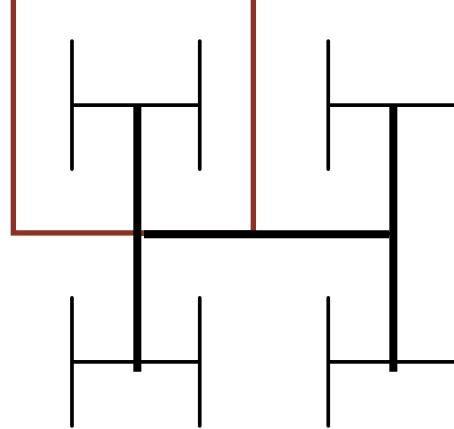
H-tree of order n

- If n is 0, do nothing.
- Draw an H, centered.
- Draw four H-trees of order $n - 1$ and half the size, centered at the tips of the H.

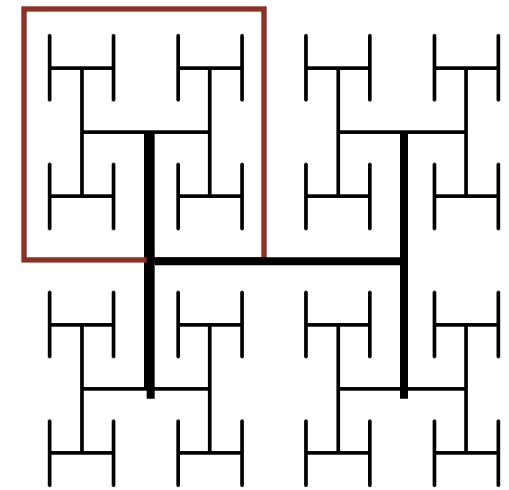
order 1



order 2

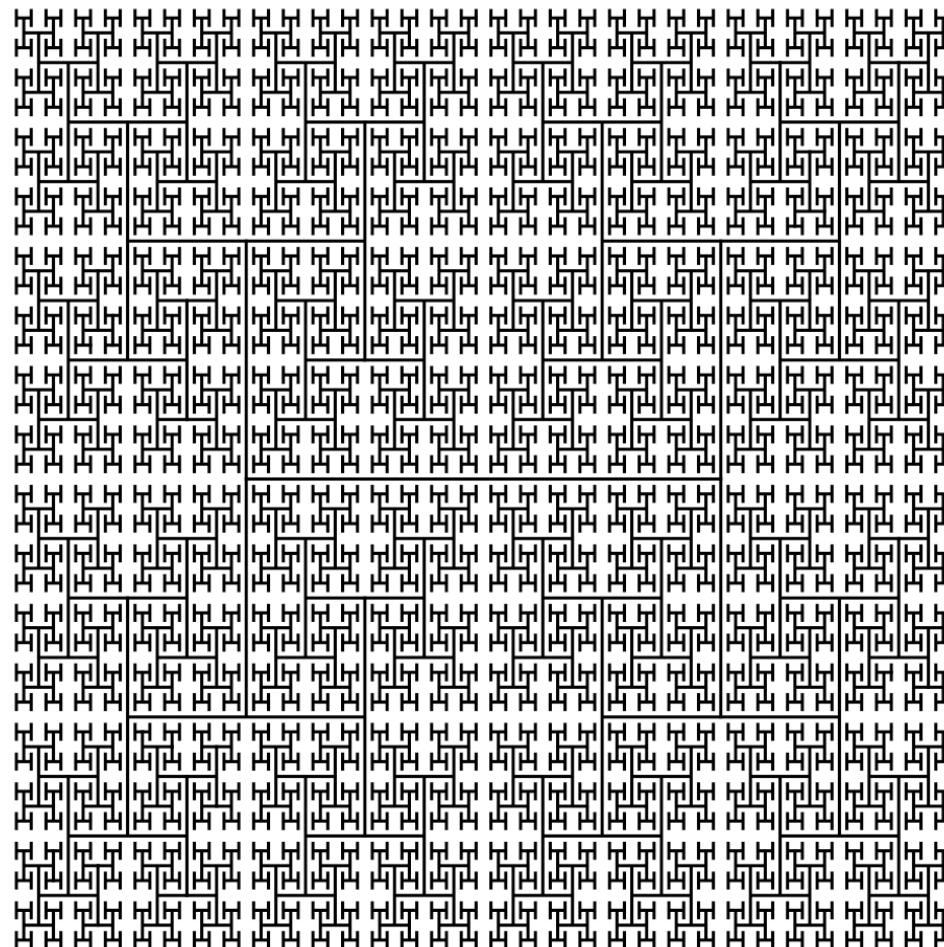


order 3



H-trees

Application. Connect
a large set of
regularly spaced sites
to a single source.

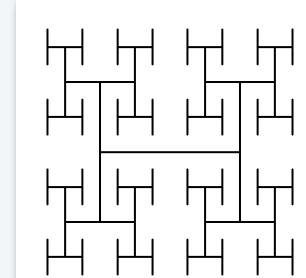
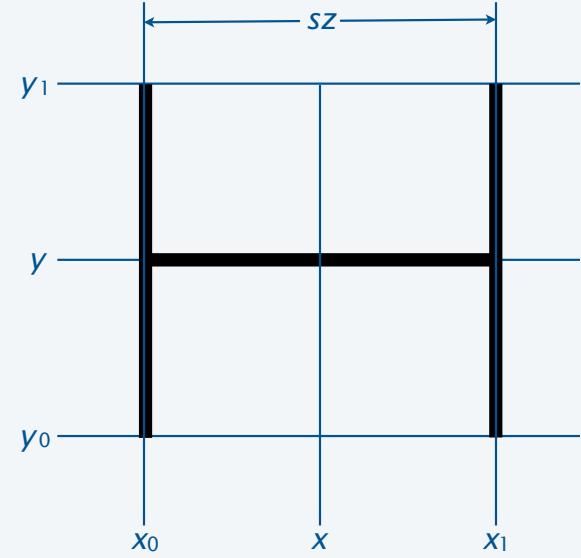


order 6

Recursive H-tree implementation

```
public class Htree
{
    public static void draw(int n, double sz, double x, double y)
    {
        if (n == 0) return;
        double x0 = x - sz/2, x1 = x + sz/2;
        double y0 = y - sz/2, y1 = y + sz/2;
        StdDraw.line(x0, y, x1, y);
        StdDraw.line(x0, y0, x0, y1);           | draw the H,
        StdDraw.line(x1, y0, x1, y1);           | centered on (x, y)
        draw(n-1, sz/2, x0, y0);
        draw(n-1, sz/2, x0, y1);           | draw four
        draw(n-1, sz/2, x1, y0);           | half-size H-trees
        draw(n-1, sz/2, x1, y1);
    }
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        draw(n, .5, .5, .5);
    }
}
```

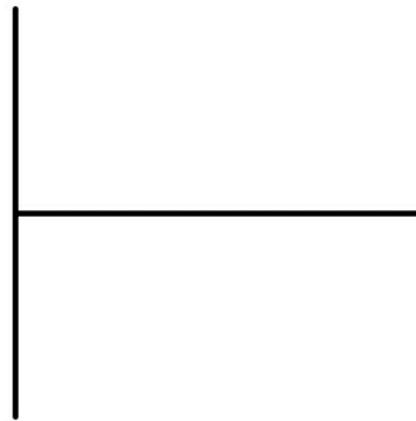
% java Htree 3



Deluxe H-tree implementation

```
public class HtreeDeluxe
{
    public static void draw(int n, double sz,
                           double x, double y)
    {
        if (n == 0) return;
        double x0 = x - sz/2, x1 = x + sz/2;
        double y0 = y - sz/2, y1 = y + sz/2;
        StdDraw.line(x0, y, x1, y);
        StdDraw.line(x0, y0, x0, y1);
        StdDraw.line(x1, y0, x1, y1);
        StdAudio.play(PlayThatNote.note(n, .25*n));
        draw(n-1, sz/2, x0, y0);
        draw(n-1, sz/2, x0, y1);
        draw(n-1, sz/2, x1, y0);
        draw(n-1, sz/2, x1, y1);
    }
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        draw(n, .5, .5, .5);
    }
}
```

```
% java HtreeDeluxe 4
```



Fractional Brownian motion

A process that models many phenomenon.

- Price of stocks.
- Dispersion of fluids.
- Rugged shapes of mountains and clouds.
- Shape of nerve membranes.

...

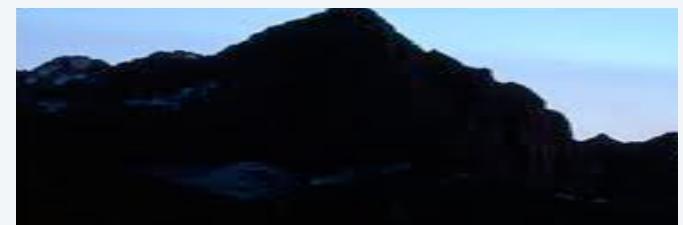
Price of an actual stock



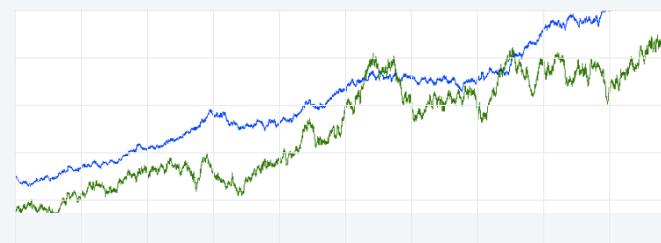
Brownian bridge model



An actual mountain



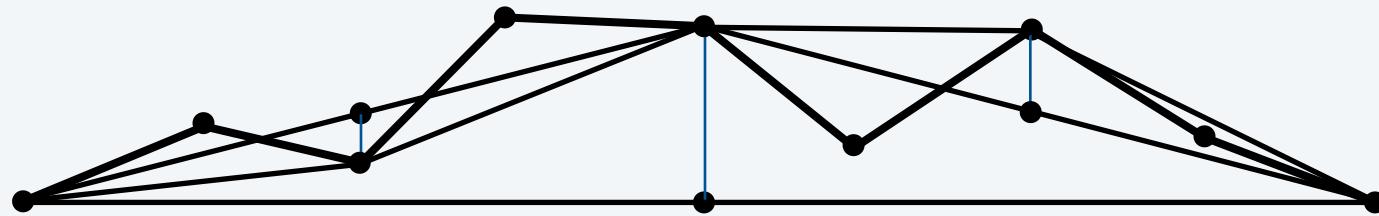
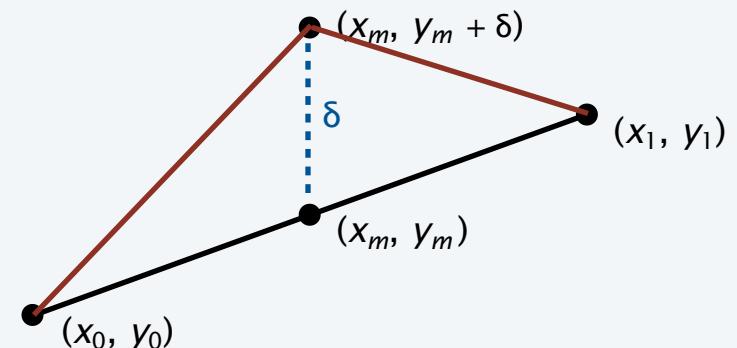
Black-Scholes model (two different parameters)



Fractional Brownian motion simulation

Midpoint displacement method

- Consider a line segment from (x_0, y_0) to (x_1, y_1) .
- If sufficiently short draw it *and return*
- Divide the line segment in half, at (x_m, y_m) .
- Choose δ at random *from Gaussian distribution*.
- Add δ to y_m .
- Recur on the left and right line segments.



Brownian motion implementation

```
public class Brownian
{
    public static void
    curve(double x0, double y0, double x1, double y1,
          double var, double s)
    {
        if (x1 - x0 < .01)
        { StdDraw.line(x0, y0, x1, y1); return; }
        double xm = (x0 + x1) / 2;
        double ym = (y0 + y1) / 2;
        double stddev = Math.sqrt(var);
        double delta = StdRandom.gaussian(0, stddev);
        curve(x0, y0, xm, ym+delta, var/s, s);
        curve(xm, ym+delta, x1, y1, var/s, s);
    }

    public static void main(String[] args)
    {
        double hurst = Double.parseDouble(args[0]);
        double s = Math.pow(2, 2*hurst);
        curve(0, .5, 1.0, .5, .01, s); ← control parameter
    }                                         (see text)
}
```

% java Brownian 1



% java Brownian .125

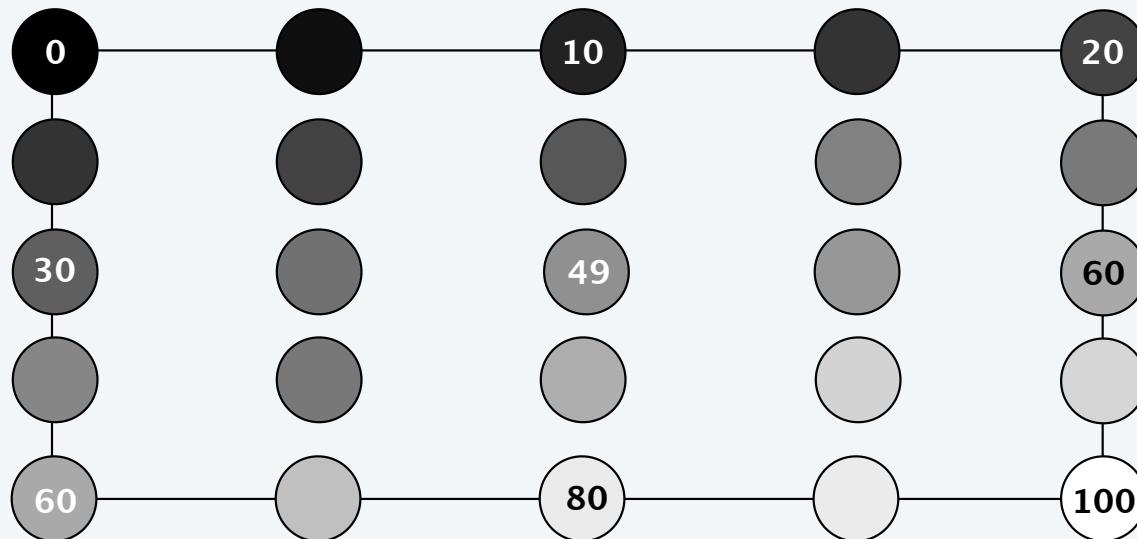


A 2D Brownian model: plasma clouds

Midpoint displacement method

- Consider a rectangle centered at (x, y) with pixels at the four corners.
- If the rectangle is small, do nothing.
- Color the midpoints of each side the average of the endpoint colors.
- Choose δ at random *from Gaussian distribution*.
- Color the center pixel the average of the four corner colors *plus* δ
- Recurse on the four quadrants.

Books site code actually
draws a rectangle to
avoid artifacts



A Brownian cloud

A Brownian landscape



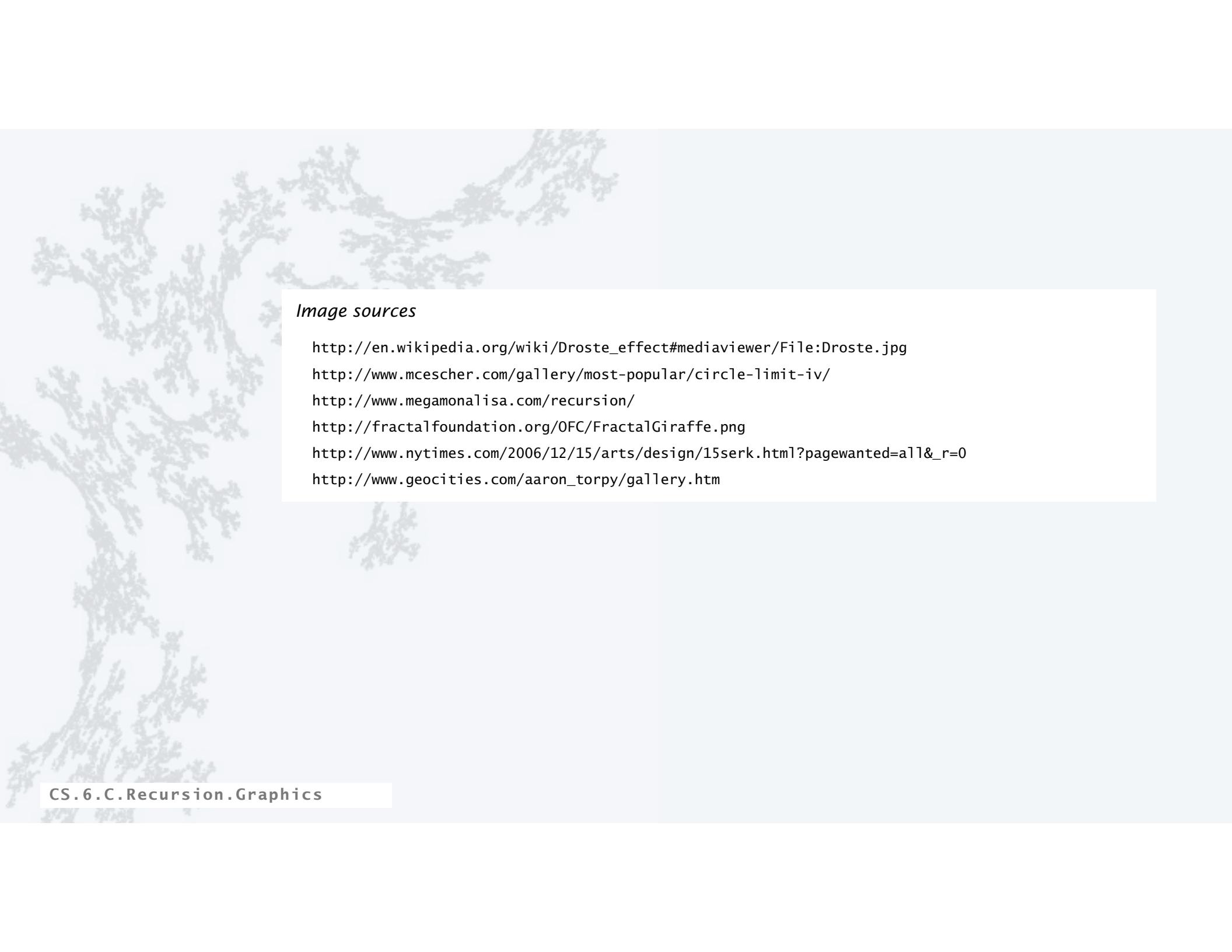


Image sources

http://en.wikipedia.org/wiki/Droste_effect#mediaviewer/File:Droste.jpg
<http://www.mcescher.com/gallery/most-popular/circle-limit-iv/>
<http://www.megamonalisa.com/recursion/>
<http://fractalfoundation.org/OFC/FractalGiraffe.png>
http://www.nytimes.com/2006/12/15/arts/design/15serk.html?pagewanted=all&_r=0
http://www.geocities.com/aaron_torpy/gallery.htm



6. Recursion

- Foundations
- A classic example
- Recursive graphics
- **Avoiding exponential waste**
- Dynamic programming

Fibonacci numbers

Let $F_n = F_{n-1} + F_{n-2}$ for $n > 1$ with $F_0 = 0$ and $F_1 = 1$.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
F_n	0	1	1	2	3	5	8	13	21	34	55	89	144	233	...



Leonardo Fibonacci
c. 1170 – c. 1250

Models many natural phenomena and is widely found in art and architecture.

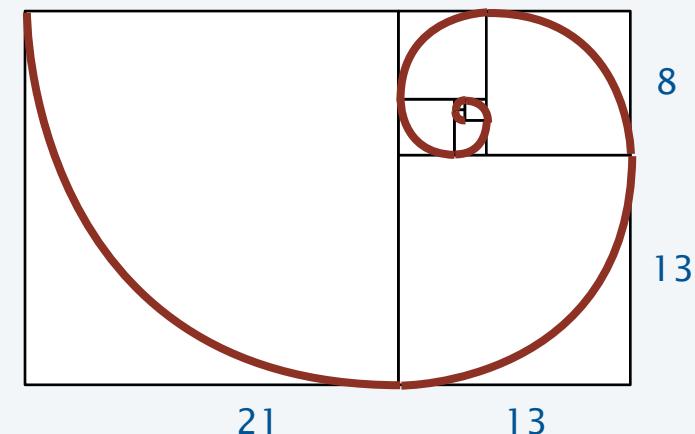
Examples.

- Model for reproducing rabbits.
- Nautilus shell.
- Mona Lisa.
- ...

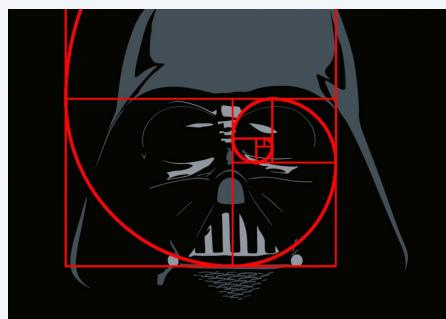
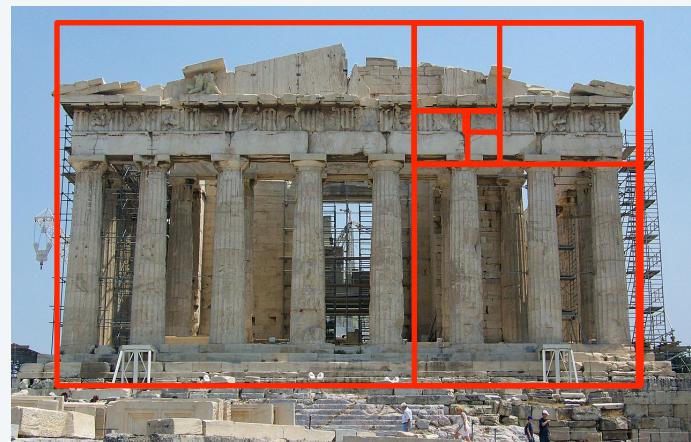
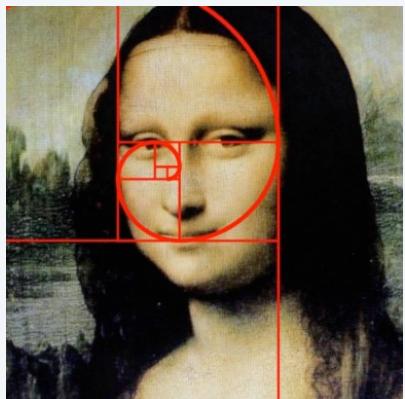
Facts (known for centuries).

- $F_n / F_{n-1} \rightarrow \Phi = 1.618\dots$ as $n \rightarrow \infty$
- F_n is the closest integer to $\Phi^n / \sqrt{5}$

golden ratio F_n / F_{n-1}



Fibonacci numbers and the golden ratio in the wild



1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15	6	1	
1	7	21	35	35	21	7	1

Computing Fibonacci numbers

Q. [Curious individual.] What is the exact value of F_{60} ?

A. [Novice programmer.] Just a second. I'll write a recursive program to compute it.

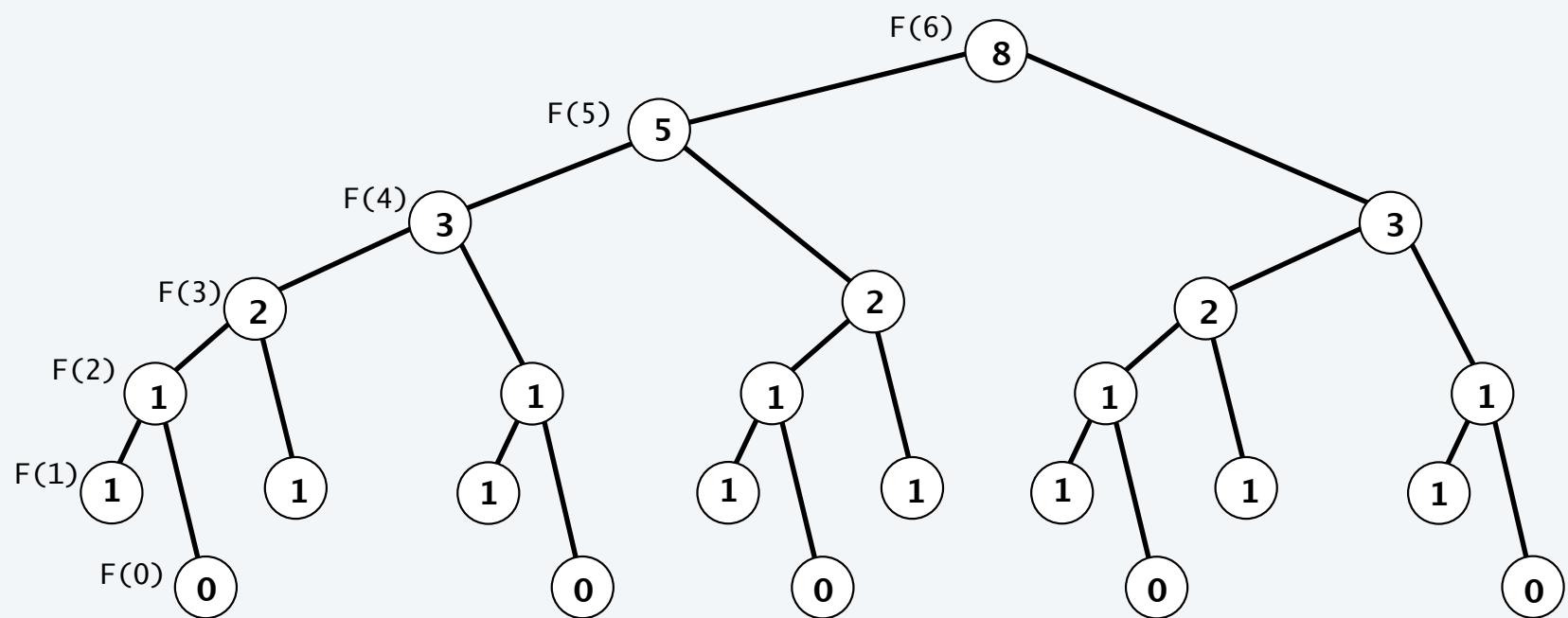
```
public class FibonacciR
{
    public static long F(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return F(n-1) + F(n-2);
    }
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        StdOut.println(F(n));
    }
}
```

```
% java FibonacciR 5
5
% java FibonacciR 6
8
% java FibonacciR 10
55
% java FibonacciR 12
144
% java FibonacciR 50
12586269025
% java FibonacciR 60
```

takes a few minutes
Hmmm. Why is that?

Is something wrong with my computer?

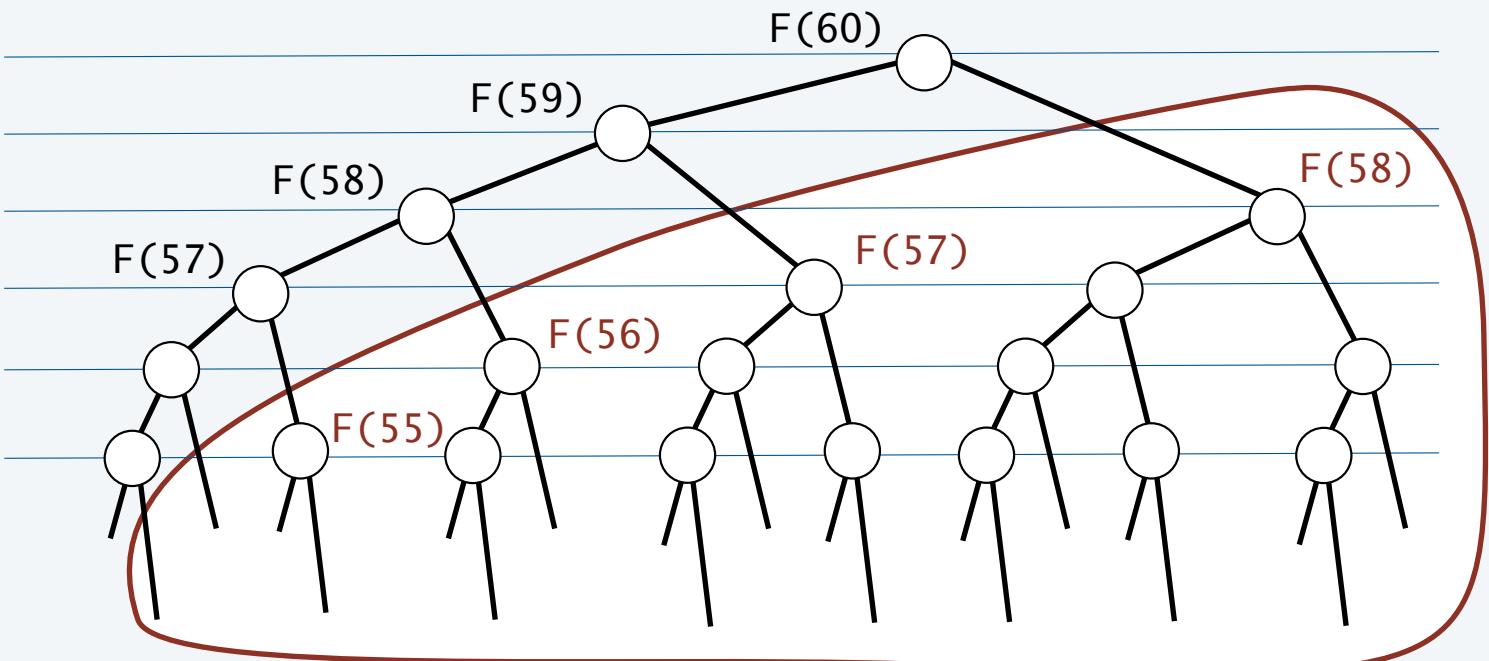
Recursive call tree for Fibonacci numbers



Exponential waste

Let C_n be the number of times $F(n)$ is called when computing $F(60)$.

n	C_n	
60	1	F_1
59	1	F_2
58	2	F_3
57	3	F_4
56	5	F_5
55	8	F_6
...	...	
0	$>2.5 \times 10^{12}$	F_{61}



*Exponentially wasteful to recompute all these values.
(trillions of calls on $F(0)$, not to mention calls on $F(1), F(2), \dots$)*

Exponential waste dwarfs progress in technology

If you engage in exponential waste, you *will not* be able to solve a large problem.

1970s



n	<i>time to compute F_n</i>
30	minutes
40	hours
50	weeks
60	years
70	centuries
80	millenia

2010s: 10,000+ times faster

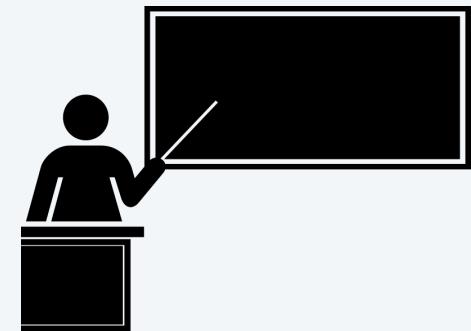


Macbook Air

n	<i>time to compute F_n</i>
50	minutes
60	hours
70	weeks
80	years
90	centuries
100	millenia

1970s: "That program won't compute F_{60} before you graduate!"

2010s: "That program won't compute F_{80} before you graduate!"



Avoiding exponential waste

Memoization

- Maintain an array `memo[]` to remember all computed values.
- If value known, just return it.
- Otherwise, compute it, remember it, and then return it.

```
public class FibonacciM
{
    static long[] memo = new long[100];
    public static long F(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        if (memo[n] == 0)
            memo[n] = F(n-1) + F(n-2);
        return memo[n];
    }
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        StdOut.println(F(n));
    }
}
```

```
% java FibonacciM 50
12586269025
% java FibonacciM 60
1548008755920
% java FibonacciM 80
23416728348467685
```

Simple example of *dynamic programming* (next).

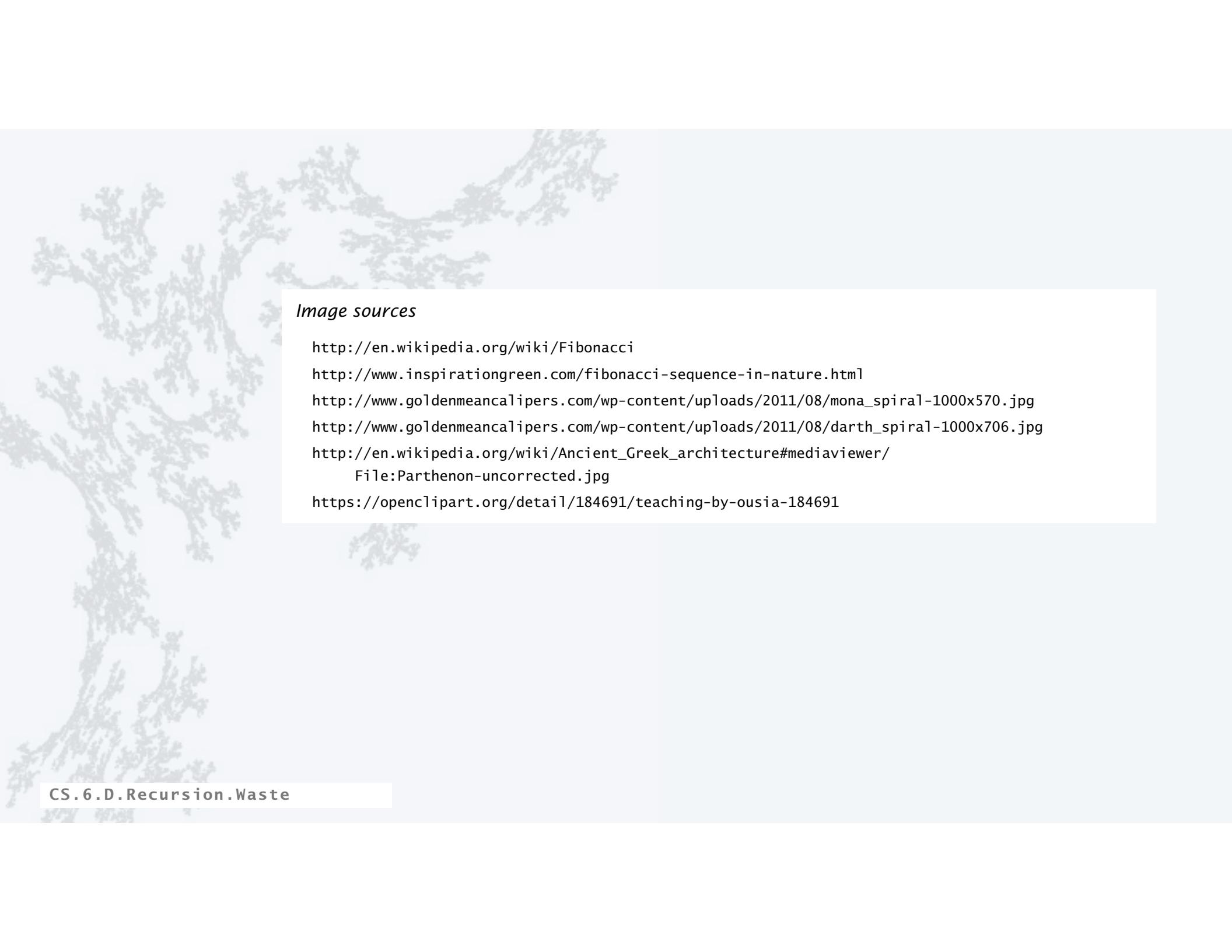


Image sources

<http://en.wikipedia.org/wiki/Fibonacci>
<http://www.inspirationgreen.com/fibonacci-sequence-in-nature.html>
http://www.goldenmeancalipers.com/wp-content/uploads/2011/08/mona_spiral-1000x570.jpg
http://www.goldenmeancalipers.com/wp-content/uploads/2011/08/darth_spiral-1000x706.jpg
[http://en.wikipedia.org/wiki/Ancient_Greek_architecture#mediaviewer/
File:Parthenon-uncorrected.jpg](http://en.wikipedia.org/wiki/Ancient_Greek_architecture#mediaviewer/File:Parthenon-uncorrected.jpg)
<https://openclipart.org/detail/184691/teaching-by-ousia-184691>



7. Recursion

- Foundations
- A classic example
- Recursive graphics
- Avoiding exponential waste
- **Dynamic programming**

An alternative to recursion that avoids recomputation

Dynamic programming.

- Build computation from the "*bottom up*".
- Solve small subproblems *and save solutions*.
- Use those solutions to build bigger solutions.

Fibonacci numbers

```
public class Fibonacci
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        long[] F = new long[n+1];
        F[0] = 0; F[1] = 1;
        for (int i = 2; i <= n; i++)
            F[i] = F[i-1] + F[i-2];
        StdOut.println(F[n]);
    }
}
```



Richard Bellman
1920-1984

```
% java Fibonacci 50
12586269025
% java Fibonacci 60
1548008755920
% java Fibonacci 80
23416728348467685
```

Key advantage over recursive solution. Each subproblem is addressed only *once*.

DP example: Longest common subsequence

Def. A *subsequence* of a string s is any string formed by deleting characters from s .

Ex 1. $s = \text{ggcaccacg}$

cac	ggcaccacg
gcaacg	ggcaccacg
ggcaacg	ggcaccacg
ggcacacg	ggcaccacg
...	

[2^n subsequences in a string of length n]

Ex 2. $t = \text{acggcggtacg}$

gacg	acggcggtacg
gggg	acggcggtacg
cggcgg	acggcggtacg
ggcaacg	acggcggtacg
gggaacg	acggcggtacg
...	

longest common subsequence

Def. The *LCS* of s and t is the longest string that is a subsequence of both.

Goal. Efficient algorithm to compute the LCS and/or its length

← numerous scientific applications

Longest common subsequence

Goal. Efficient algorithm to compute the *length* of the LCS of two strings s and t .

Approach. Keep track of the length of the LCS of $s[i..M)$ and $t[j..N)$ in $\text{opt}[i, j]$

Ex: $i = 6, j = 7$

$s[6..9) = \text{acg}$
 $t[7..12) = \text{atacg}$
 $\text{LCS}(\text{cg}, \text{tacg}) = \text{cg}$
 $\text{LCS}(\text{acg}, \text{atacg}) = \text{acg}$

Three cases:

- $i = M$ or $j = N$
 $\text{opt}[i][j] = 0$
- $s[i] = t[j]$
 $\text{opt}[i][j] = \text{opt}[i+1, j+1] + 1$
- otherwise
 $\text{opt}[i][j] = \max(\text{opt}[i, j+1], \text{opt}[i+1][j])$

Ex: $i = 6, j = 4$

$s[6..9) = \text{acg}$
 $t[4..12) = \text{cggatacg}$
 $\text{LCS}(\text{acg}, \text{ggatacg}) = \text{acg}$
 $\text{LCS}(\text{cg}, \text{cggatacg}) = \text{cg}$
 $\text{LCS}(\text{acg}, \text{cggatacg}) = \text{acg}$

LCS example

	0	1	2	3	4	5	6	7	8	9	10	11	12		
	a	c	g	g	c	g	g	a	t	a	c	g			
0	g		7	7	7	6	6	6	5	4	3	3	2	1	0
1	g		6	6	6	6	5	5	5	4	3	3	2	1	0
2	c		6	5	5	5	5	4	4	4	3	3	2	1	0
3	a		6	5	4	4	4	4	4	4	3	3	2	1	0
4	c		5	5	4	4	4	3	3	3	3	3	2	1	0
5	c		4	4	4	4	4	3	3	3	3	3	2	1	0
6	a		3	3	3	3	3	3	3	3	3	2	1	0	
7	c		2	2	2	2	2	2	2	2	2	2	1	0	
8	g		1	1	1	1	1	1	1	1	1	1	1	1	0
9			0	0	0	0	0	0	0	0	0	0	0	0	0

$\text{opt}[i][j] = \max(\text{opt}[i, j+1], \text{opt}[i+1][j])$

$\text{opt}[i][j] = \text{opt}[i+1, j+1] + 1$

LCS length implementation

```
public class LCS
{
    public static void main(String[] args)
    {
        String s = args[0];
        String t = args[1];
        int M = s.length();
        int N = t.length();

        int[][] opt = new int[M+1][N+1];
        for (int i = M-1; i >= 0; i--)
            for (int j = N-1; j >= 0; j--)
                if (s.charAt(i) == t.charAt(j))
                    opt[i][j] = opt[i+1][j+1] + 1;
                else
                    opt[i][j] = Math.max(opt[i+1][j], opt[i][j+1]);
        System.out.println(opt[0][0]);
    }
}
```

```
% java LCS ggcaccacg acggcggatacg
7
```

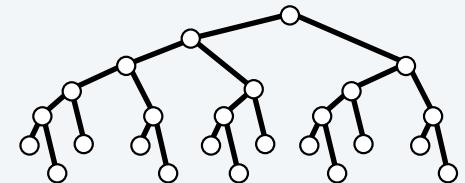
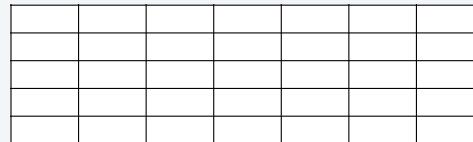
Exercise. Add code to print LCS itself (see `LCS.java` on booksite for solution).

Dynamic programming and recursion

Broadly useful approaches to solving problems by combining solutions to smaller subproblems.

Why learn DP and recursion?

- Represent a new mode of thinking.
- Provide powerful programming paradigms.
- Give insight into the nature of computation.
- Successfully used for decades.



	<i>recursion</i>	<i>dynamic programming</i>
<i>advantages</i>	Decomposition often obvious. Easy to reason about correctness.	Avoids exponential waste. Often simpler than memoization.
<i>pitfalls</i>	Potential for exponential waste. Decomposition may not be simple.	Uses significant space. Not suited for real-valued arguments. Challenging to determine order of computation

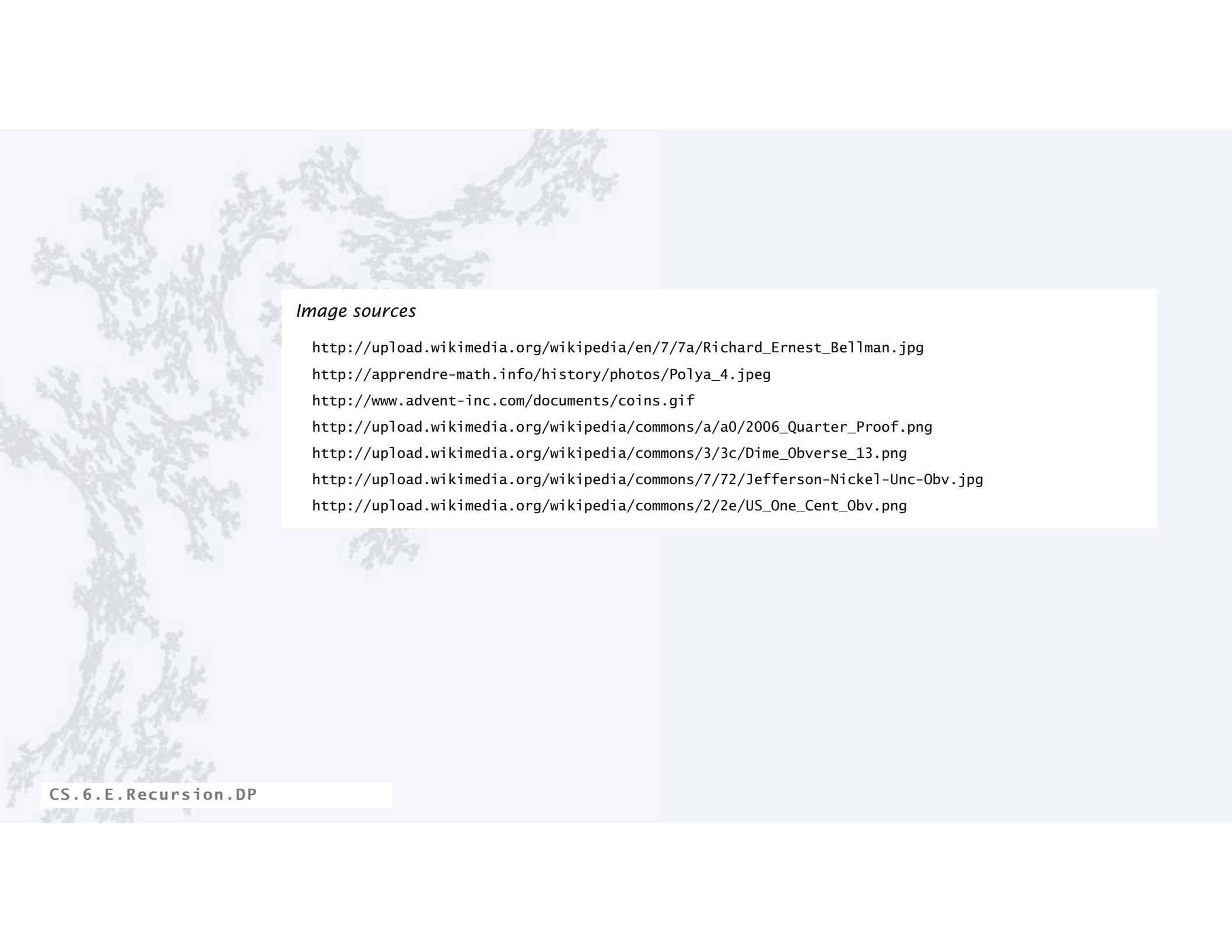
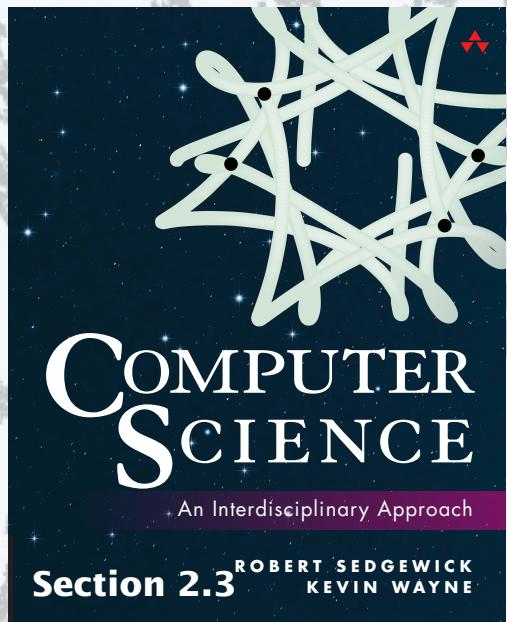


Image sources

http://upload.wikimedia.org/wikipedia/en/7/7a/Richard_Ernest_Bellman.jpg
http://apprendre-math.info/history/photos/Polya_4.jpeg
<http://www.advent-inc.com/documents/coins.gif>
http://upload.wikimedia.org/wikipedia/commons/a/a0/2006_Quarter_Proof.png
http://upload.wikimedia.org/wikipedia/commons/3/3c/Dime_Obverse_13.png
<http://upload.wikimedia.org/wikipedia/commons/7/72/Jefferson-Nickel-Unc-Obv.jpg>
http://upload.wikimedia.org/wikipedia/commons/2/2e/US_One_Cent_Obv.png



6. Recursion