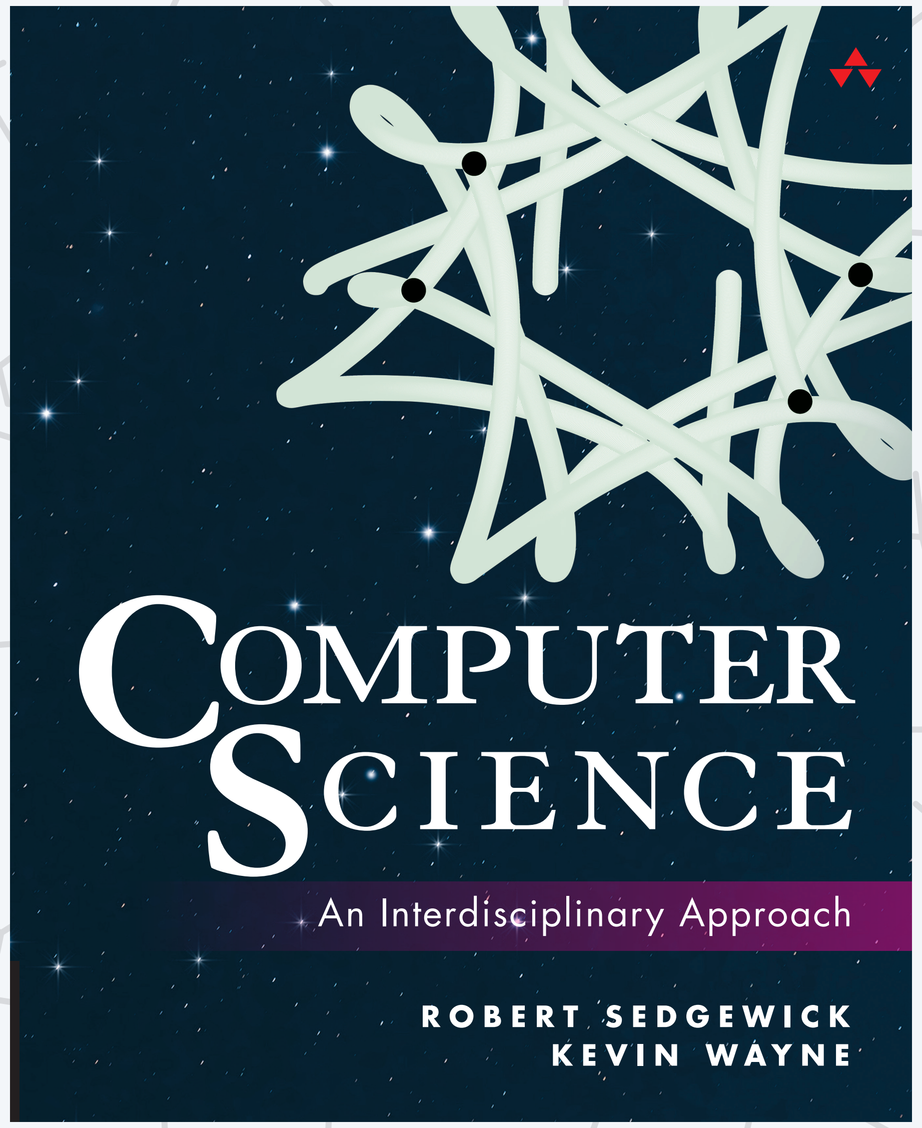
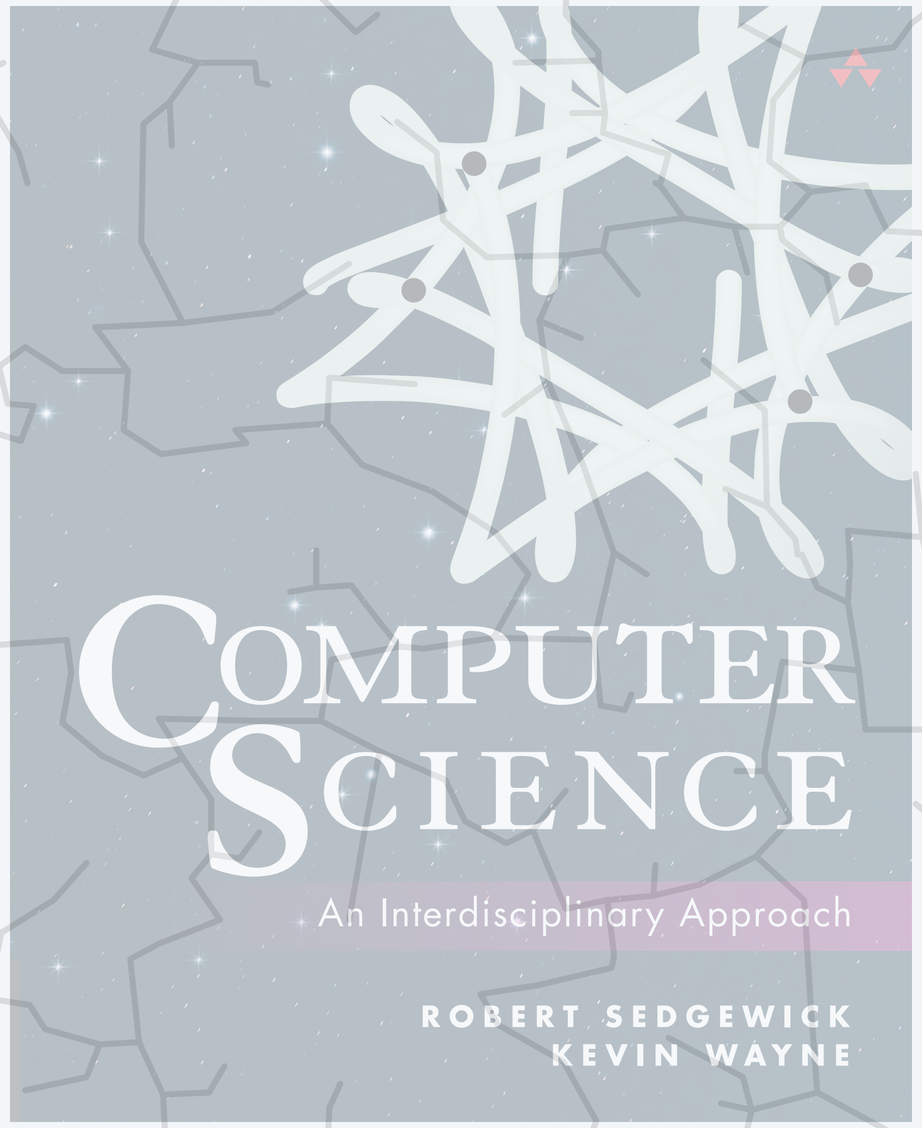


https://introcs.cs.princeton.edu

2.3 Recursion

- *foundations*
- *a classic example*
- *recursive graphics*
- *exponential waste*

DRAFT



https://introcs.cs.princeton.edu

2.3 RECURSION

- *foundations*
- *a classic example*
- *recursive graphics*
- *exponential waste*

Overview

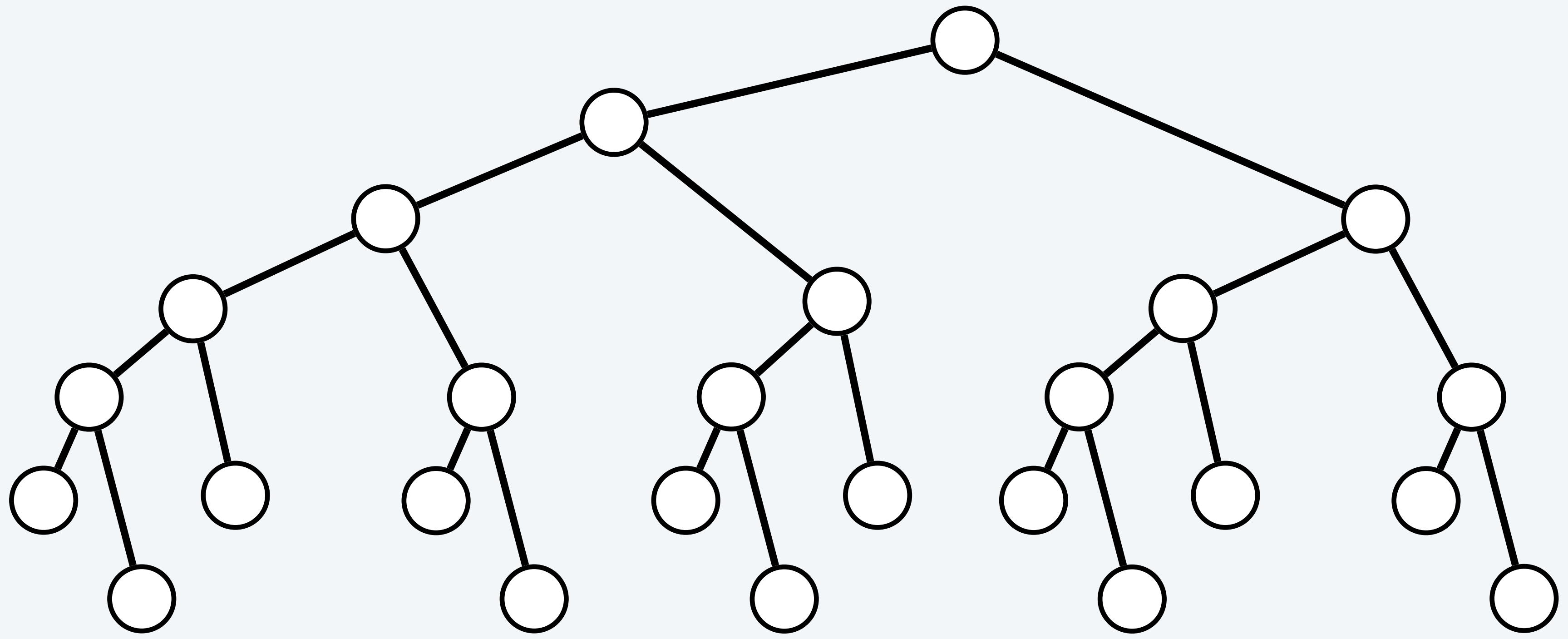
Recursion is when something is specified in terms of itself.

Why learn recursion?

- Represents a new mode of thinking.
- Provides a powerful programming paradigm.
- Reveals insight into the nature of computation.

Many computational artifacts are naturally self-referential.

- File system with folders containing folders.
- Binary trees.
- Fractal patterns.
- Depth-first search.
- Divide-and-conquer algorithms.
- ...



Recursive functions (in Java)

Recursive function. A function that calls itself.

- Base case: If the result can be computed directly, do so.
- Reduction step: Otherwise, simplify by calling the function with one (or more) other arguments.

Ex. Factorial function: $n! = n \times (n-1) \times \cdots \times 3 \times 2 \times 1$.

- Base case: $1! = 1$
- Reduction step: $n! = n \times (n-1)!$

same function with simpler argument

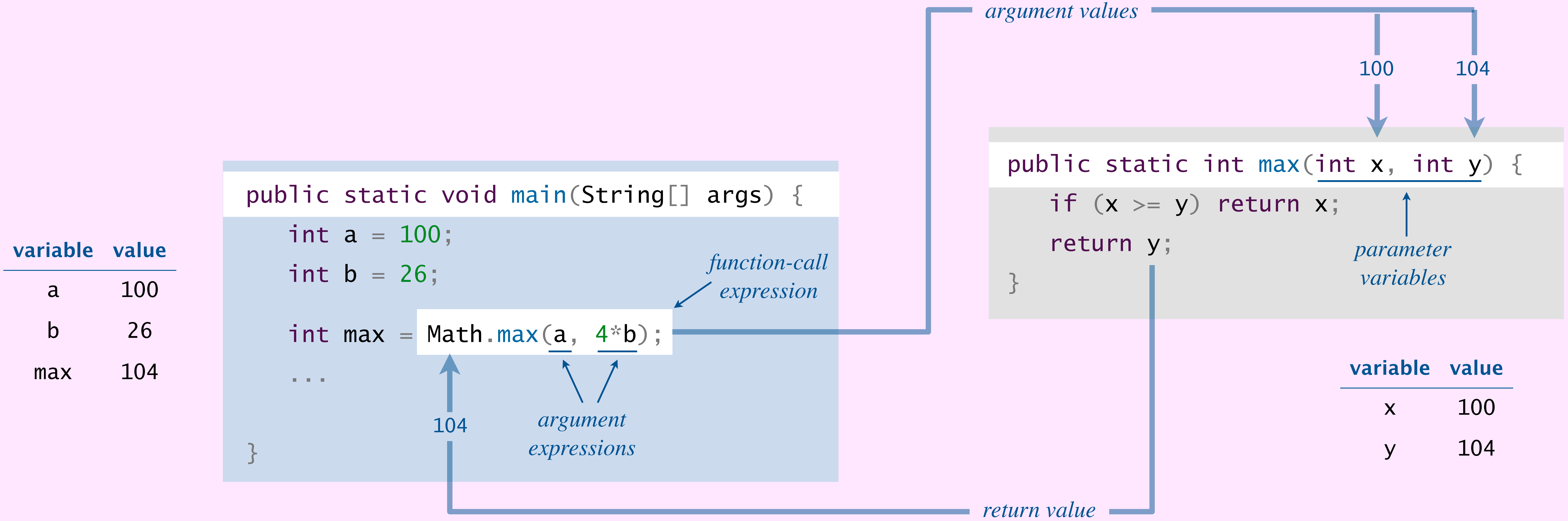
```
~/cos126/recursion> java-introcs Factorial 3
6

~/cos126/recursion> java-introcs Factorial 4
24

~/cos126/recursion> java-introcs Factorial 5
120
```

recursive function

```java
public class Factorial {
   public static int factorial(int n) {
      if (n == 1) return 1;           // base case
      return n * factorial(n-1);      // reduction step
   }

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

Review: mechanics of a function call

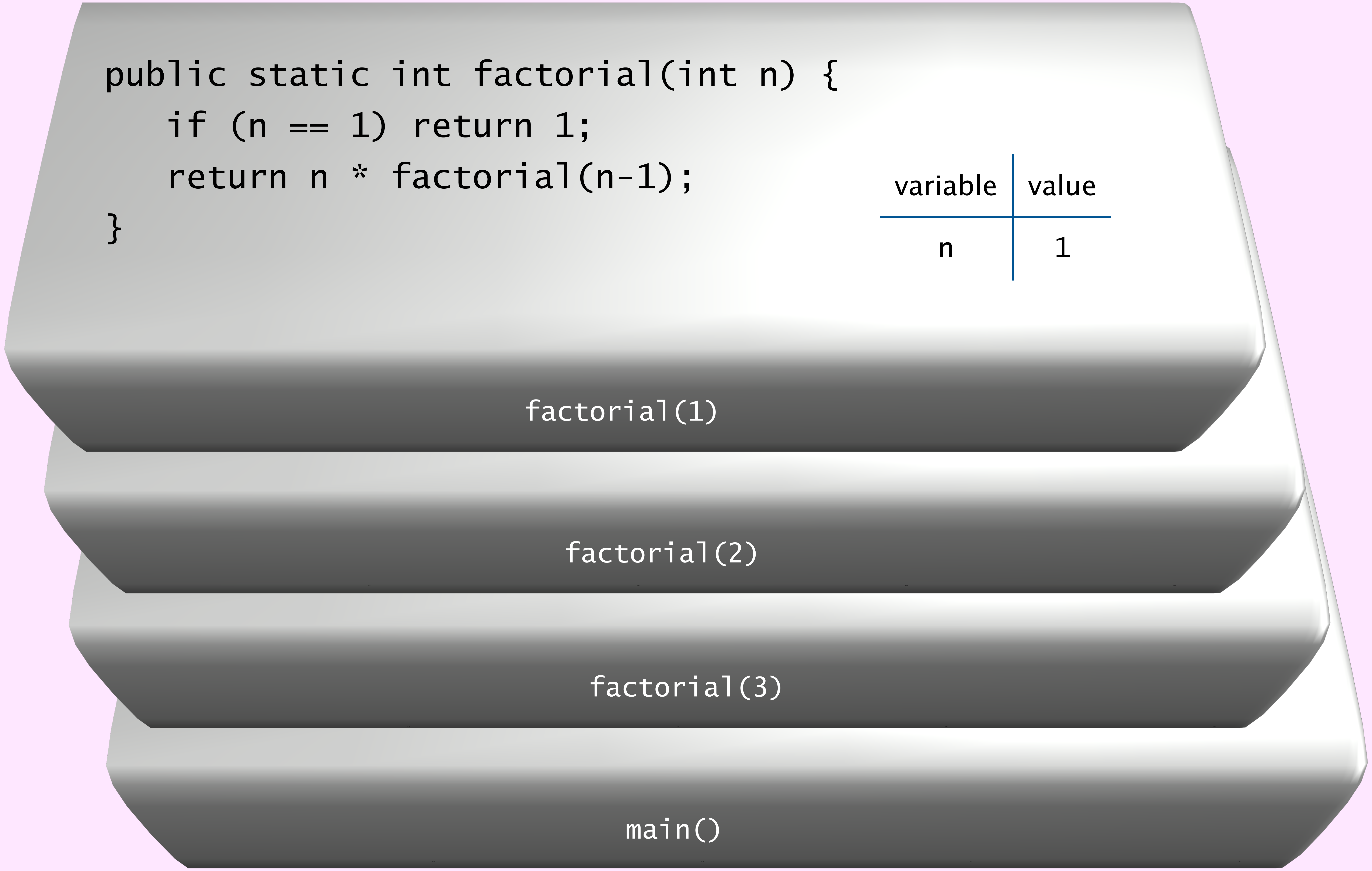
1. *Evaluate* argument expressions and *assign* values to corresponding parameter variables.
2. *Save environment* (values of all local variables and call location).
3. *Transfer control* to the function.
4. *Restore environment* (with function-call expression evaluating to return value).
5. *Transfer control* back to the calling code.



Factorial function demo

```
public static int factorial(int n) {
   if (n == 1) return 1;
   return n * factorial(n-1);
}
```

variable	value
n	1



Function-call trace

Function-call trace.

- Print name and arguments when each function is called.
- Print function's return value just before returning.
- Add indentation on function calls and subtract on returns.

```
factorial(5)
    factorial(4)
        factorial(3)
            factorial(2)
                factorial(1)
                    return 1
                return 2 * 1 = 2
            return 3 * 2 = 6
        return 4 * 6 = 24
    return 5 * 24 = 120
```

function-call trace for `factorial(5)`

```
public static int factorial(int n) {
   if (n == 1) return 1;
   return n * factorial(n-1);
}
```

Stack overflow errors

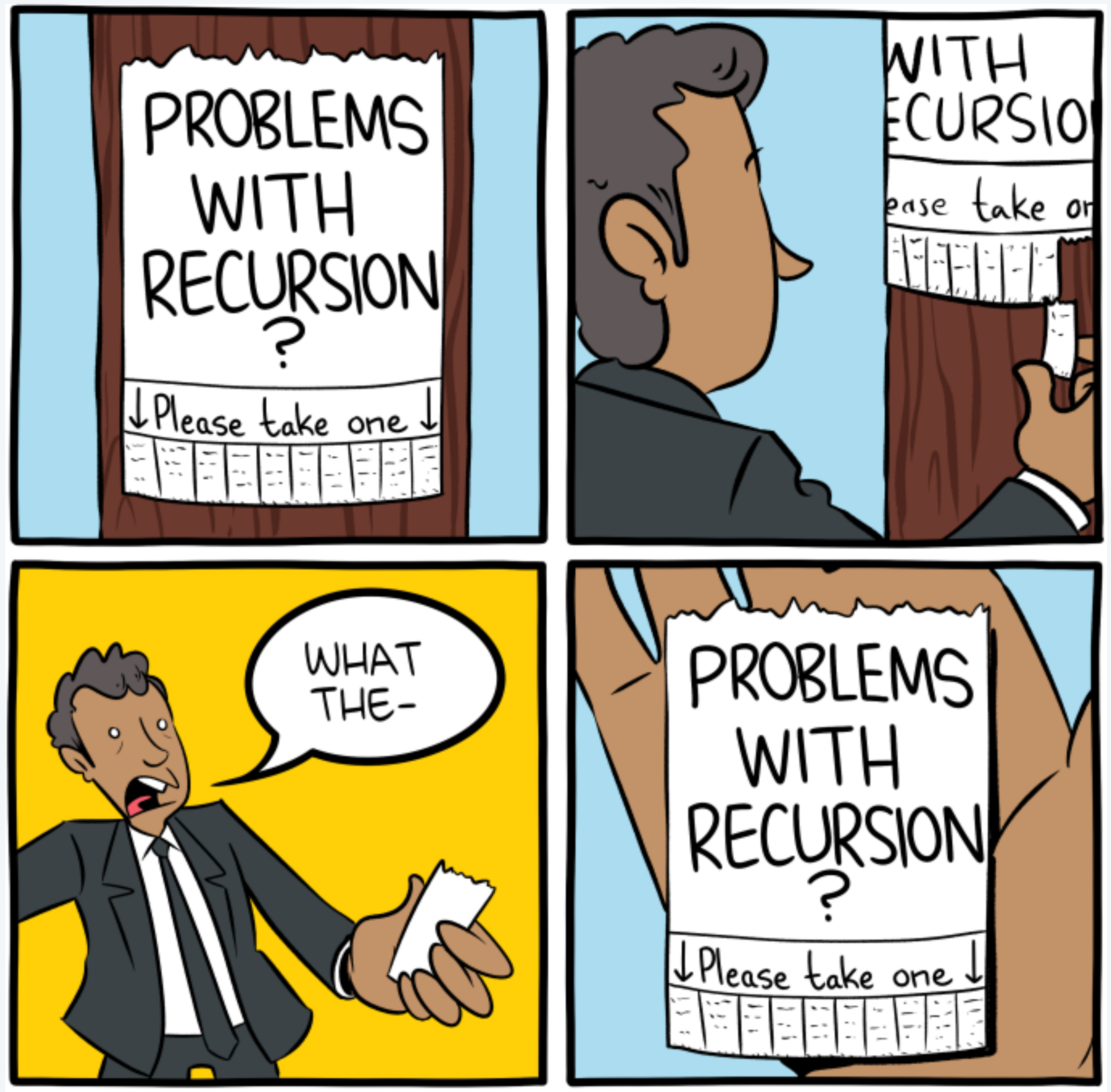
bug	buggy code	error	error message

	`public static int bad1(int n) {`
	`    return n * bad1(n-1);`
	`}`
		missing base case	`~/cos126/recursion> java-introcs Bug1 10`
			`Exception in thread "main"`
			`java.lang.StackOverflowError`
			`   at Bug1.java:4`
			`   at Bug1.java:4`
			`   at Bug1.java:4`
			`   at Bug1.java:4`
			`   ...`

	`public static int bad2(int n) {`
	`    if (n == 0) return 1;`
	`    return n * bad2(n + 1);`
	`}`
		reduction step does not converge to base case	`~/cos126/recursion> java-introcs Bug2 10`
			`Exception in thread "main"`
			`java.lang.StackOverflowError`
			`   at Bug2.java:4`
			`   at Bug2.java:4`
			`   at Bug2.java:4`
			`   at Bug2.java:4`
			`   ...`

stackoverflow

Problems with recursion?



https://www.smbc-comics.com

https://www.safelyendangered.com/comic/oh-bother

Recursion: quiz 1

What is printed by a call to `collatz(6)` ?

A. 6 3 10 5 16 8 4 2 1

B. 1 2 4 8 16 5 10 3 6

C. 2 4 8 16 5 10 3 6

D. 6 3 1

E. stack overflow error

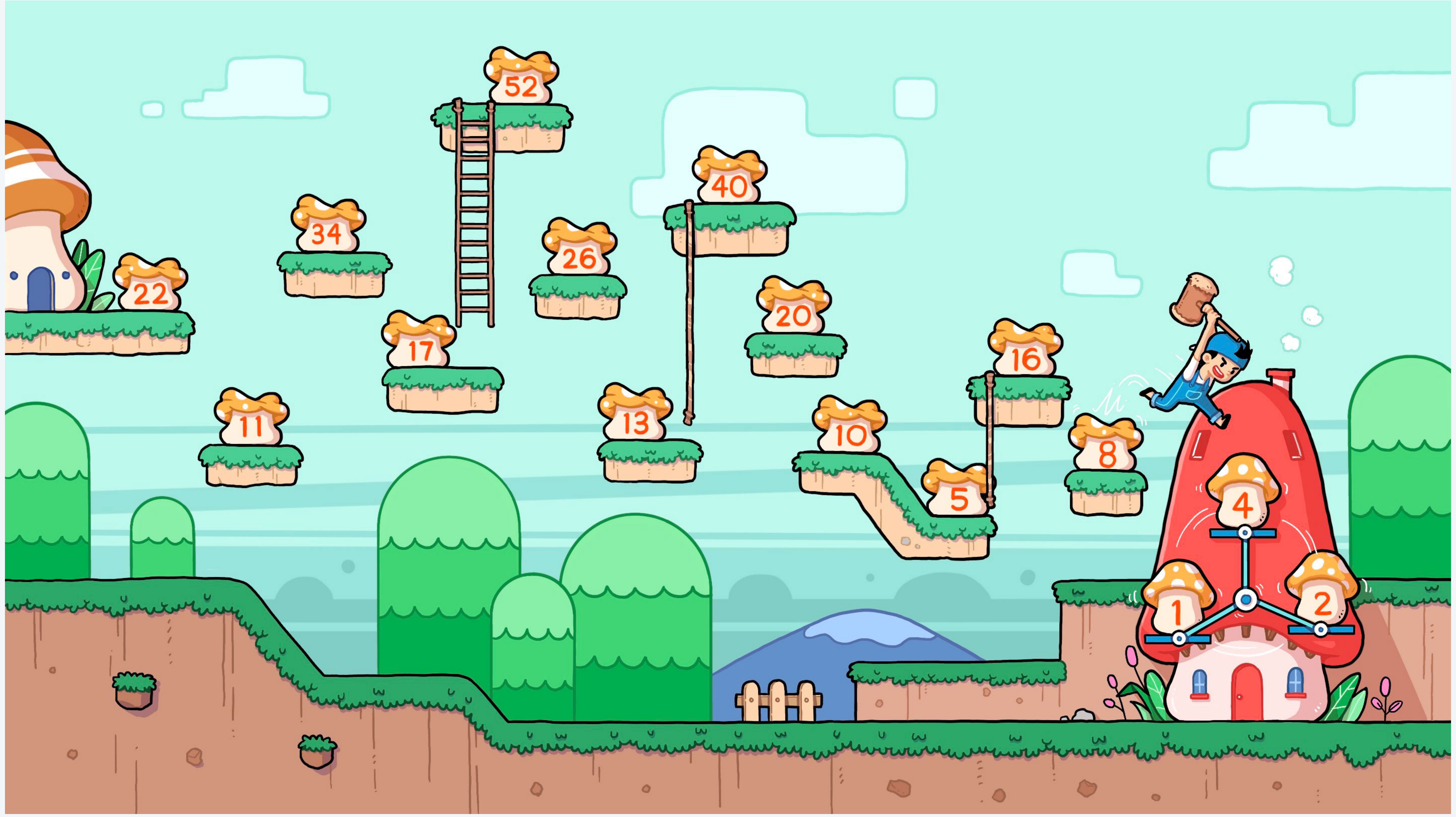
```
public static void collatz(int n) {
   StdOut.print(n + " ");
   if (n == 1) return;
   if (n % 2 == 0) collatz(n / 2);
   else collatz(3*n + 1);
}
```

integer division

Collatz sequence

Famous unsolved problem. Does `collatz(n)` terminate for all $n \geq 1$? ← *assume no arithmetic overflow*

Partial answer. Yes, for all $1 \leq n \leq 2^{68}$.



Saying the digits of a number

Goal. Say the decimal digits in a positive integer n.

- Base case: say nothing when n is zero.
- Reduction step: otherwise,
  - recursively say most significant digits ← `n / 10`
  - then, say the least significant digit ← `n % 10`

```
public static void sayDigits(int n) {
   if (n == 0) return;
   sayDigits(n / 10);
   StdAudio.play((n % 10) + ".wav");
}
```

play WAV file for digit 0–9

```
~/cos126/recursion> java-introcs SayDigits 126
[speaks "1 2 6"]

~/cos126/recursion> java-introcs SayDigits 25000
[speaks "2 5 0 0 0"]
```

```
sayDigits(126)
   sayDigits(12)
      sayDigits(1)
         sayDigits(0)
         play "1.wav"
      play "2.wav"
   play "6.wav"
```

function-call trace of `sayDigits(126)`

Q. How to say digits in binary (instead of decimal)?

A. Replace constant 10 with constant 2.

Recursion: quiz 2

What does `sayDigits(126)` **do with this version of** `sayDigits()` **?**

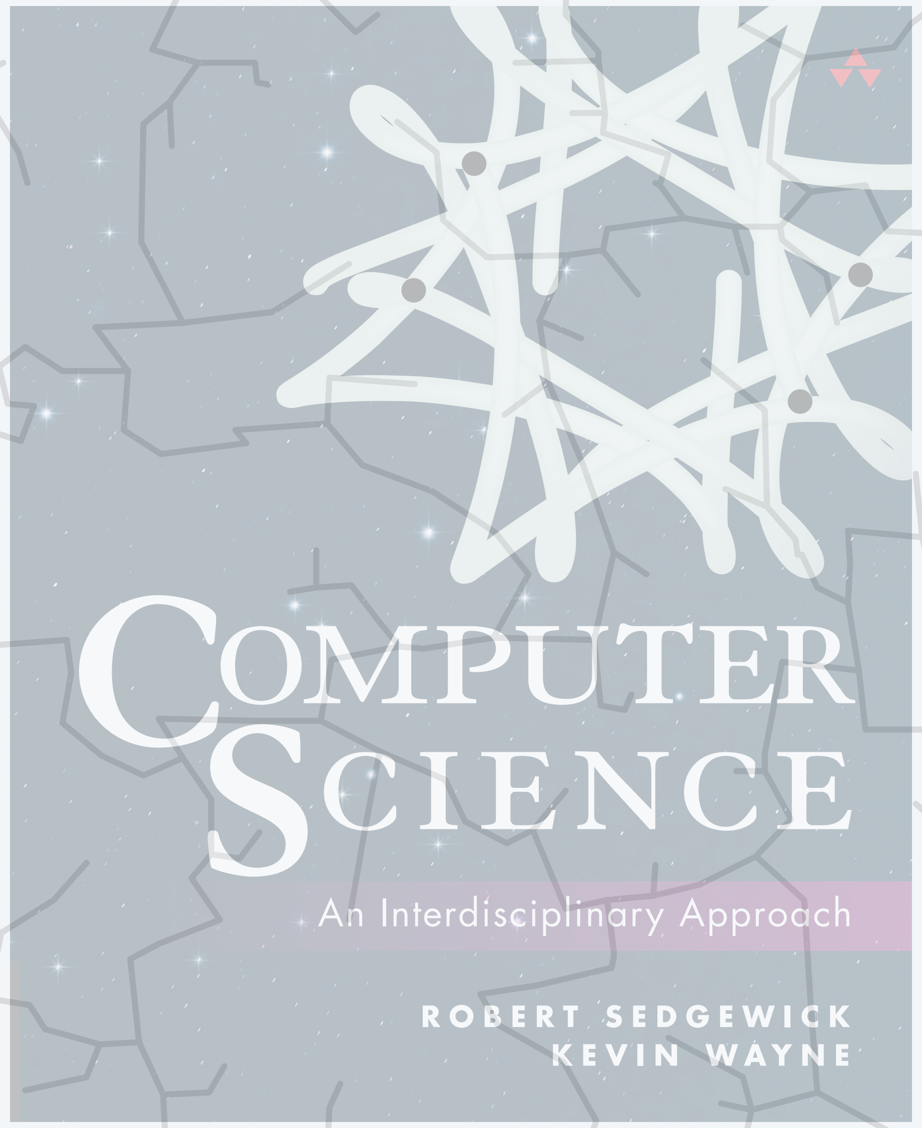
```
public static void sayDigits(int n) {
   if (n == 0) return;
   StdAudio.play((n % 10) + ".wav");
   sayDigits(n / 10);
}
```

the order of these two statements is switched

A. Speaks "1 2 6."

B. Speaks "6 2 1."

C. Stack overflow error.



https://introcs.cs.princeton.edu

2.3 Recursion

- foundations
- **a classic example**
- recursive graphics
- exponential waste

Warmup: ruler function (revisited)

Goal. Function `ruler(n)` that returns first $2^n - 1$ values of ruler function.

- Base case: empty for $n = 0$.
- Reduction step: sandwich n between two copies of `ruler(n-1)`.

1 2 1 3 1 2 1 4 1 2 1 3 1 2 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]);
      String result = ruler(n);
      StdOut.println(result);
   }
}
```

base case

reduction step

```
~/cos126/recursion> java-introcs Ruler 1
 1

~/cos126/recursion> java-introcs Ruler 2
 1 2 1

~/cos126/recursion> java-introcs Ruler 3
 1 2 1 3 1 2 1

~/cos126/recursion> java-introcs Ruler 4
 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

Tracing a recursive program

Draw the *function-call tree.*

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

function-call tree for ruler(4)

Which string does `ruler(3)` return for this version of `ruler()` ?

A. `"1 1 2 1 1 2 3 "`

B. `"1 2 1 3 1 2 1 "`

C. `"3 2 1 1 2 1 1 "`

D. `"3 2 2 1 1 1 1 "`

```
public static String ruler(int n) {
   if (n == 0) return "";
   return n + " " + ruler(n-1) + ruler(n-1);
}
```

Towers of Hanoi puzzle

A legend of uncertain origin.

- $n = 64$ disks of differing size; 3 poles; disks on middle pole, from largest to smallest.
- An ancient prophecy has commanded monks to move the disks to another pole.
- When the task is completed, the world will end.

Rules.

- Can move only one disk at a time.
- Cannot put a larger disk on top of a smaller disk.

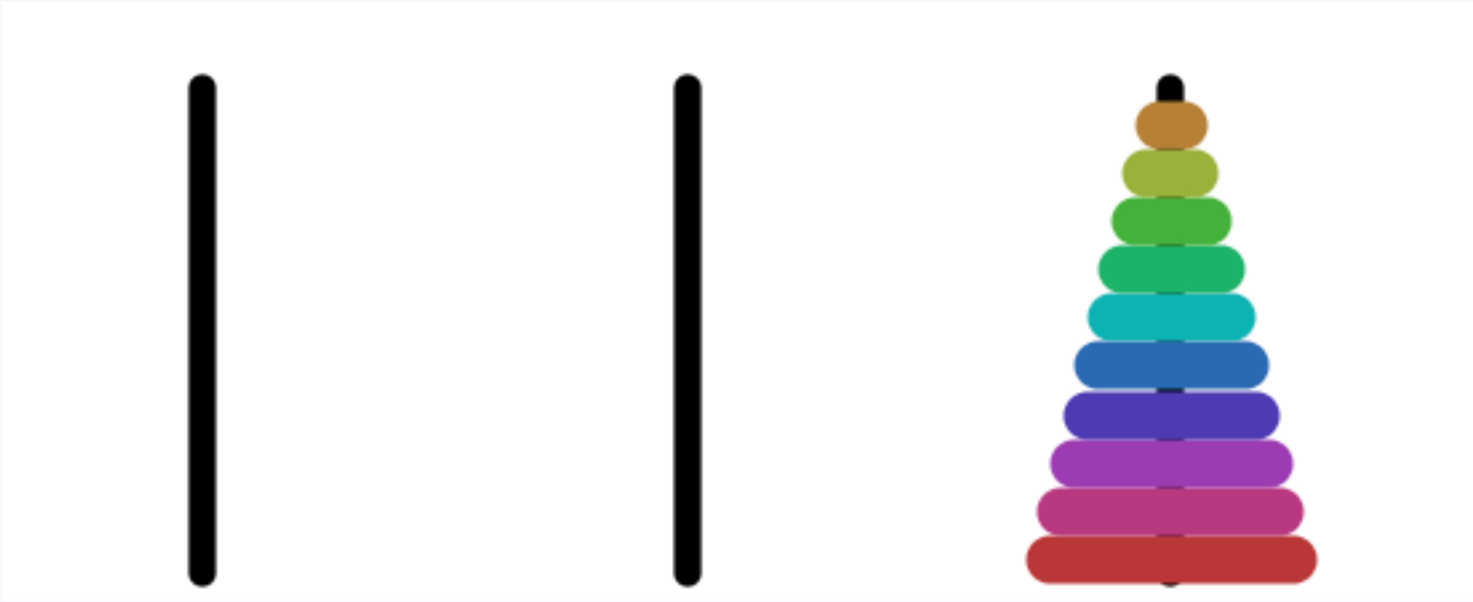
Q1. How to generate a list of instruction for monks.

Q2. When might the world end?

start

n = 10

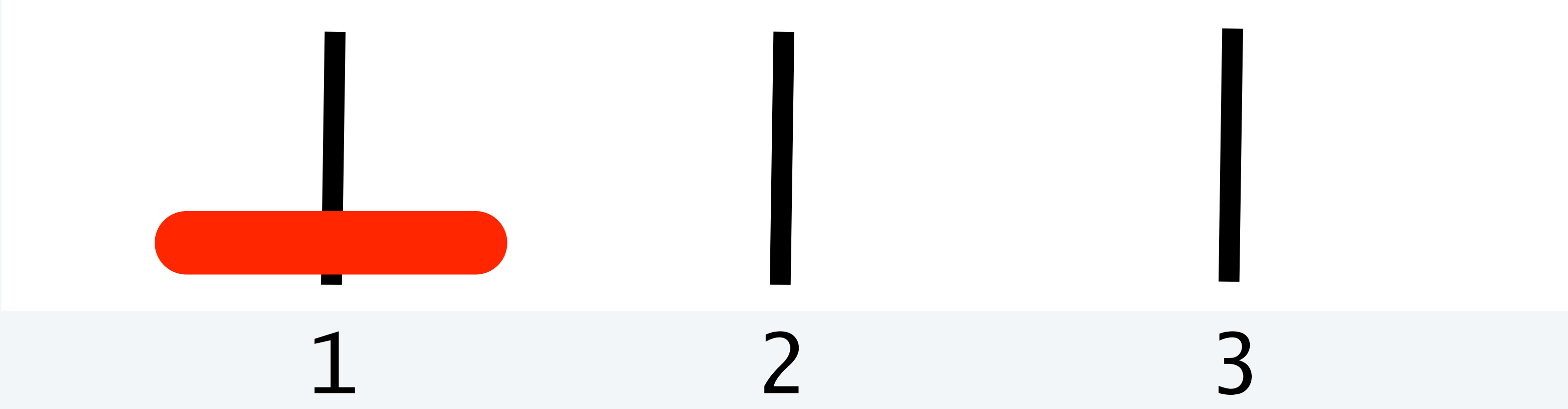
goal



Towers of Hanoi solution

For instructions, use cyclic wraparound.

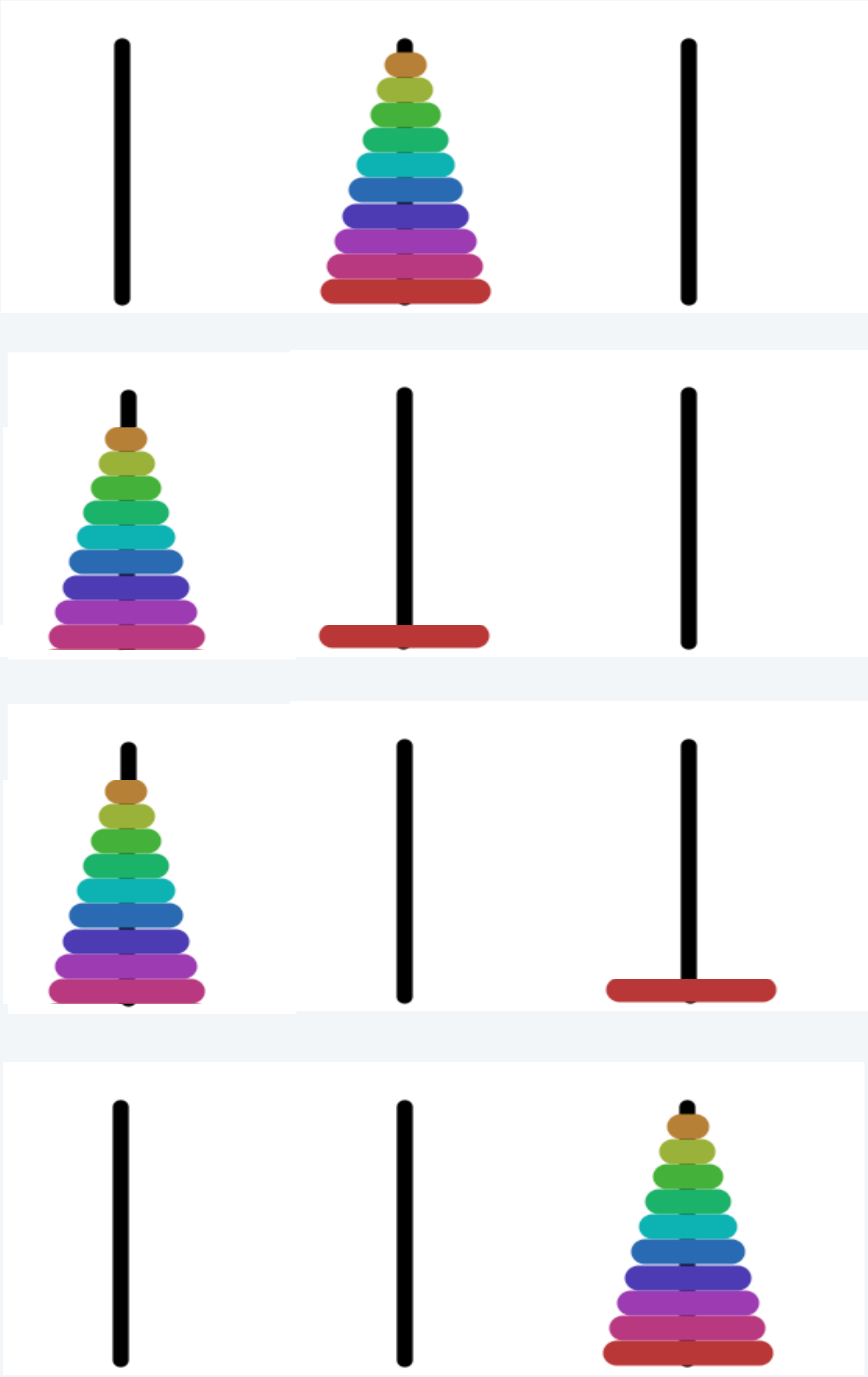
- Move right means 1 to 2, 2 to 3, or 3 to 1.
- Move left means 1 to 3, 3 to 2, or 2 to 1.



A recursive solution. [to move stack of n disks to the right]

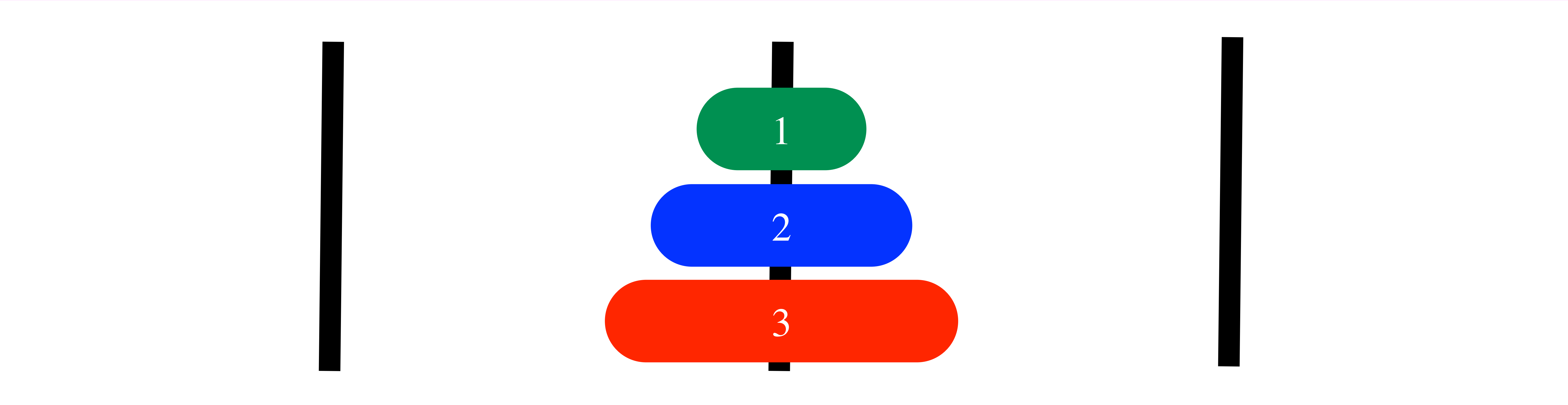
- Base case: if $n = 0$ disks, do nothing.
- Reduction step: otherwise,
  - move $n - 1$ smallest disks to the *left* (recursively)
  - move largest disk to the *right*
  - move $n - 1$ smallest disks to the *left* (recursively)

analogous to moving stack of n disks to the right

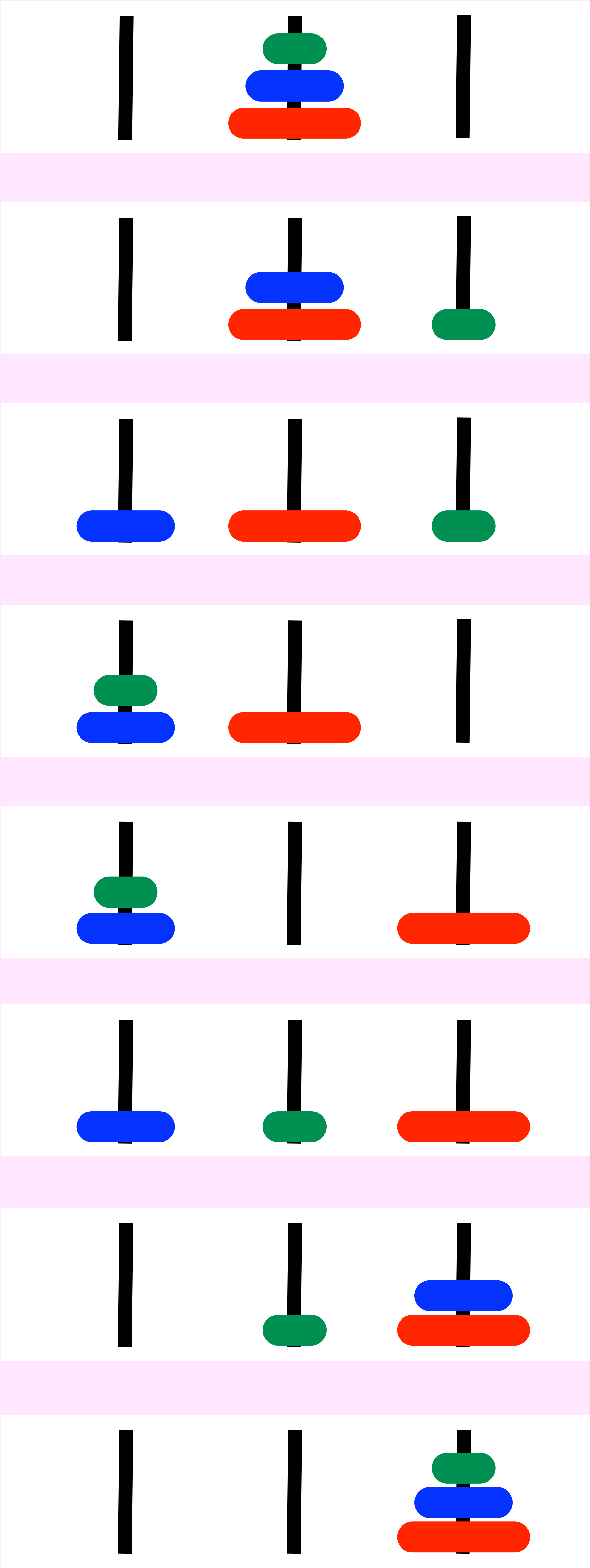


Towers of Hanoi solution (n = 3)

Notation. Label disks from smallest (1) to largest (n).



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



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

Towers of Hanoi: mutually recursive solution

Goal. Function `hanoiRight(n)` that returns instructions for n disk puzzle. ← *and also a similar function* `hanoiLeft(n)`

- Base case: if $n = 0$ disks, do nothing.
- Reduction step: otherwise, sandwich moving disk n right between two calls to `hanoiLeft(n-1)`

```
public class Hanoi {
   public static String hanoiRight(int n) {
      if (n == 0) return " ";
      return hanoiLeft(n-1) + n + "R" + hanoiLeft(n-1);
   }

   public static String hanoiLeft(int n) {
      if (n == 0) return " ";
      return hanoiRight(n-1) + n + "L" + hanoiRight(n-1);
   }

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

move stack of n disks right (hanoiRight)

move stack of n disks left (hanoiLeft)

⚠ **concise but tricky code; read carefully!**

```
~/cos126/recursion> java-introcs Hanoi 3
 1R 2L 1R 3R 1R 2L 1R

~/cos126/recursion> java-introcs Hanoi 4
 1L 2R 1L 3L 1L 2R 1L 4R 1L 2R 1L 3L 1L 2R 1L
```

Function-call tree for towers of Hanoi

Properties.

- Each disk always moves in the same direction.
- Moving smallest disk always alternates with (unique legal) move not involving smallest disk.
- Solution to puzzle with n disks makes $2^n - 1$ moves.

Answers for towers of Hanoi

Q. How to generate instructions for monks?

A1. [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 …

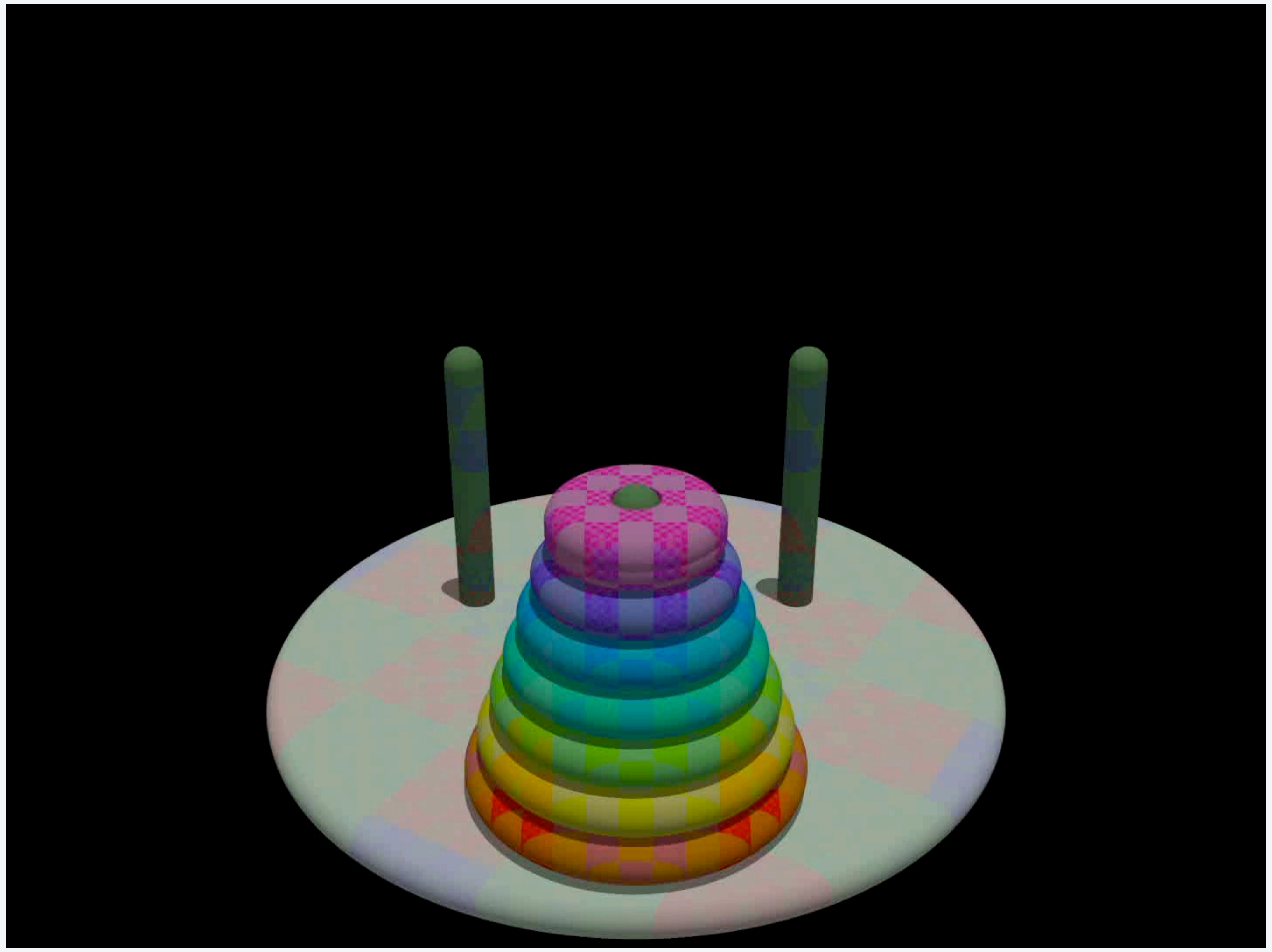
A2. [short form] Alternate 1L with the only legal move not involving disk 1.

if n is odd, alternate 1R

Q. When might the world end?

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

recursive solution provably uses fewest moves



Recursion vs. iteration

Fact 1. Any recursive program can be rewritten with loops (and no recursion).

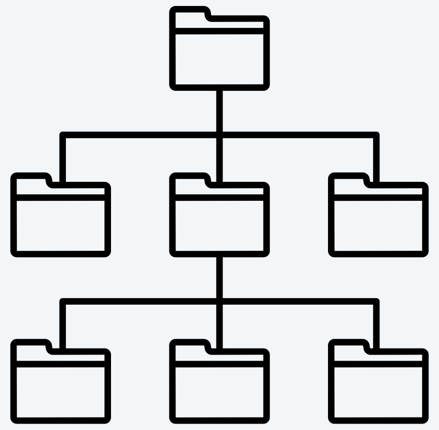
Fact 2. Any program with loops can be rewritten with recursion (and no loops).

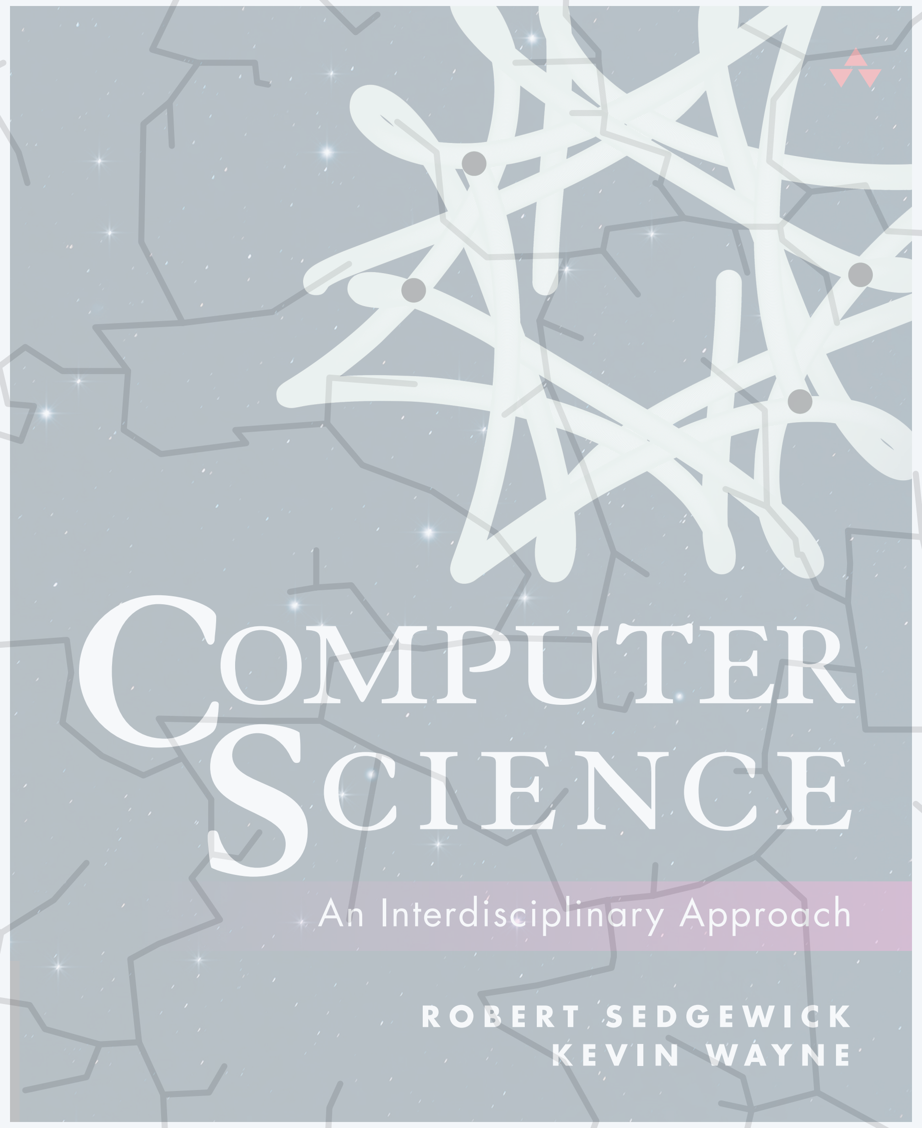
loops	recursion
more memory efficient (no function-call stack)	*concise and elegant code*
easier to trace code (fewer variables)	*easier to reason about code (fewer mutable variables)*

Q. When should I use recursion?

A1. The problem is naturally recursive (e.g., towers of Hanoi).

A2. The data is naturally recursive (e.g., filesystem with folders).



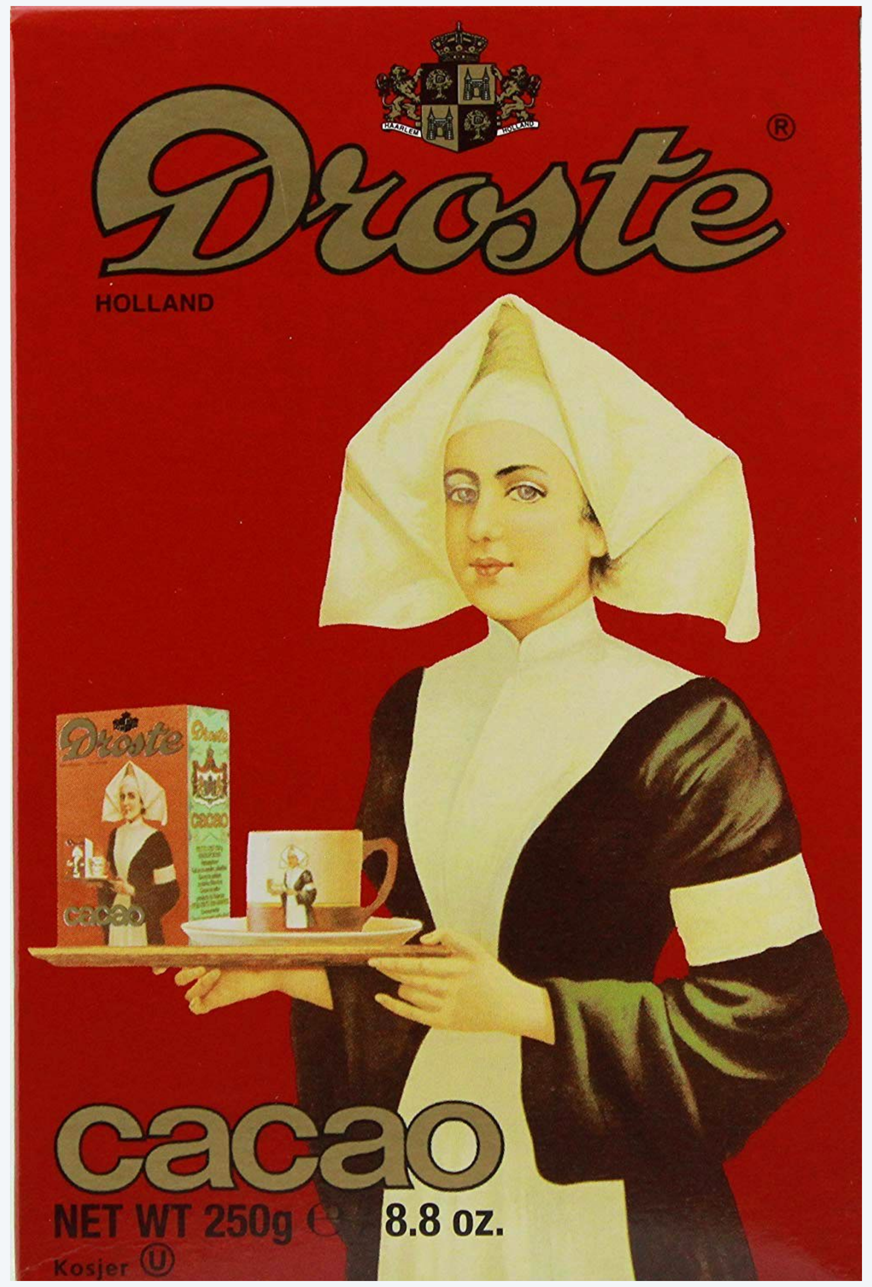


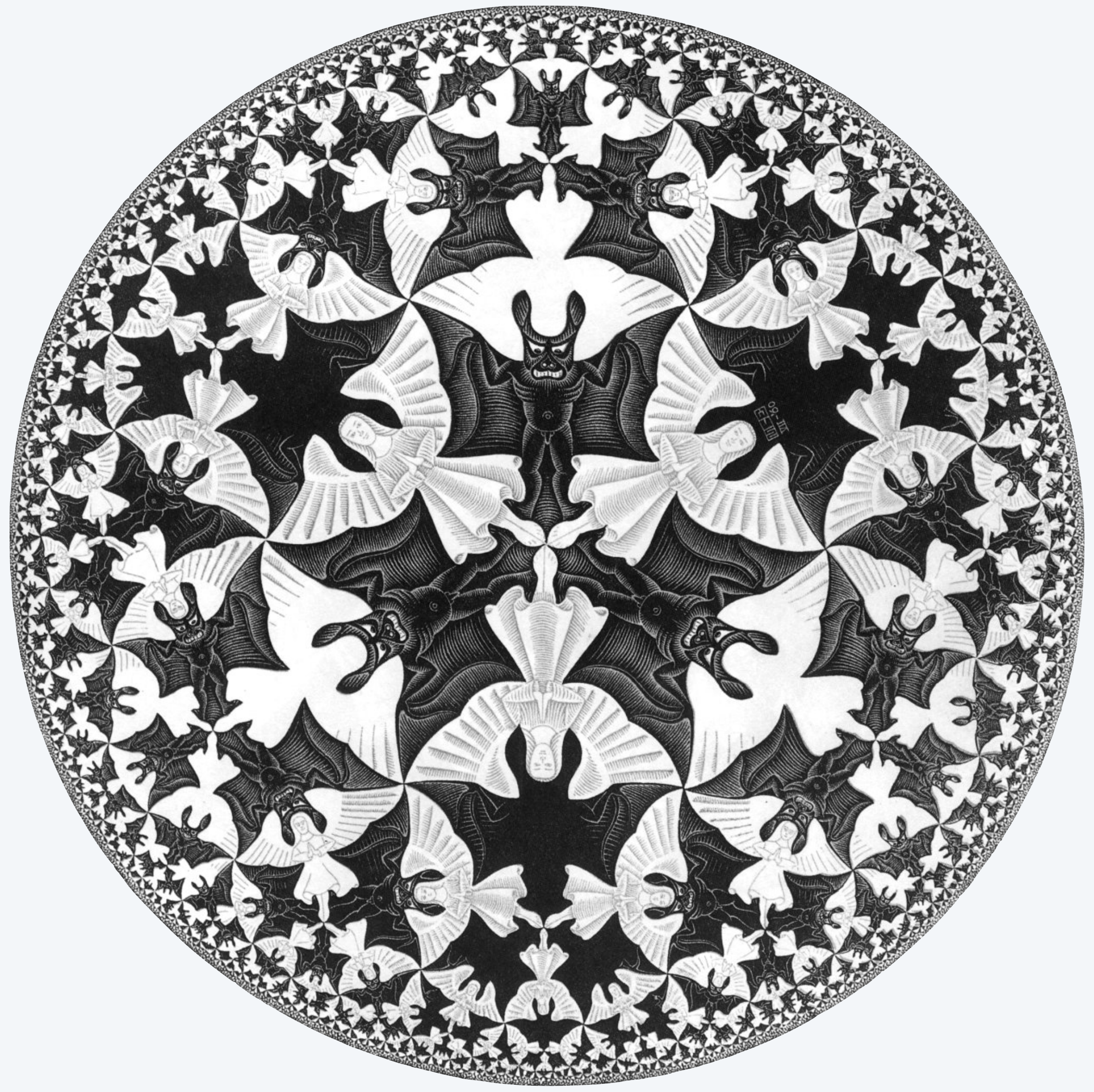
https://introcs.cs.princeton.edu

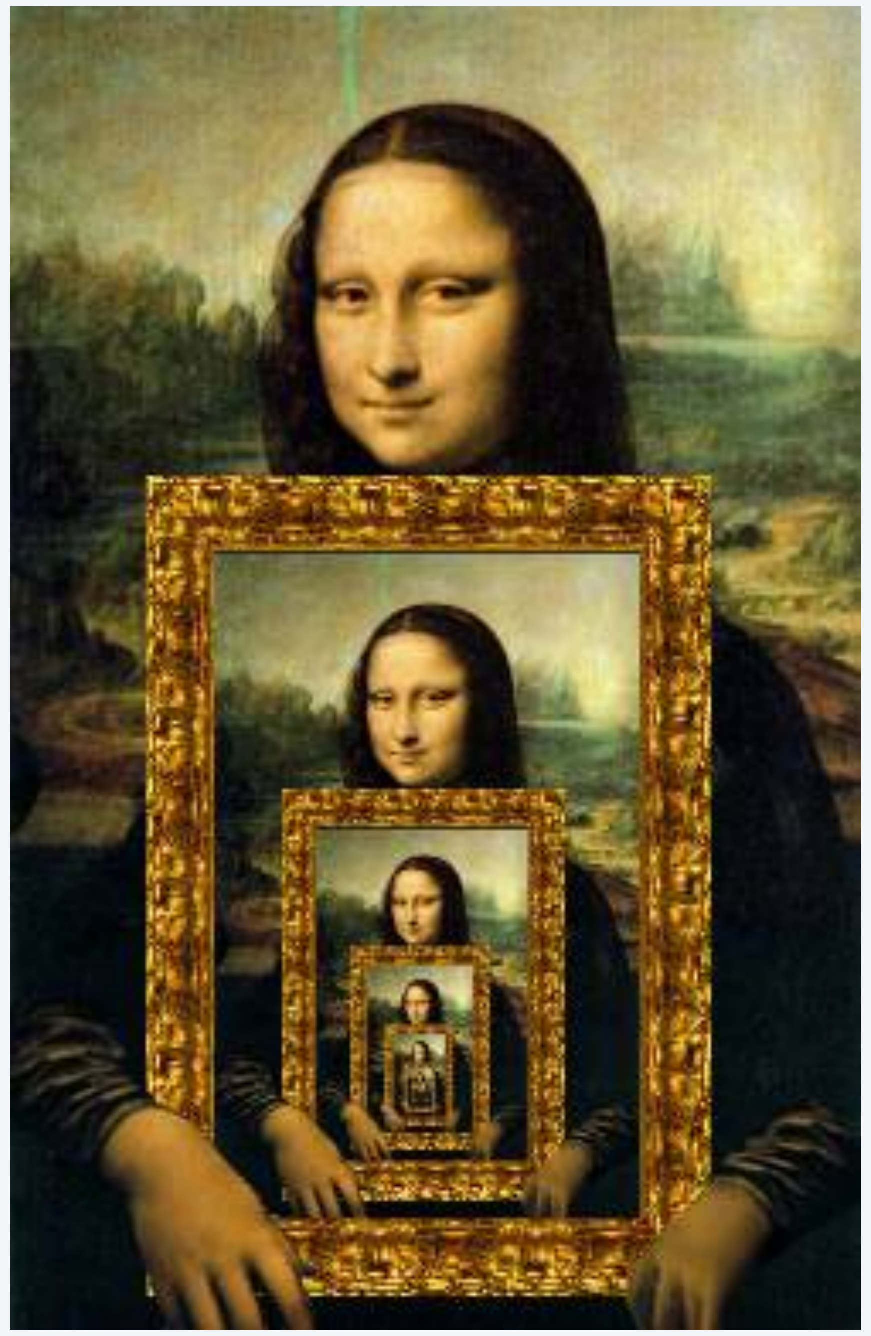
2.3 Recursion

- *foundations*
- *a classic example*
- ***recursive graphics***
- *exponential waste*

Recursive graphics in the wild







WEEKEND Arts FINE ARTS LEISURE
N E41
FRIDAY, DECEMBER 15, 2006
The New York Times

Design Life Now *at the Cooper-Hewitt National Design Museum includes this toy from the New York company Kidrobot.*

Fruits of Design, Certified Organic

ROBERTA SMITH
ART REVIEW

It's Triennial time at the Cooper-Hewitt National Design Museum. This means that the former Andrew Carnegie mansion is up to its neck in mostly American design from the last three years. Like its predecessors, "Design Life Now," the museum's third National Design Triennial, is a crazed affair that illuminates a volatile, contradictory, ever-expanding field but fails to call it to order.

The exhibition has been organized by the Cooper-Hewitt curators Barbara Bloemink, Ellen Lupton and Matilda McQuaid and a guest, Brooke Hodge, a curator at the Museum of Contemporary Art, Los Angeles.

Once again the Triennial answers the question "What's design?" with the evasive catchall "What's not?" Covering so many bases so equivocally, it never gets around to tackling the weightier questions of "What is good design?" or, more to the point, "What is design good for?" It refuses to take sides on the issue of whether design should aim for social or environmental benefit or serve a relatively decorative purpose. Still, the show's benefits are many, even if you have to work for them.

The displays here range from genius to schlock, delightful to dispiriting. They cover life-extending innovations, completely frivolous reiterations of received ideas (far too many of which trace to Surrealism) and more varieties of recycling than you can easily count. Fashion, building materials, furniture, toys, theatrical sets, jewelry and textiles, medical and military hardware, all qualify as design according to this exhibition.

The main point comes across loud and clear: design permeates every aspect of contemporary life. Everything that exists is designed, whether natural or cultural. And while all of nature's designs are intelligent, whether you go by Darwin or the Bible, the human kind are much

Continued on Page 51

Lars Klove for The New York Times
From "The Yale Book of Quotations" to "Postcards From Mars," a selection of the best holiday books.

The Gifts to Open Again and Again

WILLIAM GRIMES
BOOKS

I've made my list, and I'm checking it twice. It's a list of the qualities that make the ideal holiday book, and after carefully considering the books of Christmas past, I have come up with some guidelines. A gift book should either be no surprise or a big surprise: the one you always wanted or the one you never knew you wanted. It should either be expensive and large, or cheap and small. It should be high-minded or totally frivolous. And no matter what, it should not require sustained attention, which is impossible during the yuletide season. My gift selections, chosen entirely at random but with exquisite taste, satisfy at least two of these requirements.

Let's open the big presents first. The season's whopper, in every way, is "New York 2000," the fifth installment in Robert A. M. Stern's architectural history of New York. The series starts in 1880, when 10 stories qualified as a skyscraper, and has now caught up to the new millennium. Taken together, the volumes make an enormous, endlessly fascinating family scrapbook for New Yorkers, who can coo over baby pictures of the Flatiron Building and leaf forward, through many hundreds of pages and thousands of photographs, to the big, grown-up New York of the Lipstick Building, countless Trump projects and the new Tweed Courthouse.

At 1,520 pages and 10 pounds 12 ounces, "New York

Continued on Page 46

Serkan Ozkaya

Black, White and Read All Over Over

By RANDY KENNEDY

In one of Jorge Luis Borges's best-known short stories, "Pierre Menard, Author of the Quixote," a 20th-century French writer sets out to compose a verbatim copy of Cervantes's 17th-century masterpiece simply because he thinks he can, originality perhaps not being all it's cracked up to be.

He manages two chapters word for word, a spontaneous duplicate that Borges's narrator finds to be "infinitely richer" than the original because it contains all manner of new meanings and inflections, wrenched as it is from its proper time and context.

When a young Turkish artist named Serkan Ozkaya set out recently to practice his skills as a copyist — a scrivener, as he says — his goals were a little less ambitious than channeling Cervantes. He simply wanted to draw and see printed a faithful copy of all the type and pictures planned for a broadsheet page of this newspaper: this very page you are reading right now, which shows his version of the page you are reading right now, which shows his version of his version of the page you are reading right now, which. . . .

Do not be alarmed: There has been no break in the space-time-newsprint continuum.

Mr. Ozkaya, a 33-year-old artist who lives and works in Istanbul, did not propose this exercise in handmade surrealism because of a particular love of calligraphy or newspapers or, for that matter, even drawing, which he admits he is not very good at. The

Continued on Page 51

Serkan Ozkaya's drawing of the page you are reading right now, showing his drawing of the page you are reading right now, showing. . . .

Divine and Devotee Meet Across Hinges

HOLLAND COTTER
ART REVIEW

WASHINGTON — For toothache, dial St. Apollonia ASAP. She'll bring relief in a flash. Keep St. Matthew, ex-banker, in mind in April; he'll help get your taxes in shape. Everyone knows that a prayer to St. Roch, protector from plague, is as good as a flu shot, and that lightning will never strike when St. Barbara's on the job.

Most important, for dire and intangible problems — moral confusion, inconsolable grief, sickness of soul — there's the Virgin. Day and night she's on the toll-free hot line offering gentle attention and prudent advice.

To European Christians half a millennium ago, the Virgin and a raft of familiar saints were the exalted personnel in a kind of celestial welfare system, available to all believers. And one quick way to access its benefits was through devotional painting of the kind found in "Prayers and Portraits: Unfolding the Netherlandish Diptych" at the National Gallery of Art.

Probably nothing in Western art comes closer to formal perfection than these pictures, produced by the likes of Jan van Eyck, Rogier van der Weyden and Hugo van der Goes across an area that now encompasses the Netherlands, Belgium, Luxembourg and parts of France. These painters were pictorial magicians, creating visual worlds, cosmically abstract and microscopically realistic, of peerless breadth.

You see all of this in one glance at the 40 double-panel paintings, or diptychs, here. Then you learn gradually as you move through the show how diptych paintings have been unmade and remade, broken up and reconfigured, over the centuries, with the result that few survive in their intended form.

"Prayers and Portraits" is an attempt to restore that form, at least to a few of them. It brings art historians and art

Continued on Page 44

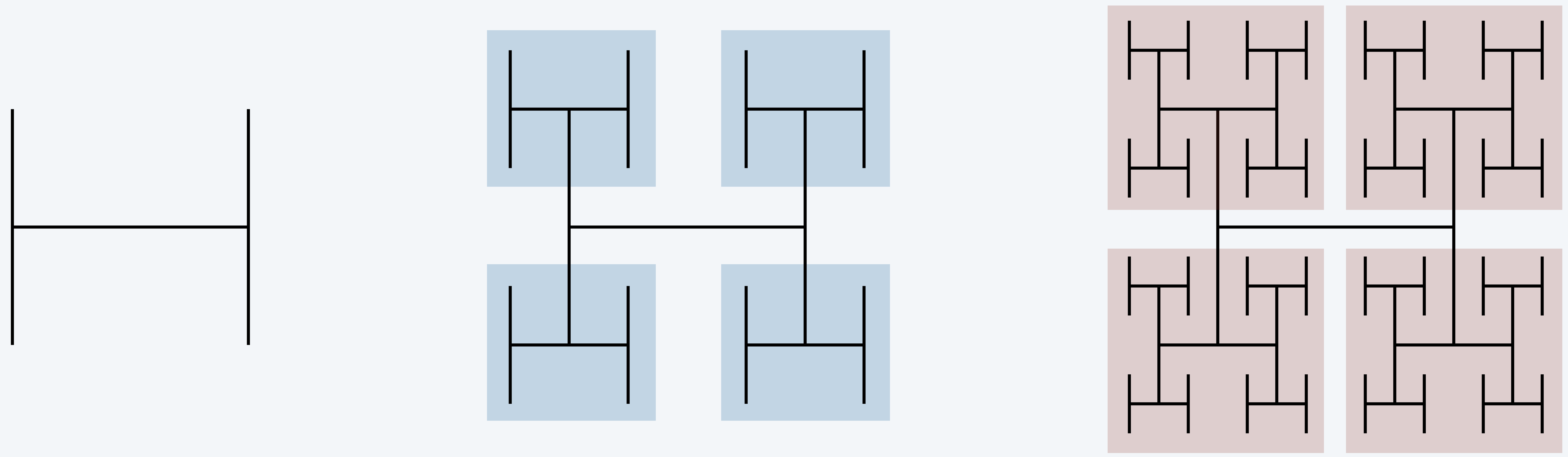
Left, Gemäldegalerie, Berlin; National Gallery of Art, Washington

Prayers and Portraits: *Unfolding the Netherlandish Diptych. Two panels of an early 16th-century diptych by Michel Sittow, left, are reunited in an exhibition at the National Gallery of Art in Washington through Feb. 4.*

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

H–tree of order n.

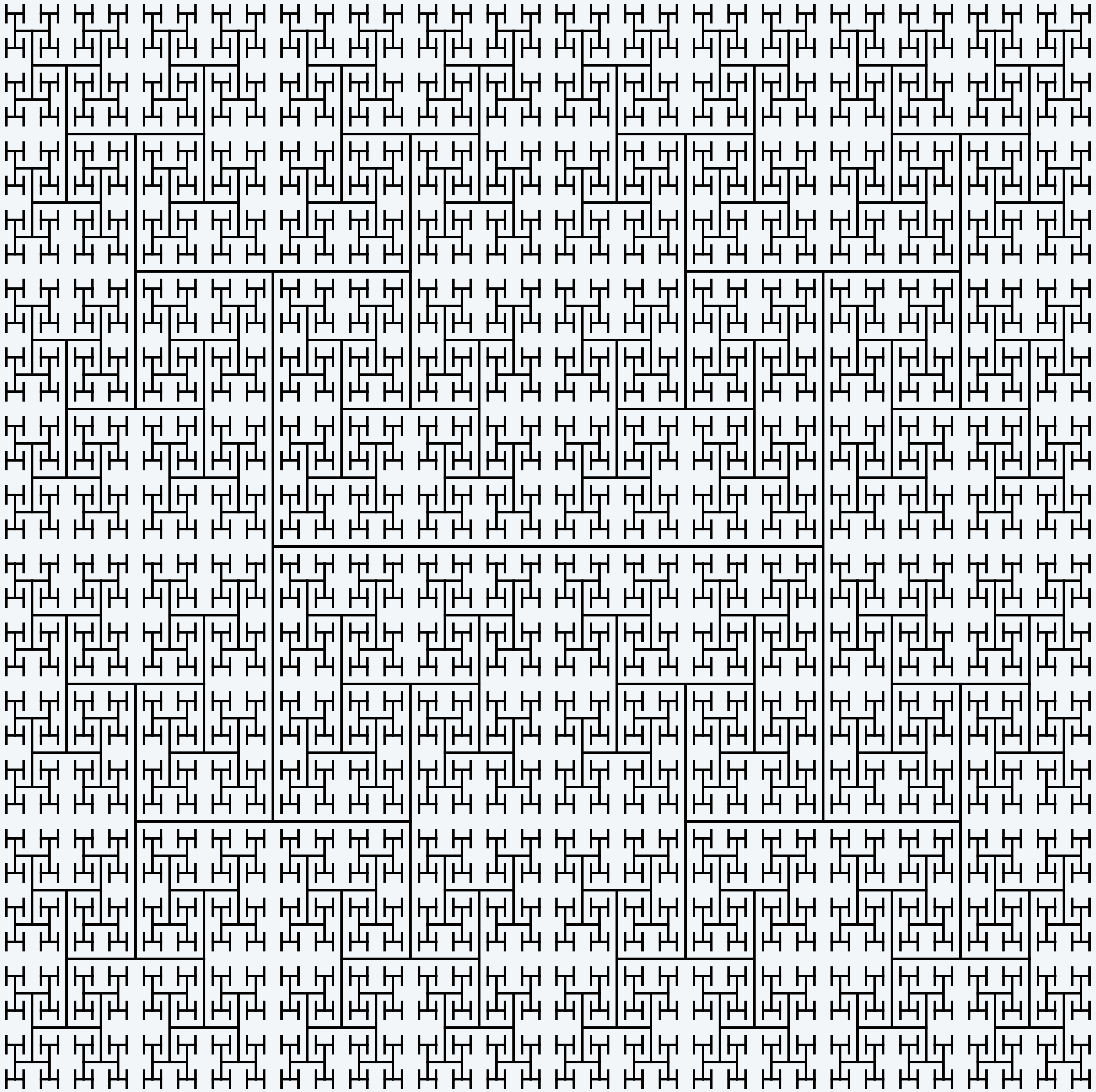
- Base case: if n is 0, draw nothing.
- Reduction step:
  - draw an H
  - draw four H–trees of order $n - 1$ and half the size, centered at the tips of the H



order 1 order 2 order 3

H-trees

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



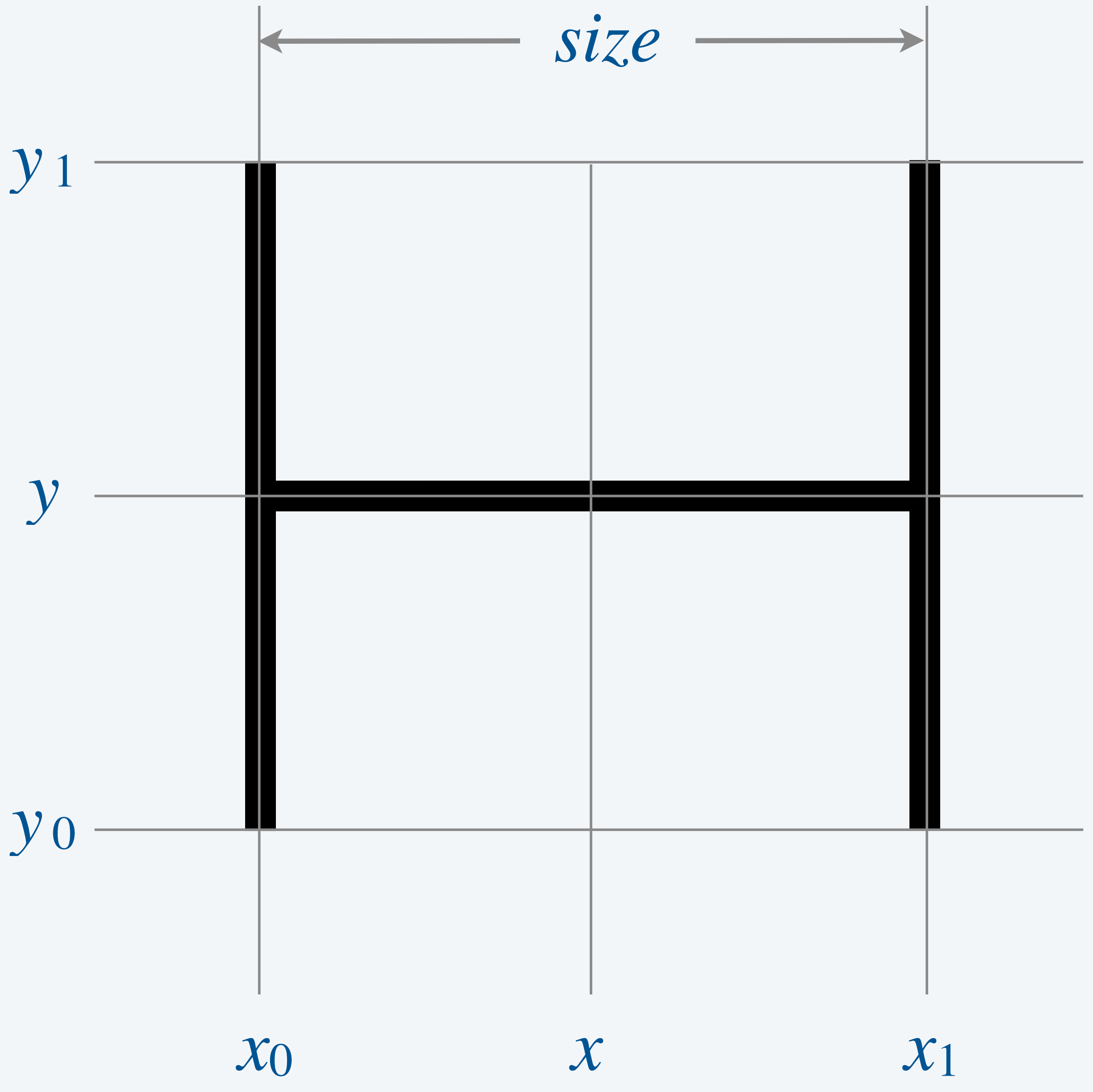
Recursive H-tree implementation

```
public class Htree {

   public static void draw(int n, double size, double x, double y) {
      if (n == 0) return;
      double x0 = x - size/2, x1 = x + size/2;
      double y0 = y - size/2, y1 = y + size/2;
      StdDraw.line(x0,  y, x1,  y);
      StdDraw.line(x0, y0, x0, y1);
      StdDraw.line(x1, y0, x1, y1);
      draw(n-1, size/2, x0, y0);   // lower left
      draw(n-1, size/2, x0, y1);   // upper left
      draw(n-1, size/2, x1, y0);   // lower right
      draw(n-1, size/2, x1, y1);   // upper right
   }

   public static void main(String[] args) {
      StdDraw.setPenRadius(0.005);
      int n = Integer.parseInt(args[0]);
      draw(n, 0.5, 0.5, 0.5);
   }
}
```

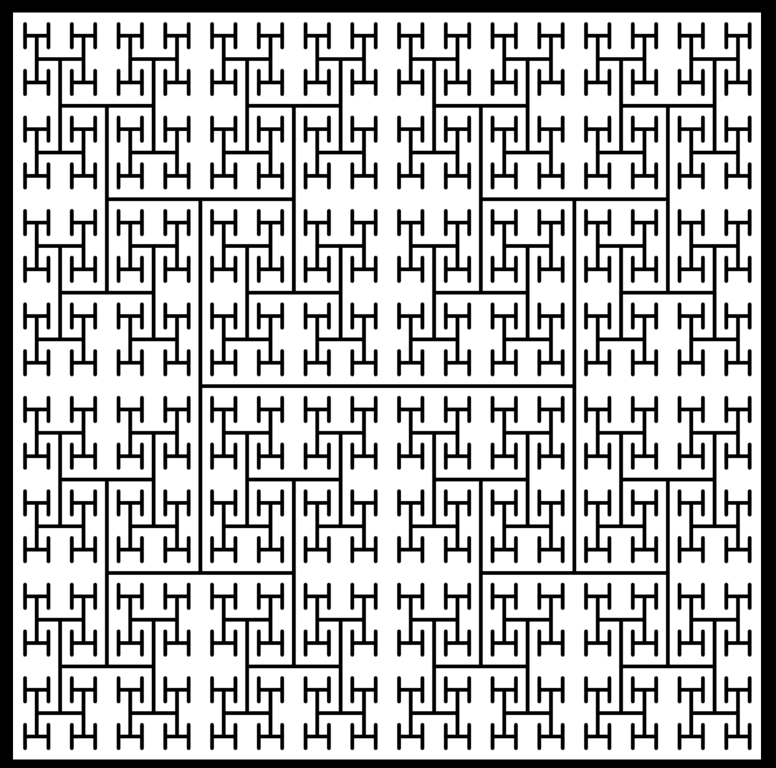
endpoints

draw the H (non-recursive)

draw four half-size H-trees (recursively)



~/cos126/recursion> java-introcs Htree 5

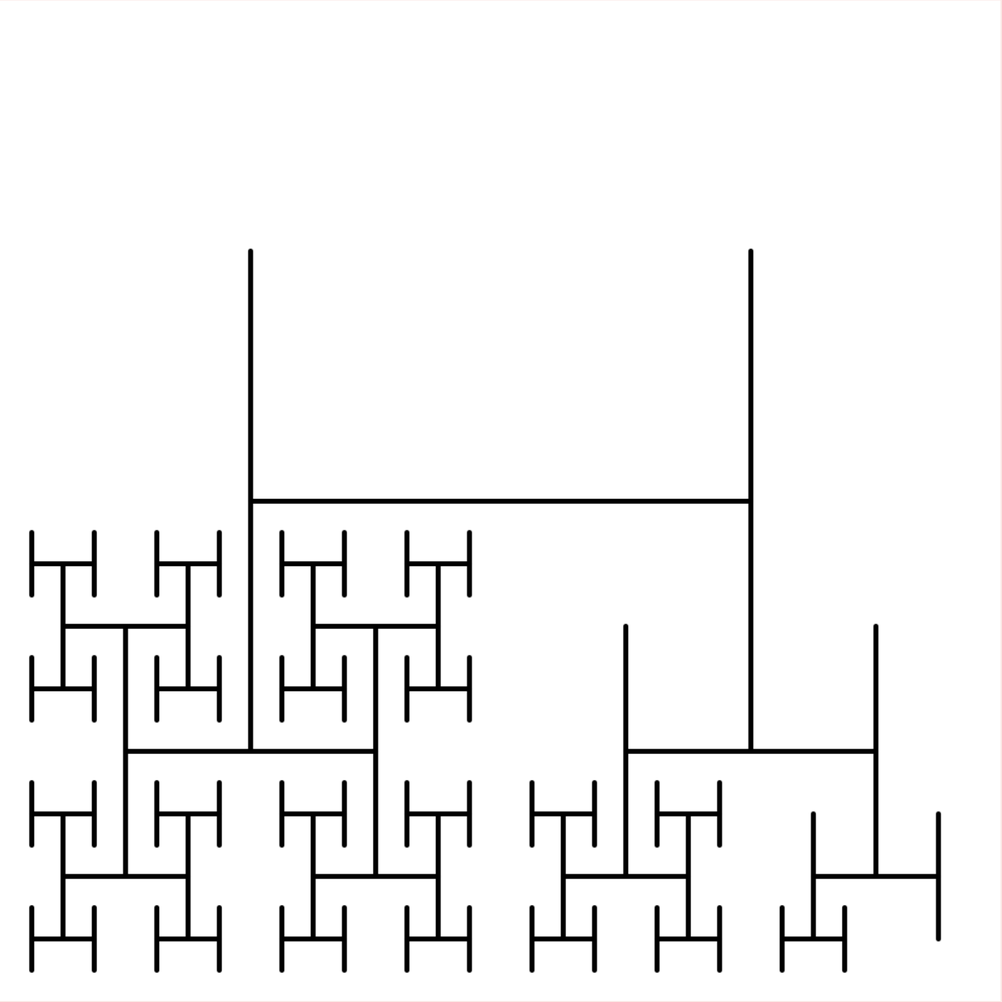


Recursion: quiz 3

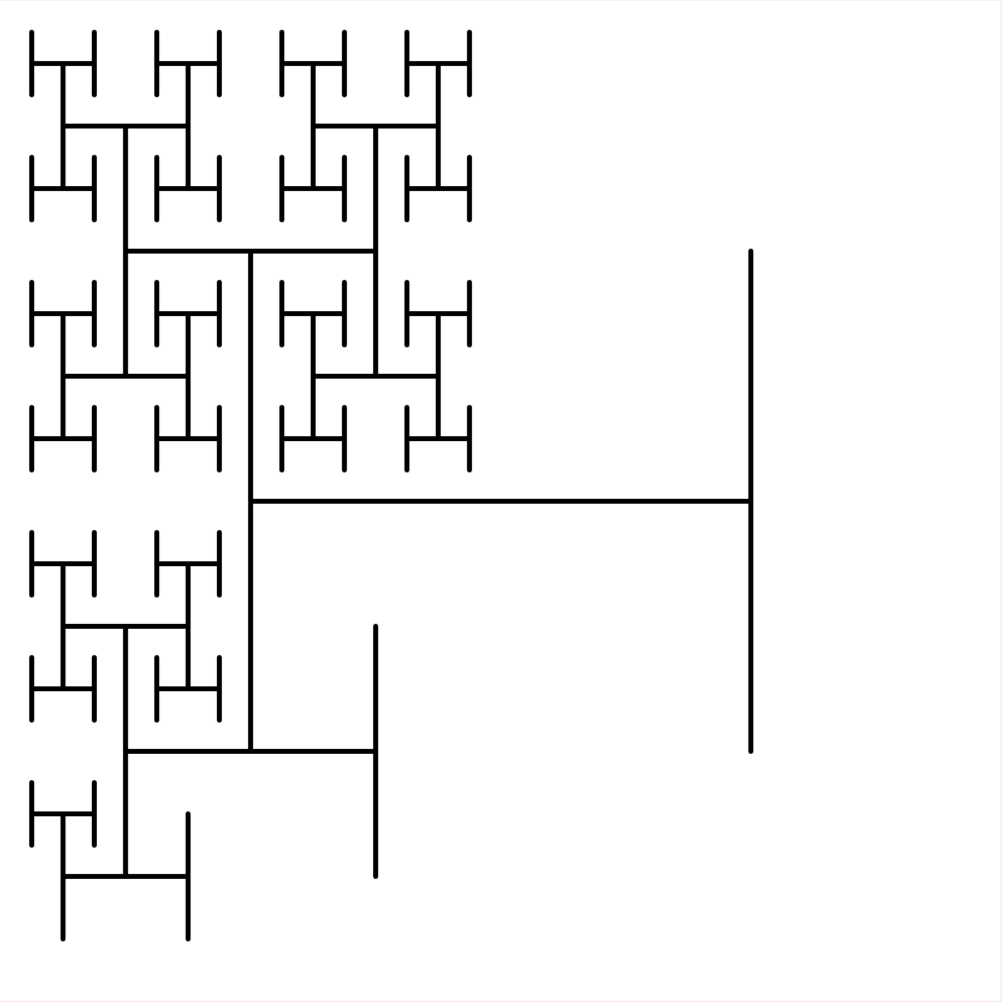
Suppose that `Htree` (with `n = 4`) is stopped after drawing the 30th H.

Which drawing will result?

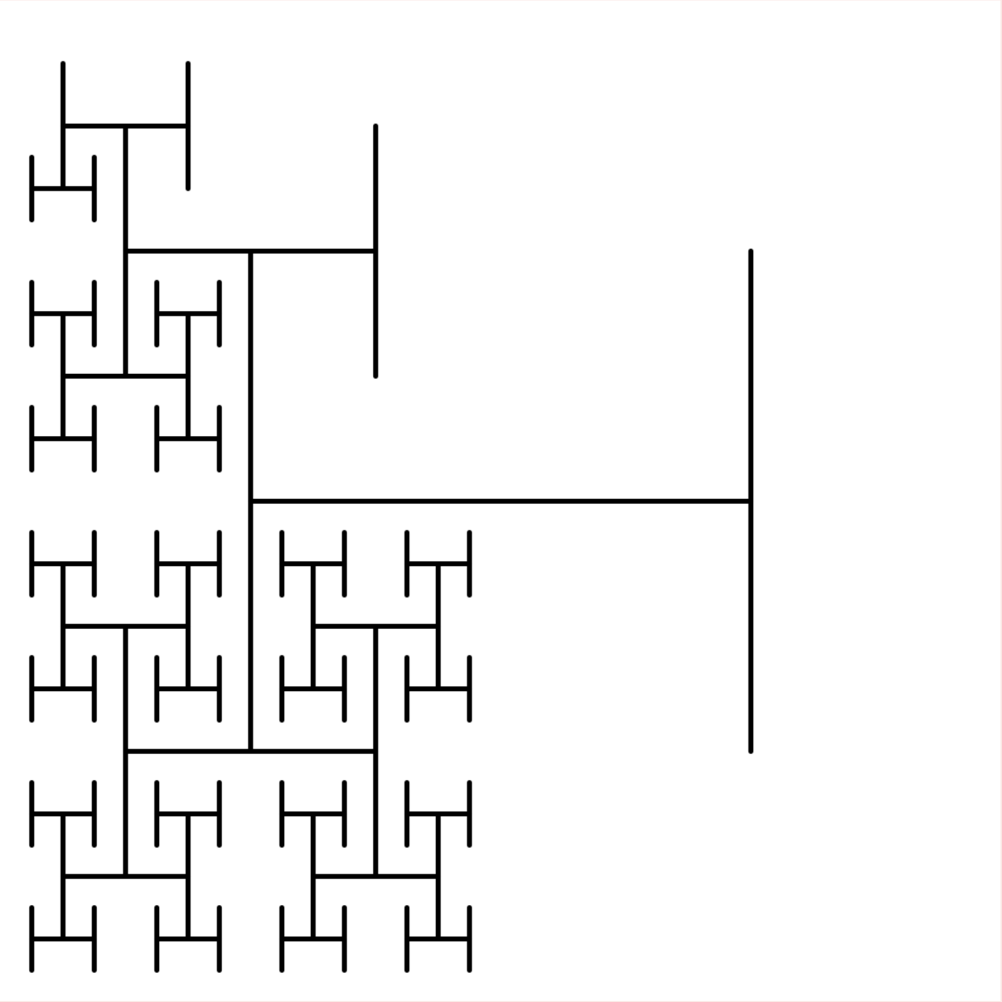
A.



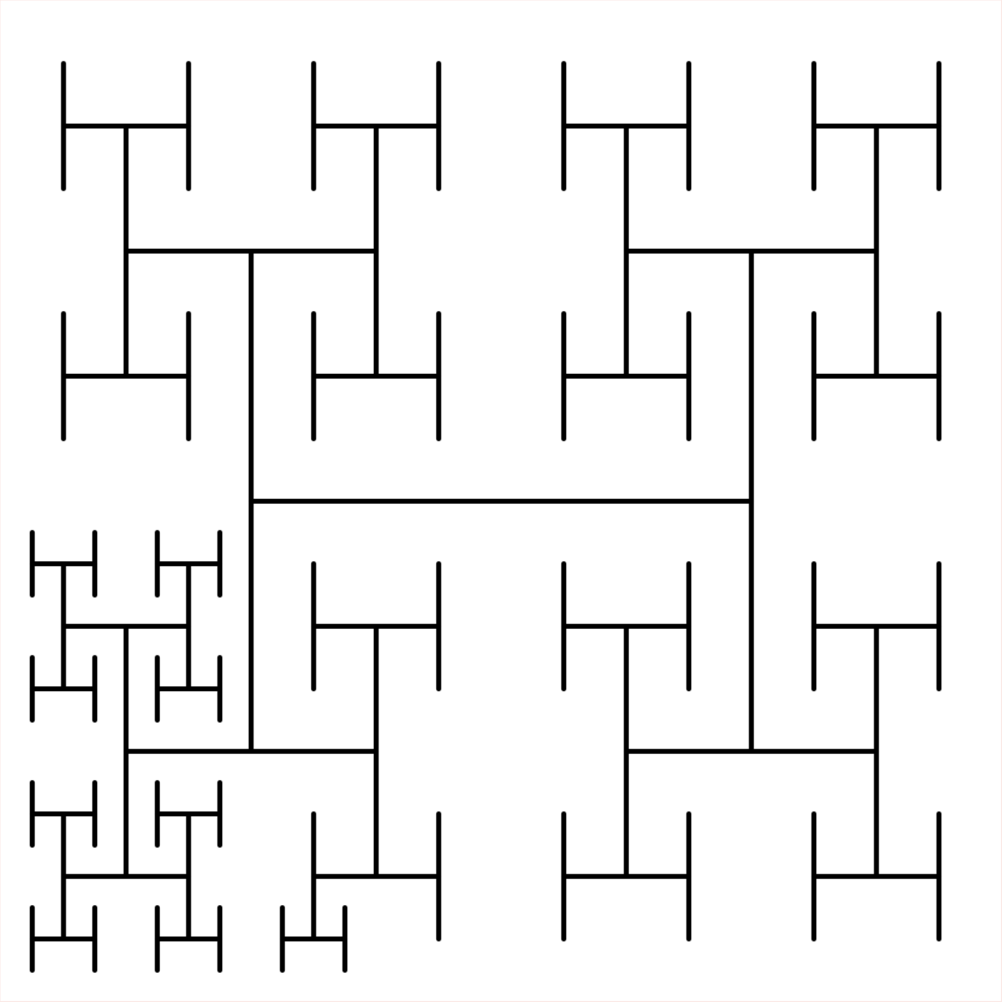
B.



C.



D.



Super H-tree implementation

Q. What will happen if we add the following statements to `draw()`, just before the recursive calls?

```
double freq = Synth.midiToFrequency(n + 45);
double duration = 0.25 * n;
double[] a = Synth.supersaw(freq, duration);
StdAudio.play(a);
```

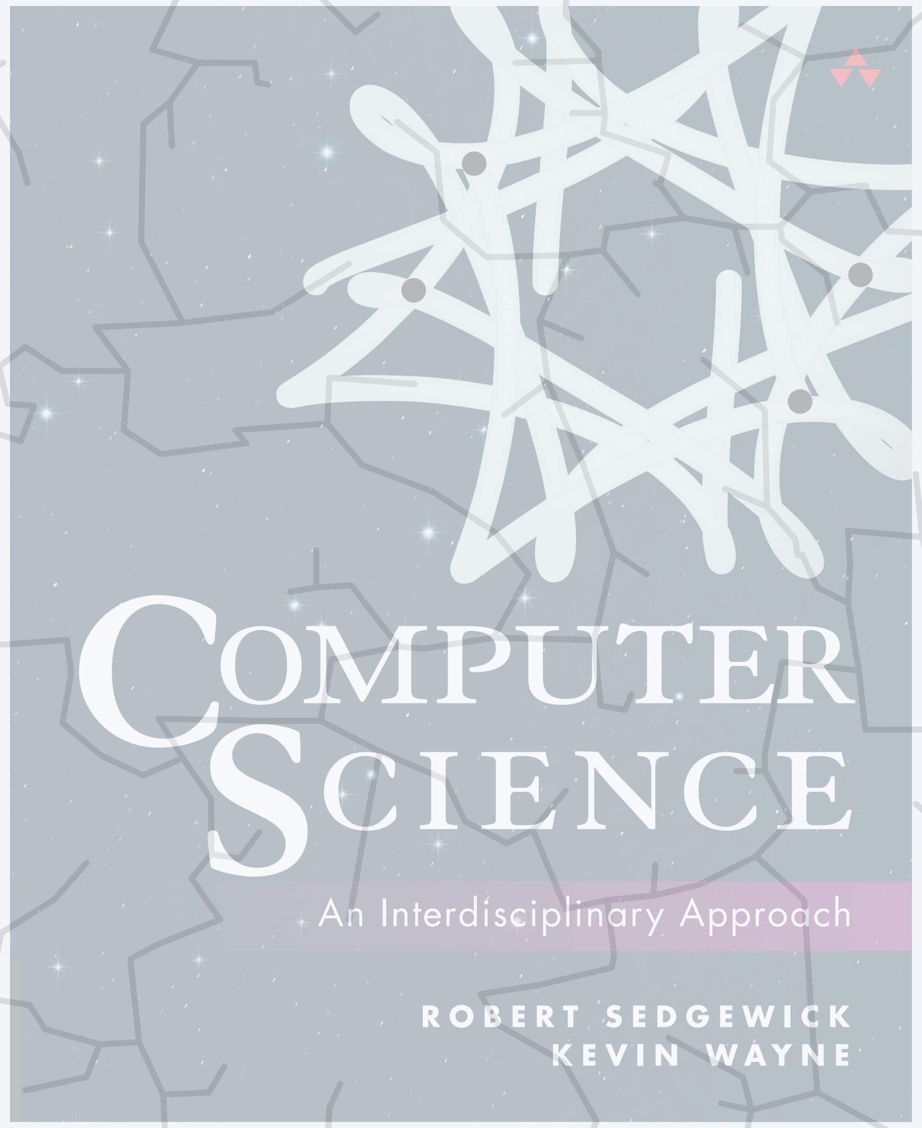
Recursive art (fall 2023)

Every semester, Princeton University's COS 126 invites students to use their newly acquired programming skills to create some amazing pieces of *recursive art*!

Here is what the Fall 2023 class has come up with!

PRINCETON UNIVERSITY

flashing images



https://introcs.cs.princeton.edu

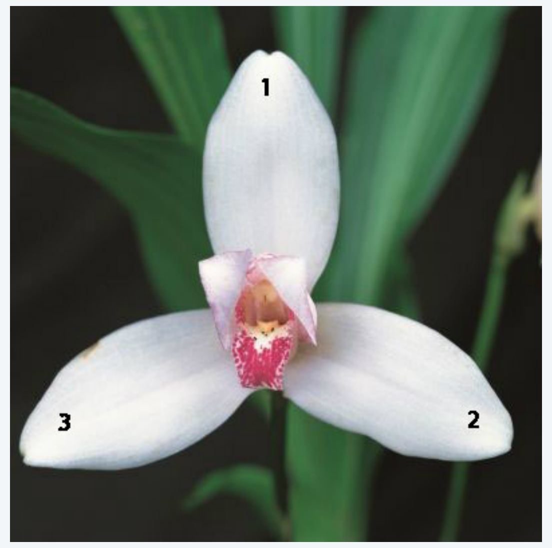
2.3 Recursion

- *foundations*
- *a classic example*
- *recursive graphics*
- *exponential waste*

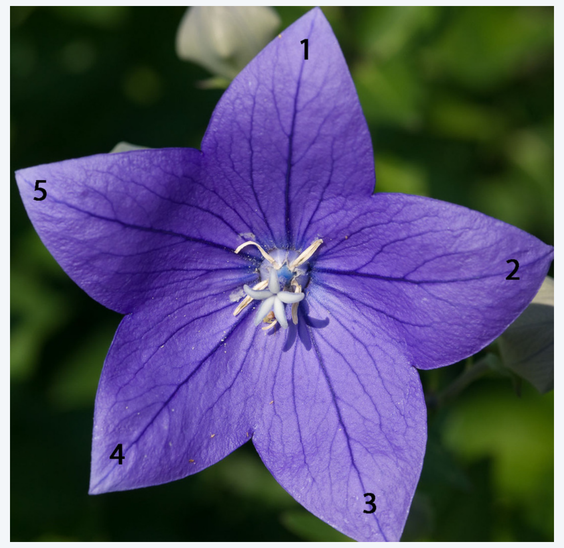
Fibonacci numbers

Fibonacci numbers. $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$

Leonardo Fibonacci



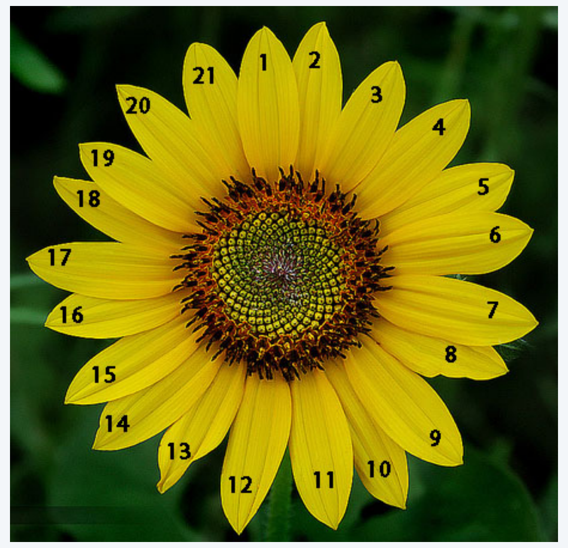
3



5

8

13



21

34

55

89

Fibonacci numbers: recursive approach

Fibonacci numbers. $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$

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

Goal. Given n, compute F_n.

Recursive approach.

- Base cases: $F_0 = 0, \ F_1 = 1$.
- Reduction step: $F_n = F_{n-1} + F_{n-2}$.

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

How long dose it take to compute `fib(80)` ?

A. Much less than 1 second.

B. About 1 second.

C. About 1 minute.

D. About 1 hour.

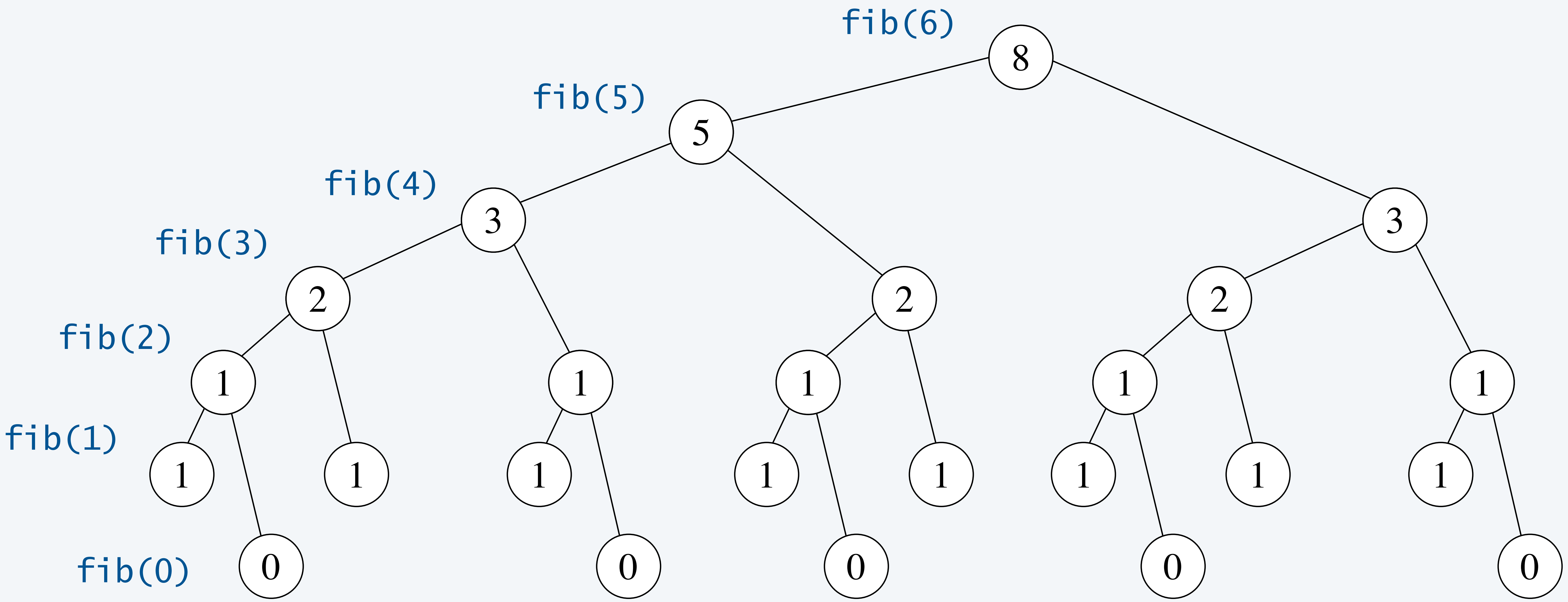
E. More than 1 hour.

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

Recursion tree for Fibonacci numbers

Recursion tree.

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

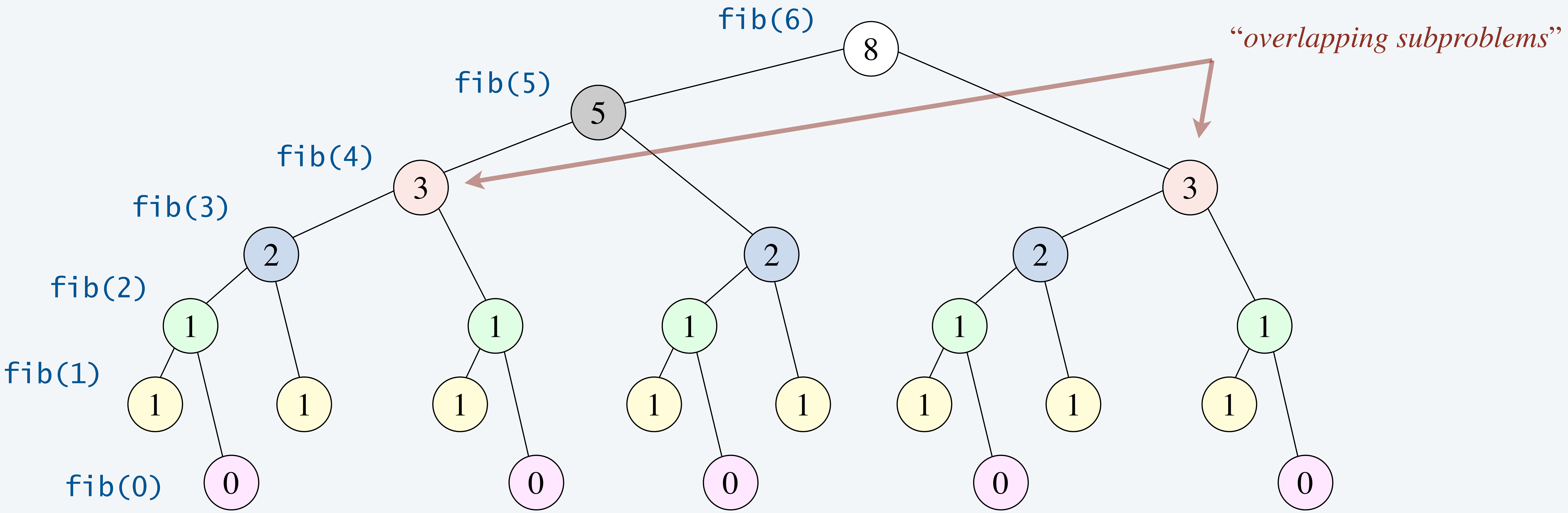


Exponential waste

Exponential waste. Same overlapping subproblems are solved repeatedly.

- fib(5) is called 1 time.
- fib(4) is called 2 times.
- fib(3) is called 3 times.
- fib(2) is called 5 times.
- fib(1) is called 8 times.

number of recursive calls are Fibonacci numbers (and grow exponentially)



Exponential waste dwarfs progress in technology

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

n	recursive calls	VAX-11 (1970s)	MacBook Pro (2020s)
30	2,692,536	*minute*	
40	331,160,280	*hours*	
50	40,730,022,146	*weeks*	*minute*
60	5,009,461,563,920	*years*	*hours*
70	616,123,042,340,256	*centuries*	*weeks*
80	75,778,124,746,287,810	*millenia*	*years*
90	9,320,093,220,751,060,616	⋮	*centuries*
100	1,146,295,688,027,634,168,200		*millenia*
⋮			⋮

↑ *exponential growth* (!)

time to compute `fib(n)` using recursive code

VAX-11/780

Macbook Pro (10,000× faster)

Avoiding exponential waste with memoization

Memoization.

- Maintain an array to remember all computed values.
- If value to compute is known, just return it; otherwise, compute it; remember it; and return it.

Impact. Calls `fibR(i)` at most twice for each `i`.

```
~/cos126/recursion> java-introcs FibonacciMemo 6
8

~/cos126/recursion> java-introcs FibonacciMemo 80
23416728348467685
```

instantaneous (!)

```java
public class FibonacciMemo {
   private static long[] memo;          // "global" variable

   public static long fib(int n) {
      memo = new long[n+1];             // initialize to all 0s (not yet known)
      return fibR(n);
   }

   private static long fibR(int n) {
      if (memo[n] != 0) return memo[n];         // F_n known
      if      (n == 0) memo[n] = 0;             // compute F_n and store in array
      else if (n == 1) memo[n] = 1;
      else memo[n] = fibR(n-1) + fibR(n-2);

      return memo[n];                   // return stored value
   }

   ...
}
```

Design paradigm. This is a simple example of memoization (top-down dynamic programming).

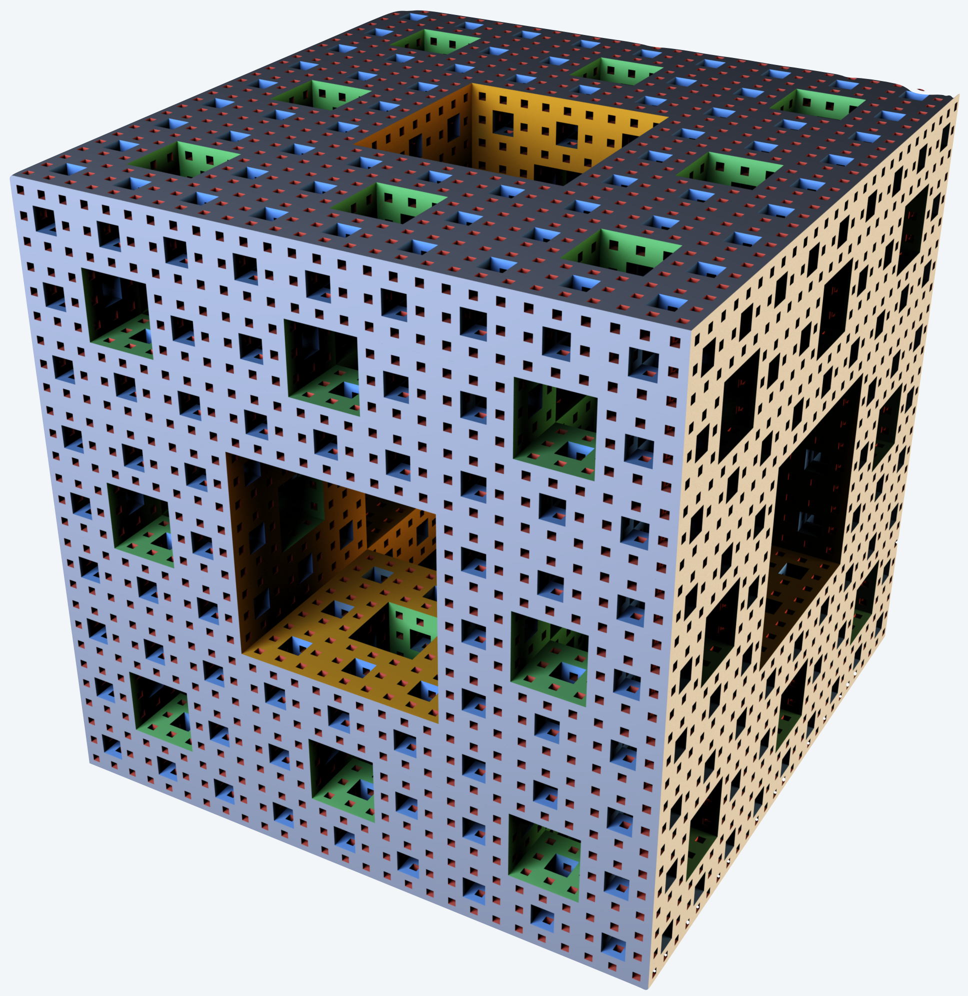
Summary

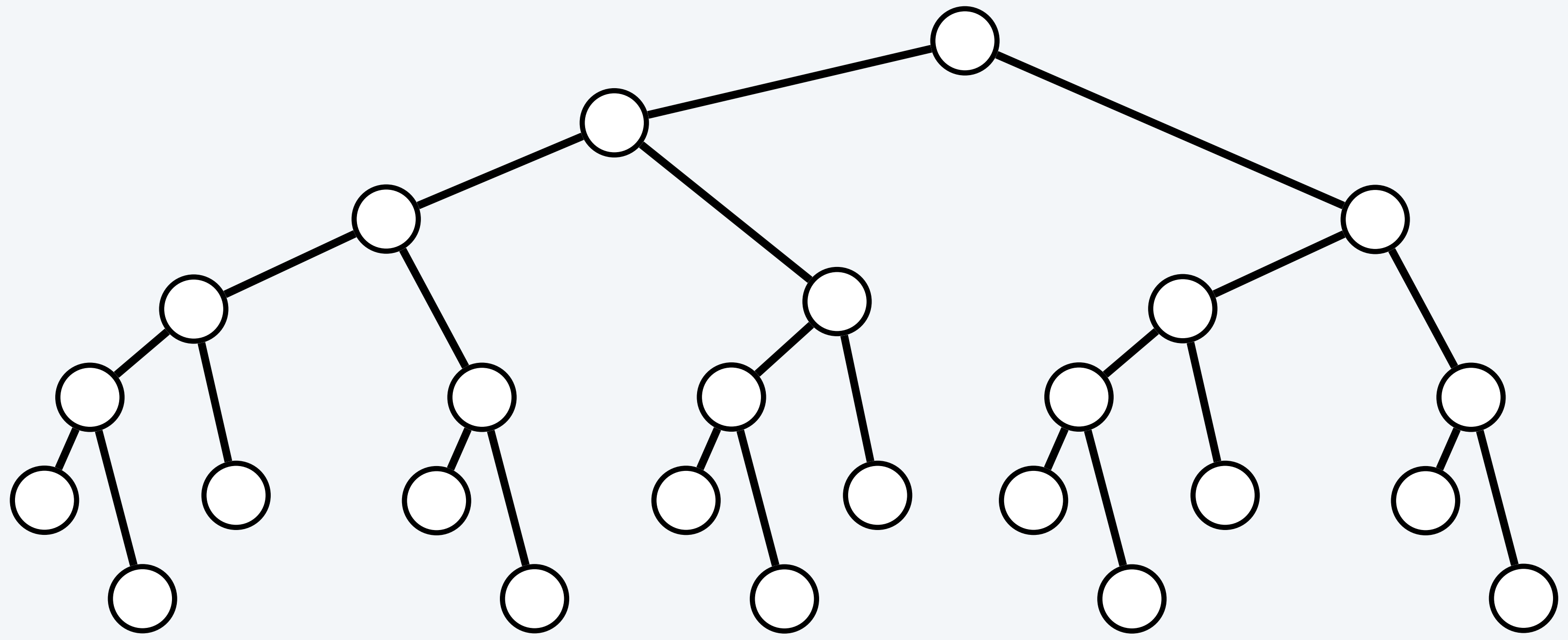
Recursive function. A function that calls itself.

Why learn recursion?

- Represents a new mode of thinking.
- Provides a powerful programming paradigm.
- Reveals insight into the nature of computation.

Dynamic programming. A powerful technique to avoid exponential waste. ← *see also* COS 226





```
// Ackermann function
public static long ack(long m, long n) {
   if (m == 0) return n+1;
   if (n == 0) return ack(m-1, 1);
   return ack(m-1, ack(m, n-1));
}
```

challenge for bored: compute ack(5, 2)

Credits

media	source	license
Painting Hands	Adobe Stock	education license
Bugs	Adobe Stock	education license
Stack Overflow Logo	Stack Overflow	
Problems with Recursion	Zach Weinersmith	
You're Eating Recursion	Safely Endangered	
Collatz Game	Quanta magazine	
File System with Folders	Adobe Stock	education license
Wooden Towers of Hanoi	Adobe Stock	education license
Towers of Hanoi Visualization	Imaginative Animations	

Credits

media	source	license
Droste Cocoa	Droste	
Recursive Giraffe	Farley Katz	
Circle Limit IV	M.C. Escher	
Recursive Mona Lisa	Mr. Rallentando	
Recursive New York Times	Serkan Ozkaya	
Leonardo Fibonacci	Wikimedia	public domain
VAX 11/780	Digital Equipment Corporation	
Macbook Pro M1	Apple	
Menger Sponge	Niabot	CC BY 3.0