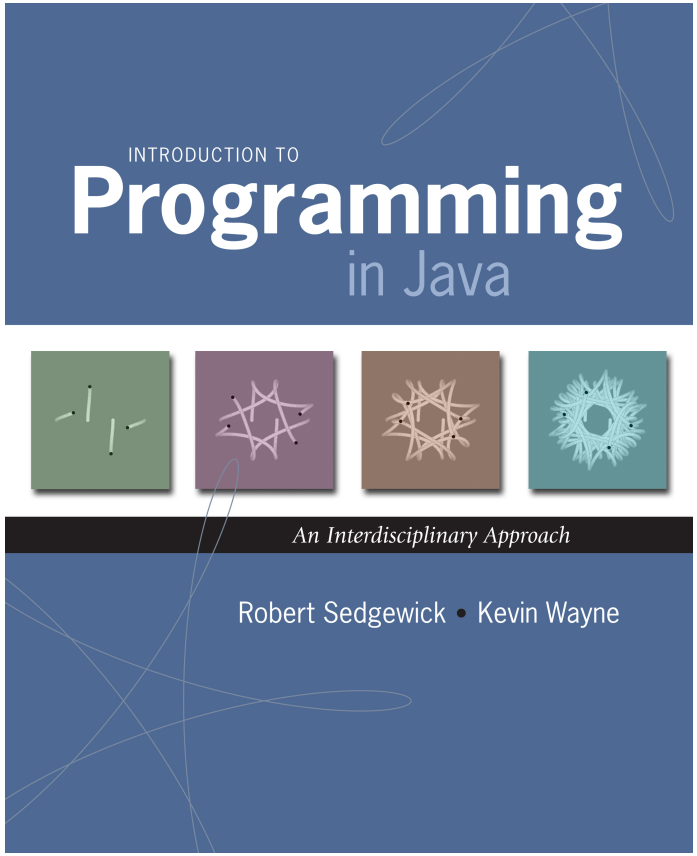
2.3 Recursion



Factorial

The factorial of a positive integer N is computed as the product of N with all positive integers less than or equal to N.

$4! = 4 \times 3 \times 2 \times 1 = 24$

$30! = 30 \times 29 \times \ldots \times 2 \times 1 =$
$265252859812191058636308480000000$

Factorial - Iterative Implementation

```
1.   int b = factorial(5);


2.   static int factorial(int n) {
3.     int f = 1;
4.
5.     for (int i=n; i>=1; i--) {
6.       f = f * i;
7.     }
8.
9.     return f;
10.  }
```

Trace it.

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$4! = 4 \times 3 \times 2 \times 1$

$5! = 5 \times 4!$

$N! = N \times (N-1)!$

Factorial can be defined in terms of itself

Recursion

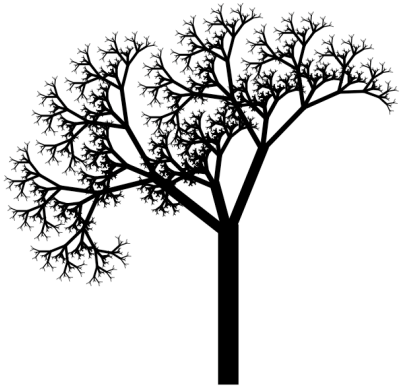
Penn Engineering

Recursion

Penn Engineering

Overview

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

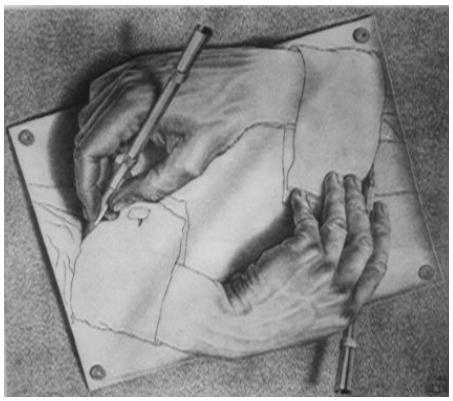


Why learn recursion?

- New mode of thinking
- Powerful programming paradigm

Many computations are naturally self-referential:

- Mergesort, FFT, gcd, depth-first search
- Linked data structures
- A folder contains files and other folders



Reproductive Parts
M. C. Escher, 1948

Closely related to mathematical induction

Penn Engineering

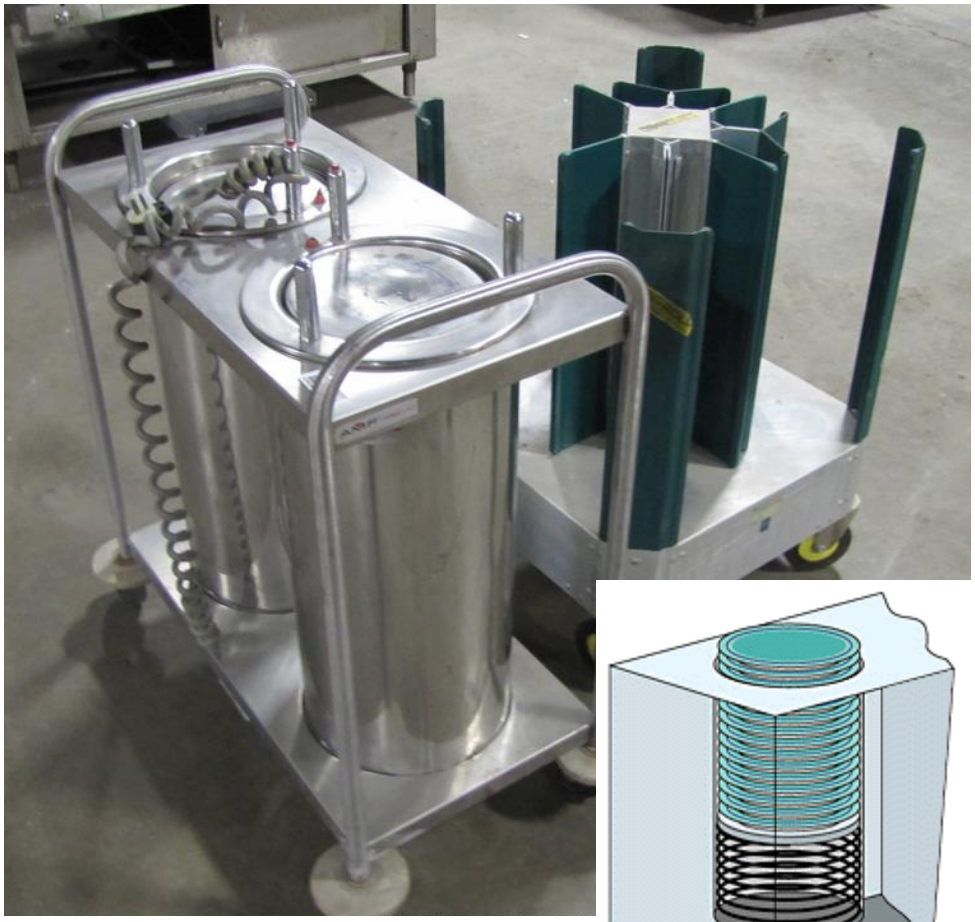
Factorial – Recursive Implementation

```
1.   int b = factorial(5);


2.   static int factorial(int n) {
3.     if (n == 1) {
4.       return 1;
5.     } else {
6.       int f = n * factorial(n-1);
7.       return f;
8.     }
9.   }
```

Trace it.

Penn Engineering

Last In First Out (LIFO) Stack of Plates



Penn Engineering

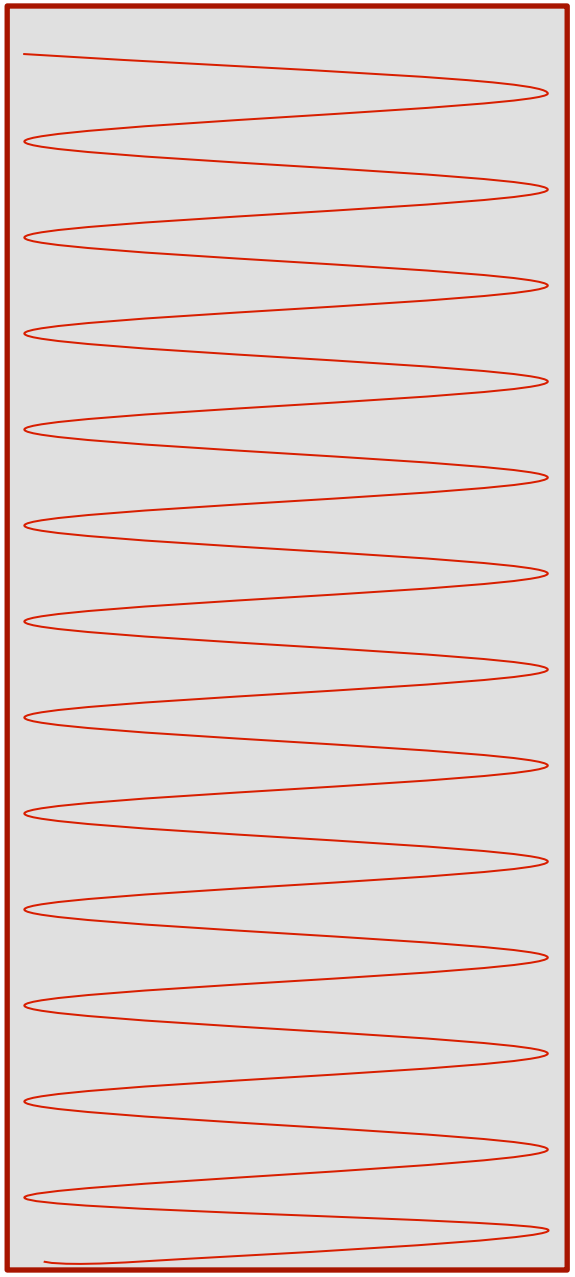
Compiled Code

```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

Call Stack



Penn Engineering

Compiled Code

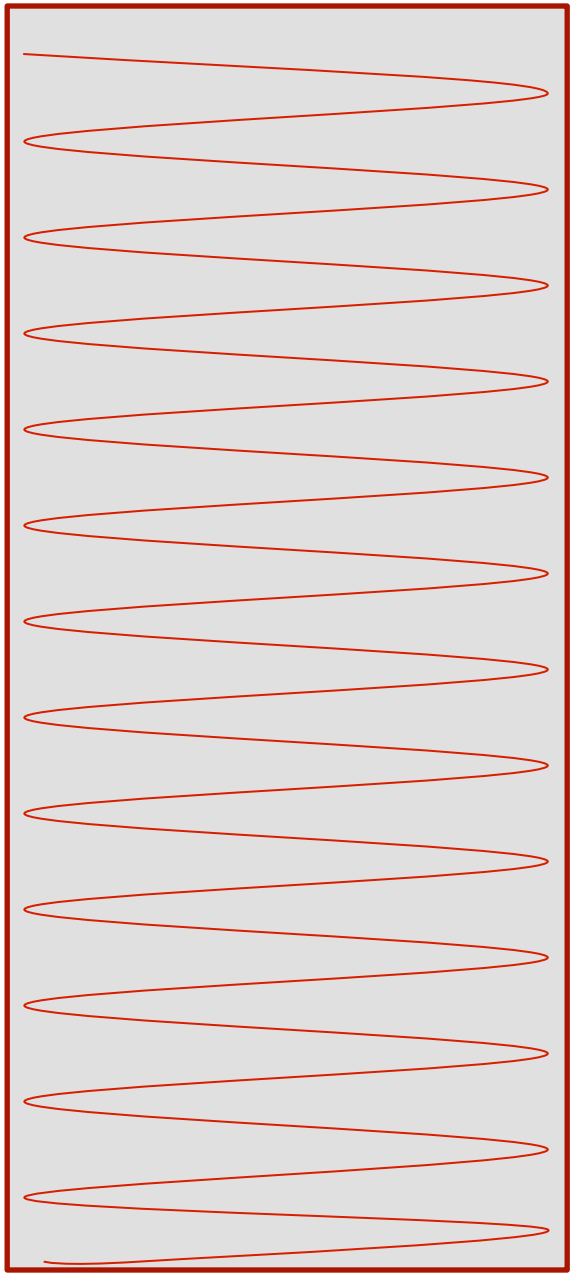
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   void main(String[] args){
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

Call Stack



Penn Engineering

Compiled Code

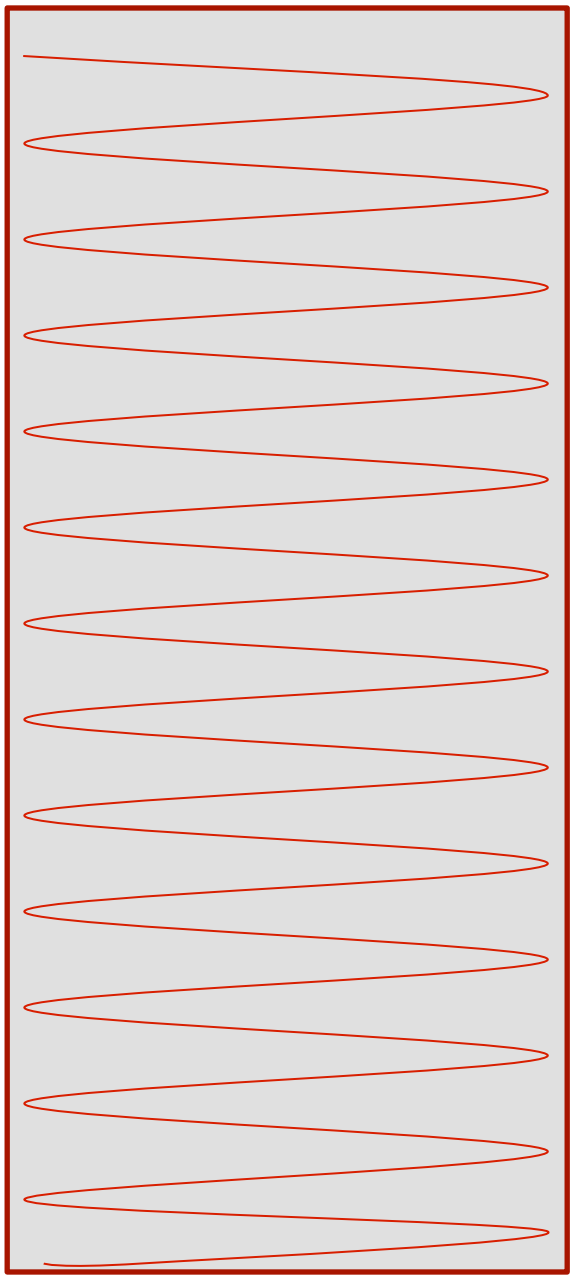
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   void main(String[] args){
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

Call Stack



Penn Engineering

Compiled Code

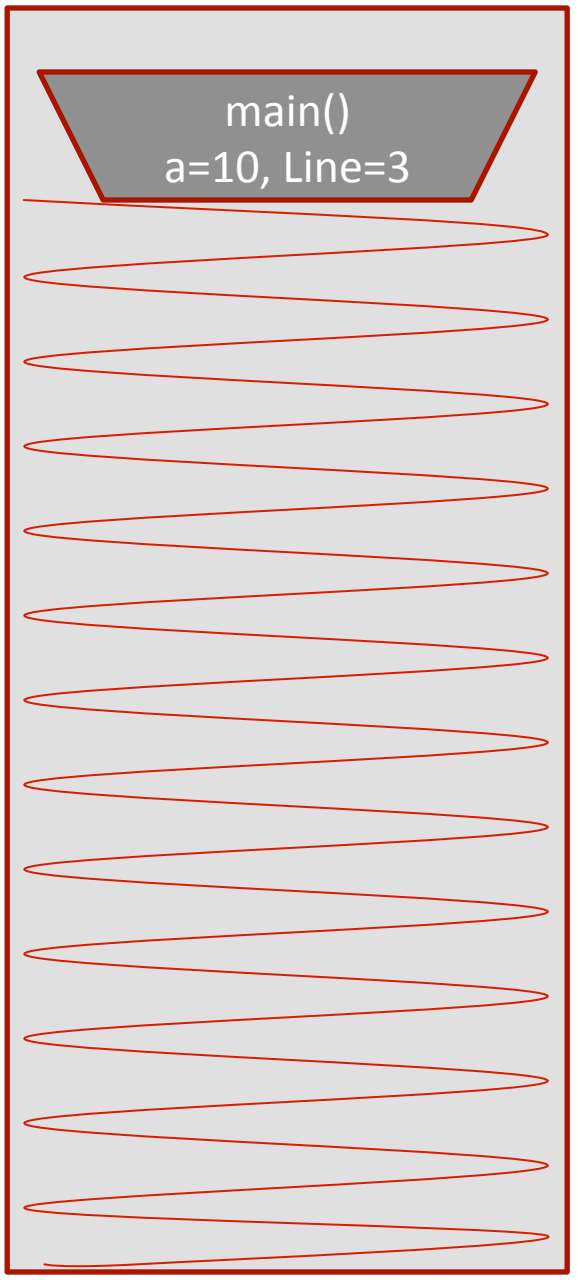
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   void main(String[] args){
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

Call Stack



Penn Engineering

Compiled Code

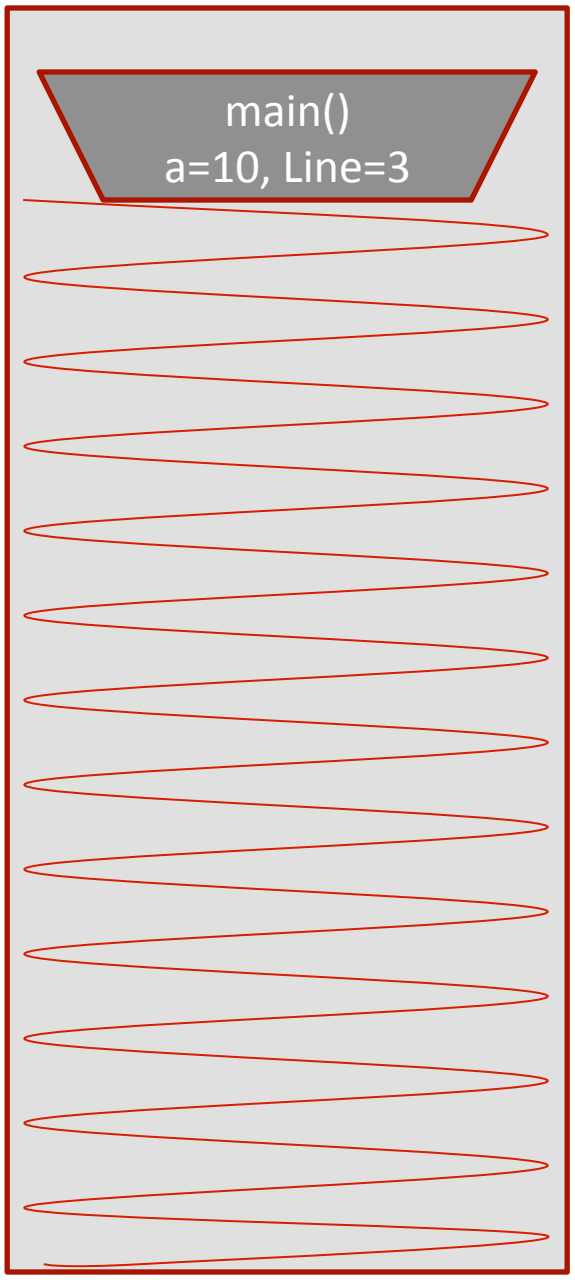
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=5) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

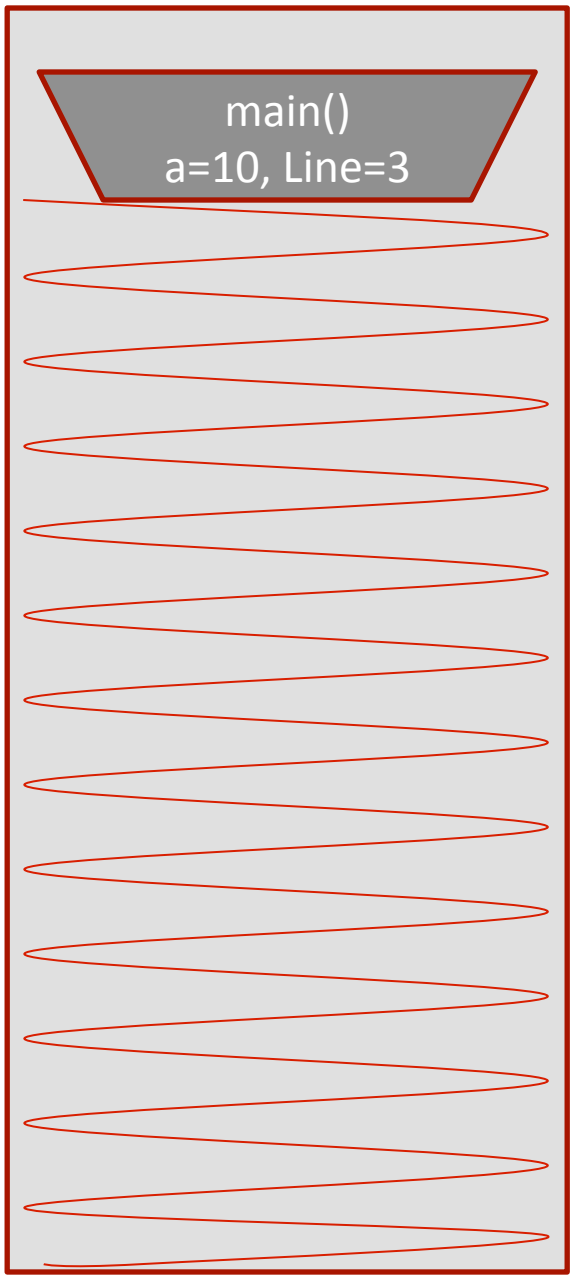
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=5) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

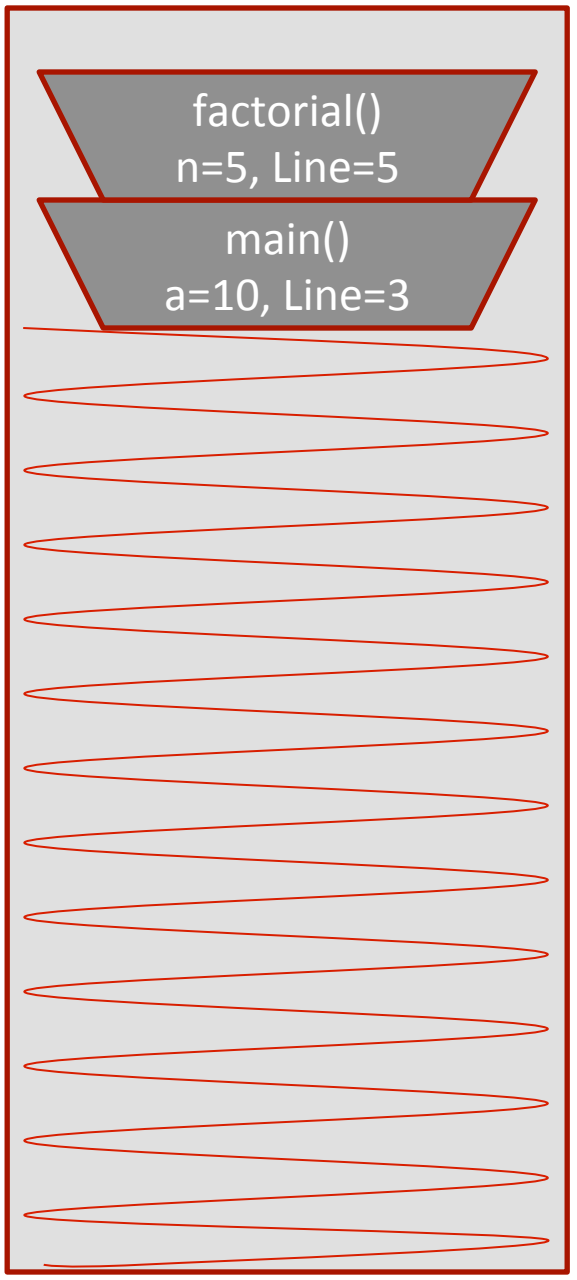
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=5) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

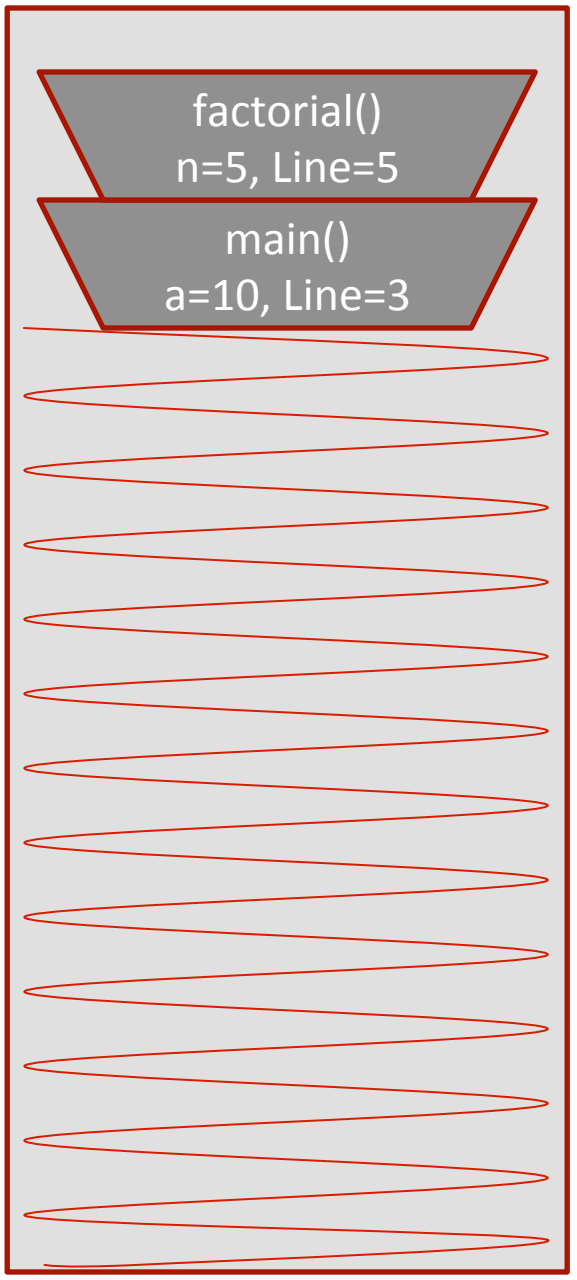
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=4) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

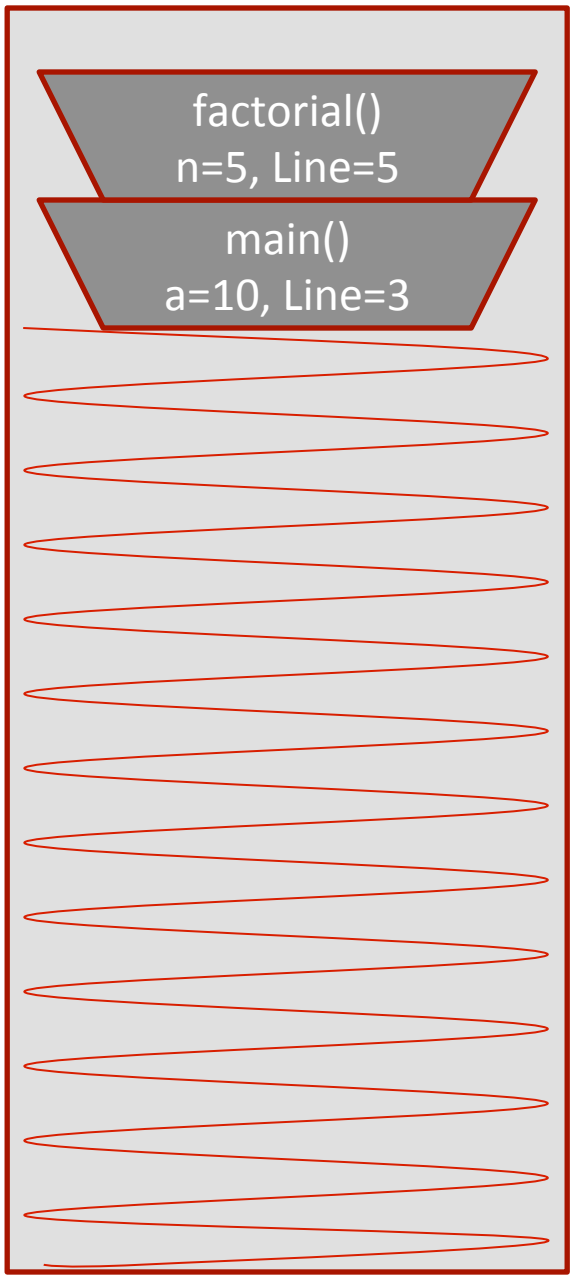
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=4) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

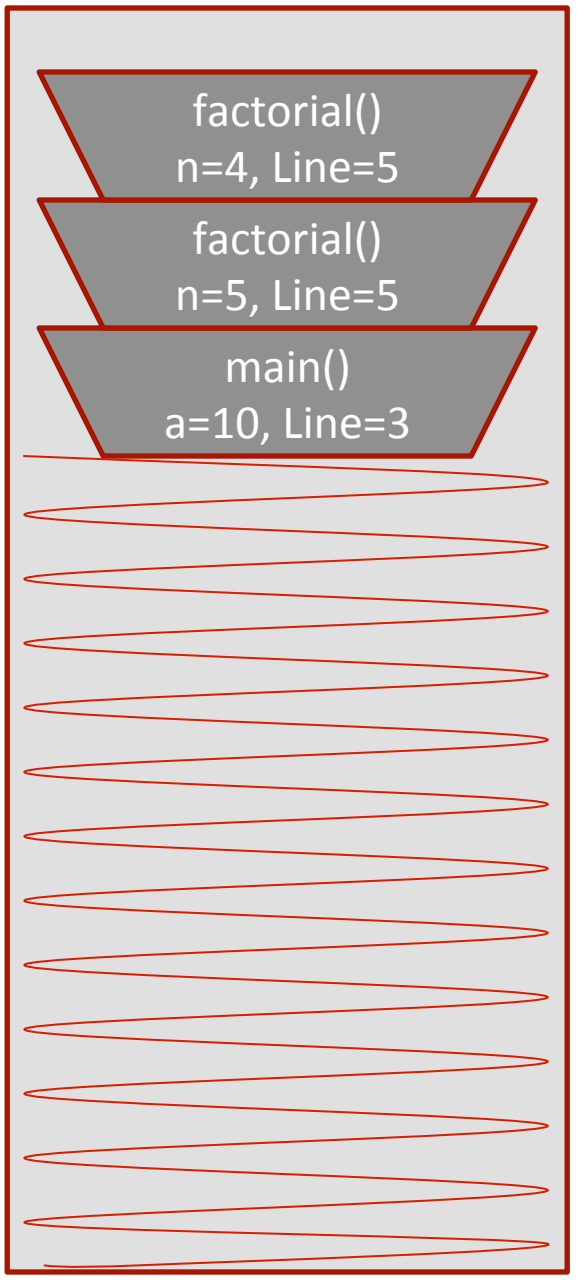
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=4) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

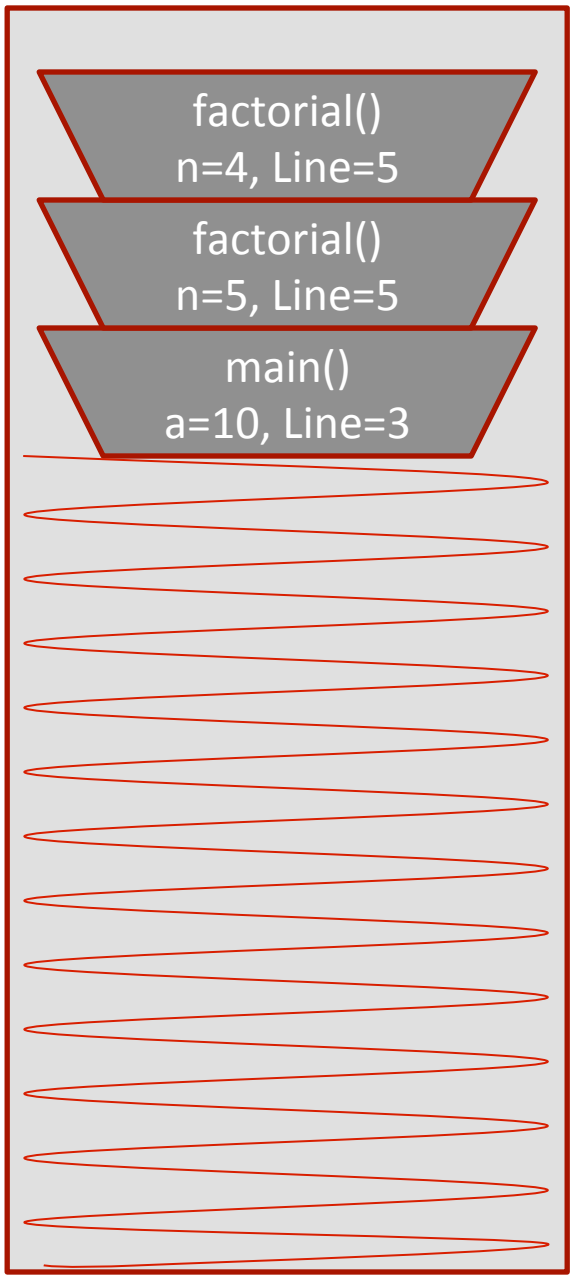
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=3) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

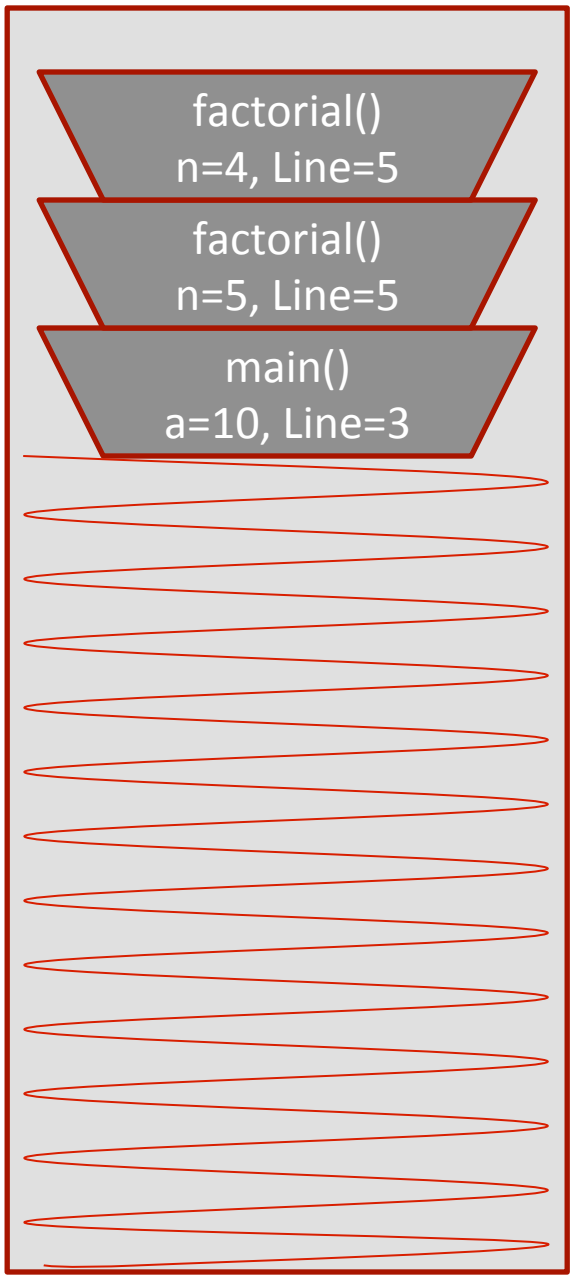
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=3) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

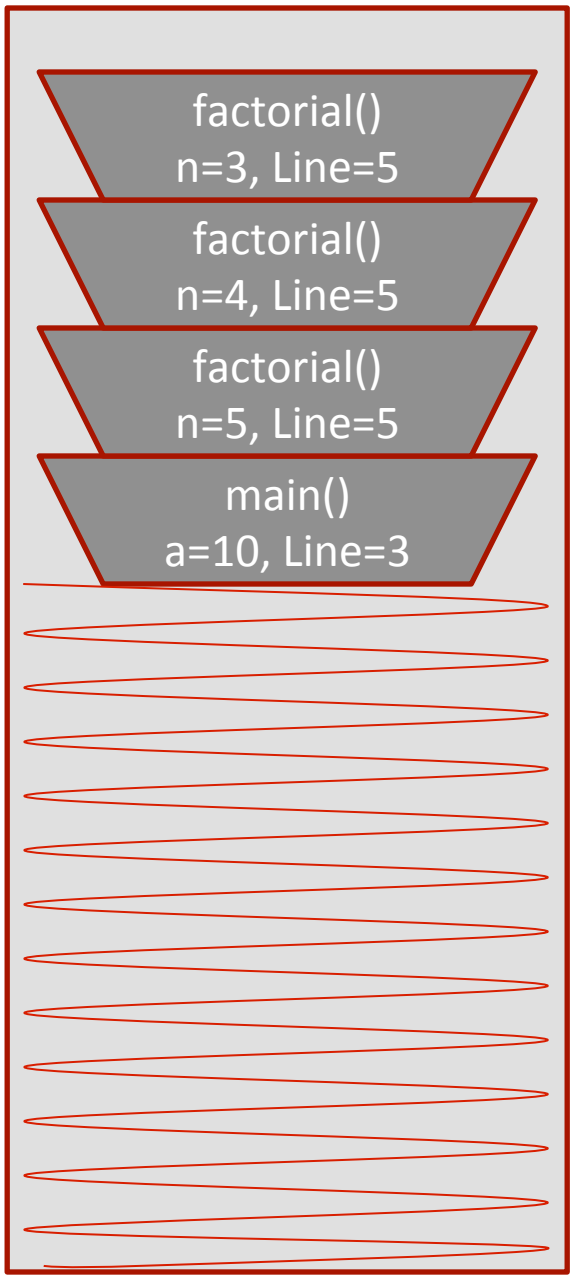
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=3) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

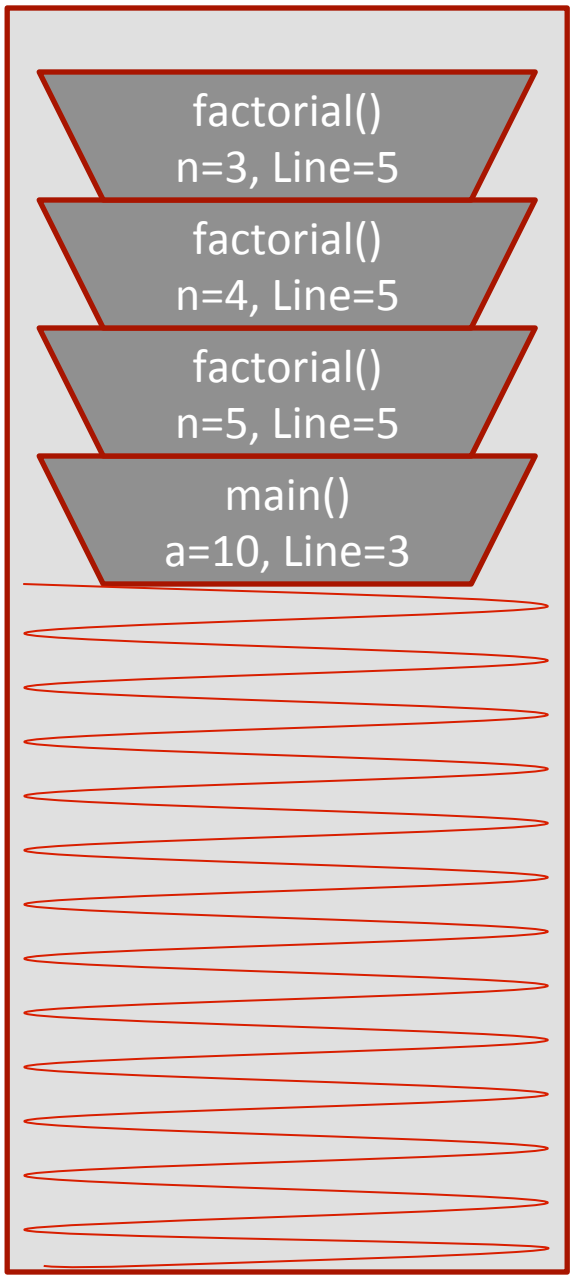
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=2) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

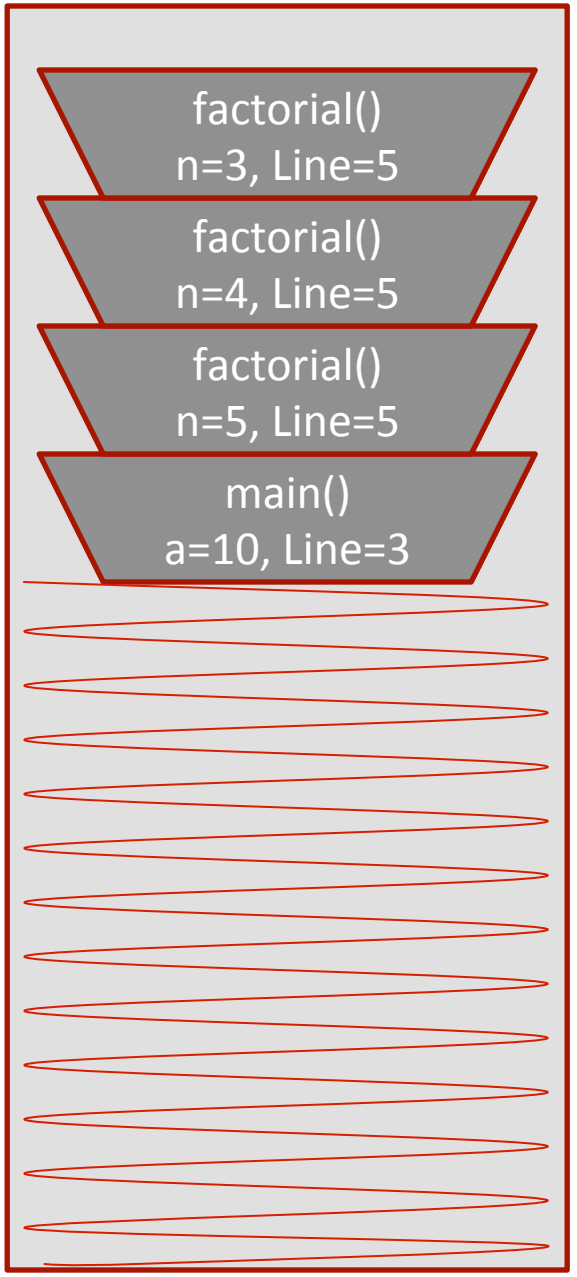
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=2) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

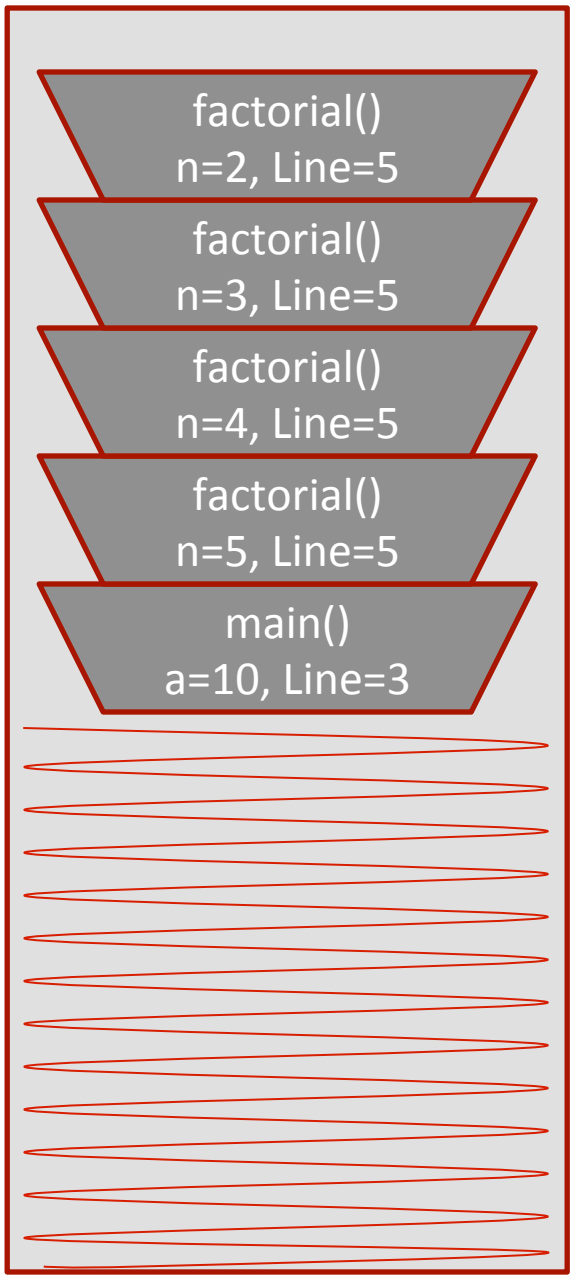
```
1.  public static void main
    (String[] args) {
2.    int a = 10;
3.    int b = factorial(5);
4.    System.out.println(b);
5.  }
```

```
1.  static int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int f = n *
              factorial(n-1);
6.      return f;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int f = n *
            factorial(n-1);
6.      return f;
7.    }
8.  }
```

Call Stack



Penn Engineering

Compiled Code

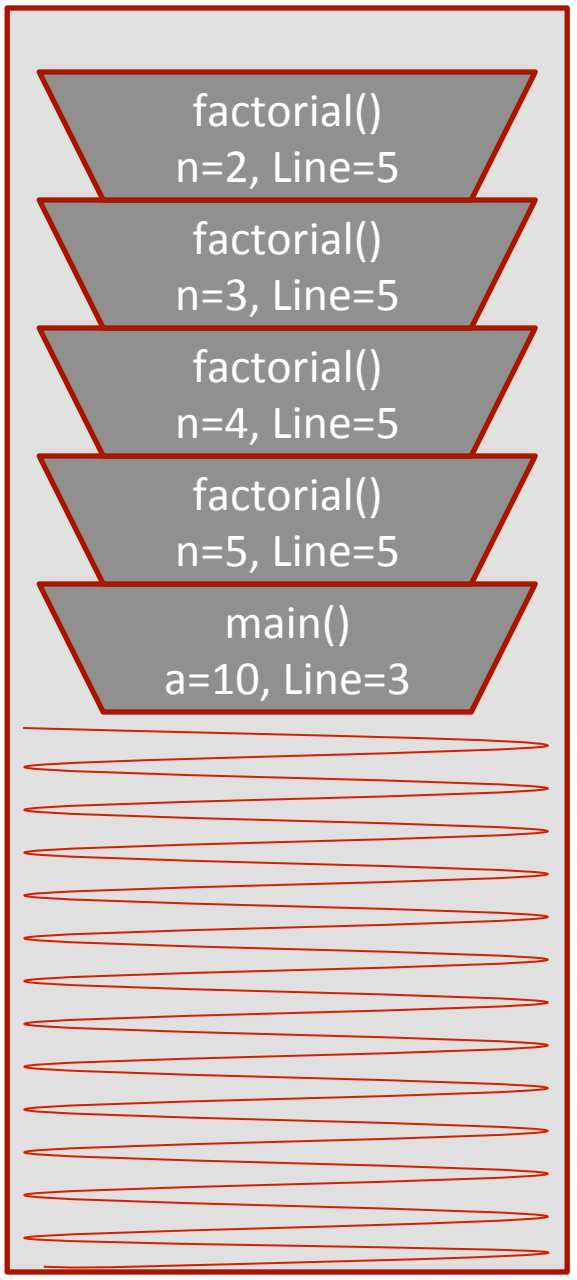
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=1) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
             factorial(n-1);
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

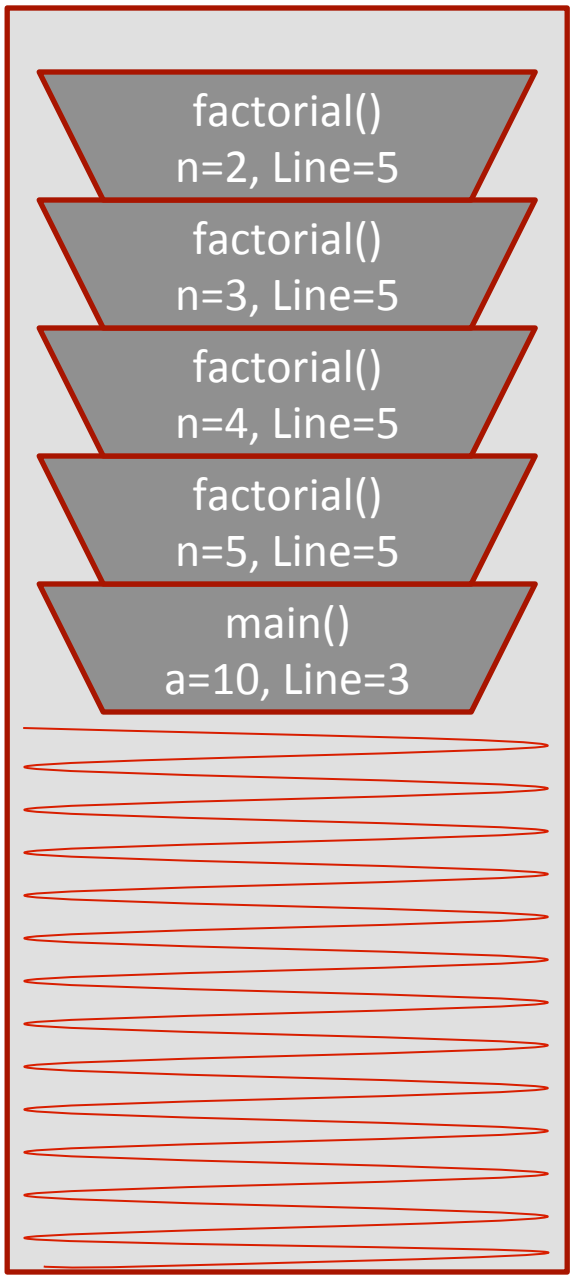
```
1.  public static void main
    (String[] args) {
2.    int a = 10;
3.    int b = factorial(5);
4.    System.out.println(b);
5.  }
```

```
1.  static int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int f = n *
            factorial(n-1);
6.      return f;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=1) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int f = n *
            factorial(n-1);
6.      return f;
7.    }
8.  }
```

Call Stack



Penn Engineering

Compiled Code

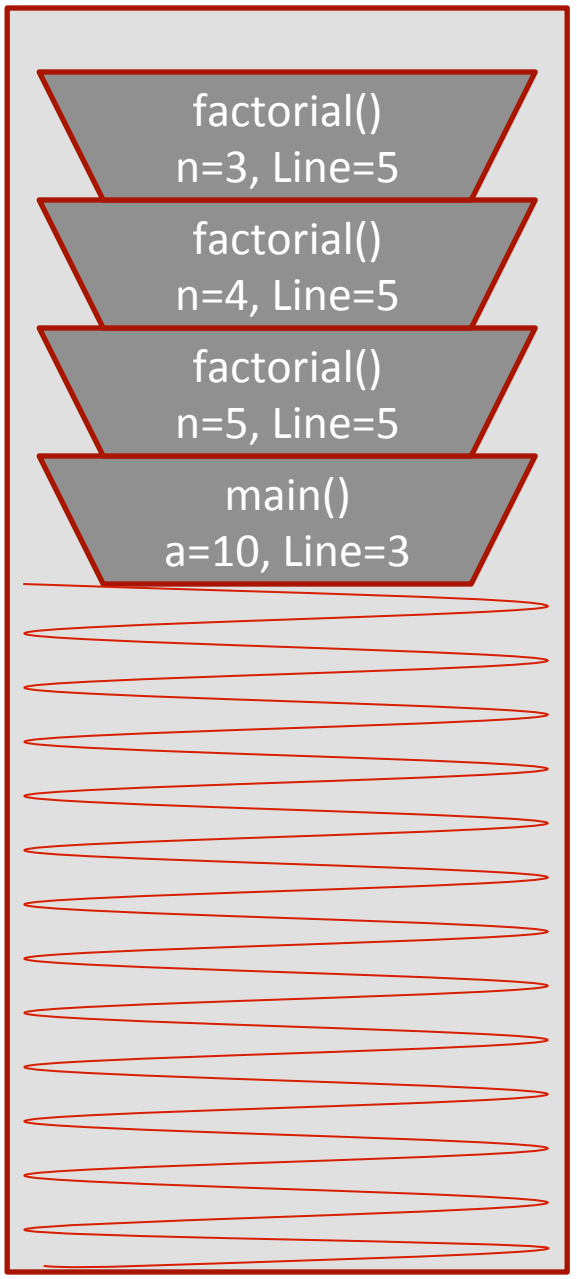
```
1.  public static void main
    (String[] args) {
2.    int a = 10;
3.    int b = factorial(5);
4.    System.out.println(b);
5.  }
```

```
1.  static int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int f = n *
            factorial(n-1);
6.      return f;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int f = n * 1;
6.      return f;
7.    }
8.  }
```

Call Stack



Penn Engineering

Compiled Code

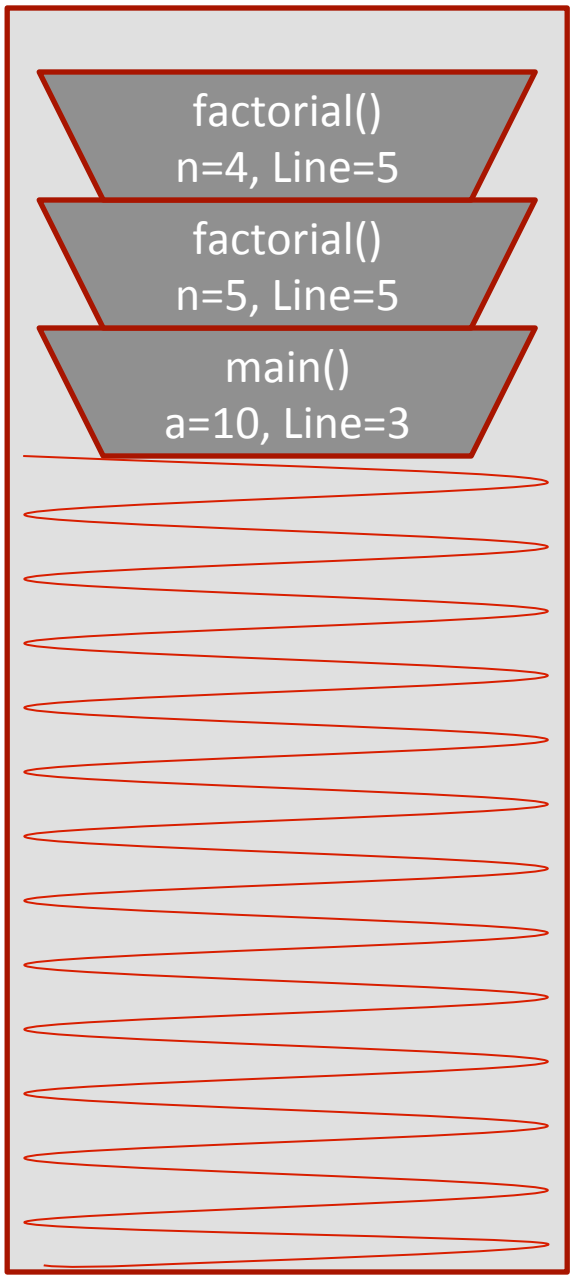
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=3) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n * 2;
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

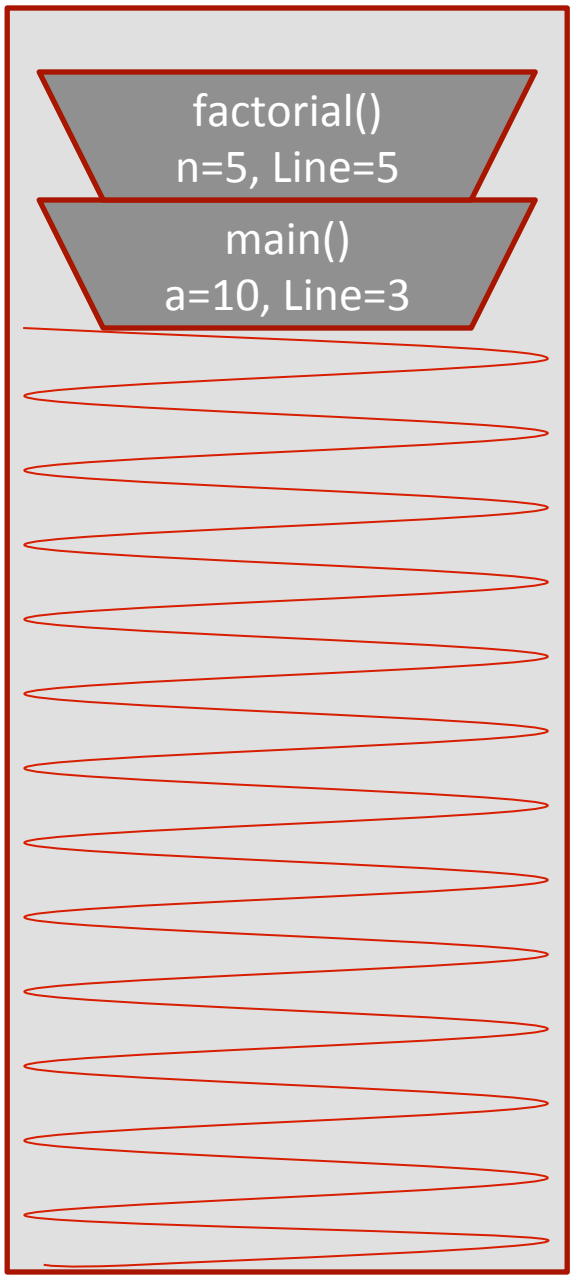
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=4) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n * 6;
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

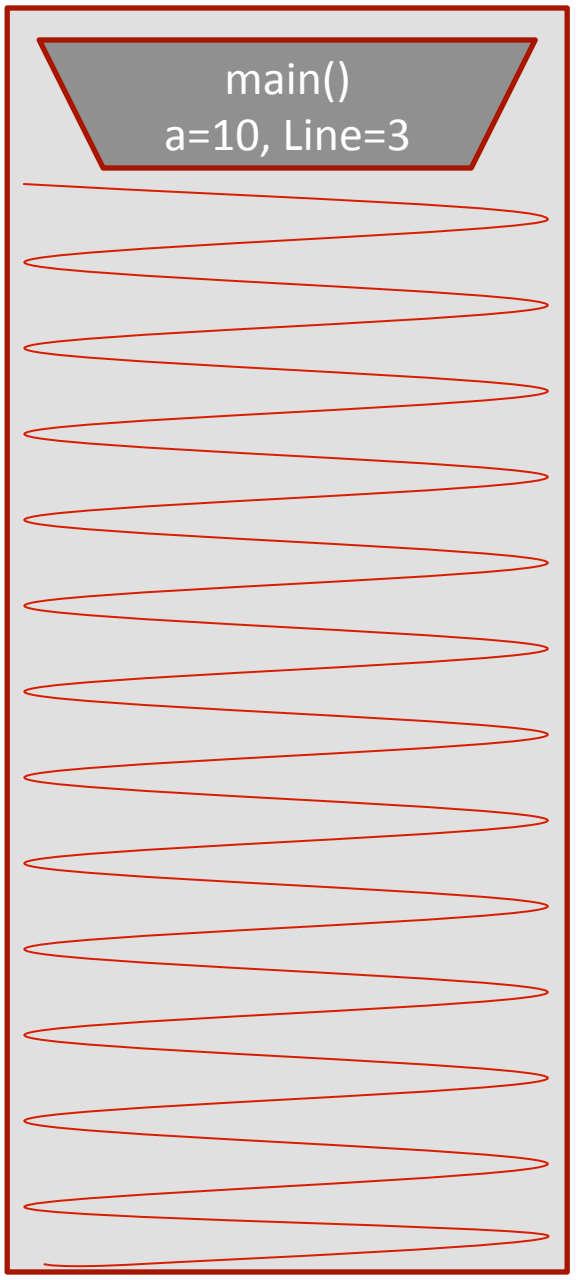
```
1.   public static void main
     (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5.   }
```

```
1.   static int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n *
               factorial(n-1);
6.       return f;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=5) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int f = n * 24;
6.       return f;
7.     }
8.   }
```

Call Stack



Penn Engineering

Compiled Code

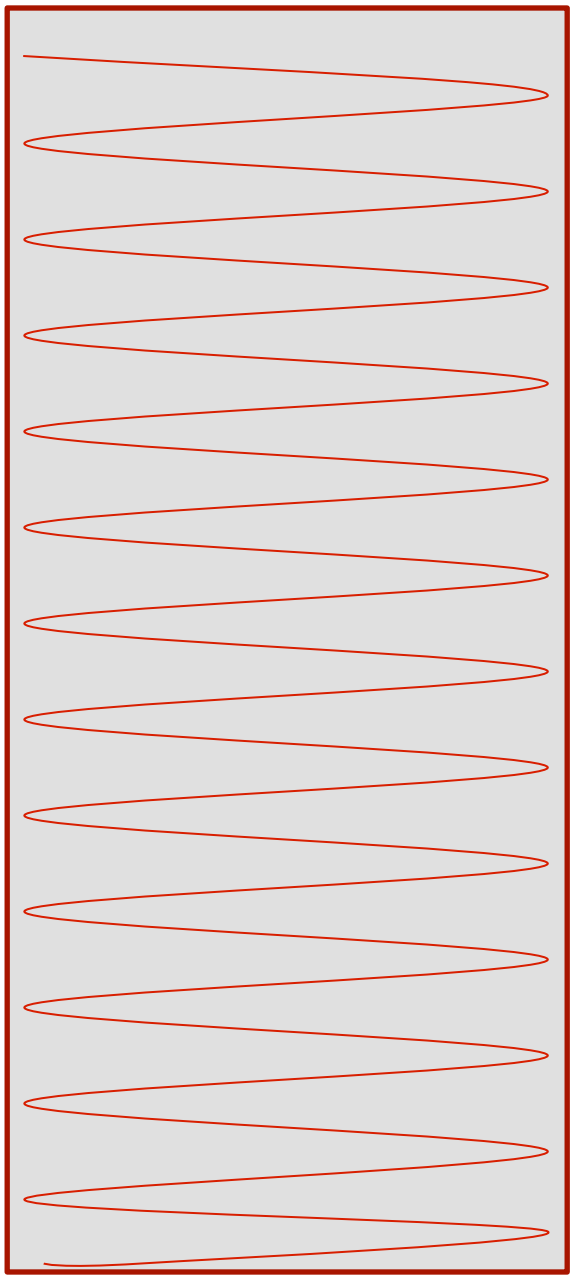
```
1.  public static void main
    (String[] args) {
2.    int a = 10;
3.    int b = factorial(5);
4.    System.out.println(b);
5.  }
```

```
1.  static int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int f = n *
            factorial(n-1);
6.      return f;
7.    }
8.  }
```

Executing Function

```
1.  void main(String[] args){
2.    int a = 10;
3.    int b = factorial(5);
      System.out.println(b);
5.  }
```

Call Stack

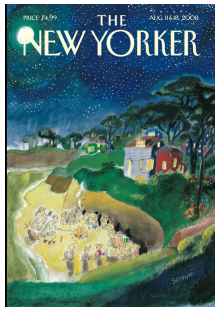


Penn Engineering

The Call Stack keeps track of …

1. all functions that are suspended, in order
2. the point in the function where execution should resume after the invoked subordinate function returns
3. a snapshot of all variables and values within the scope of the suspended function so these can be restored upon continuing execution

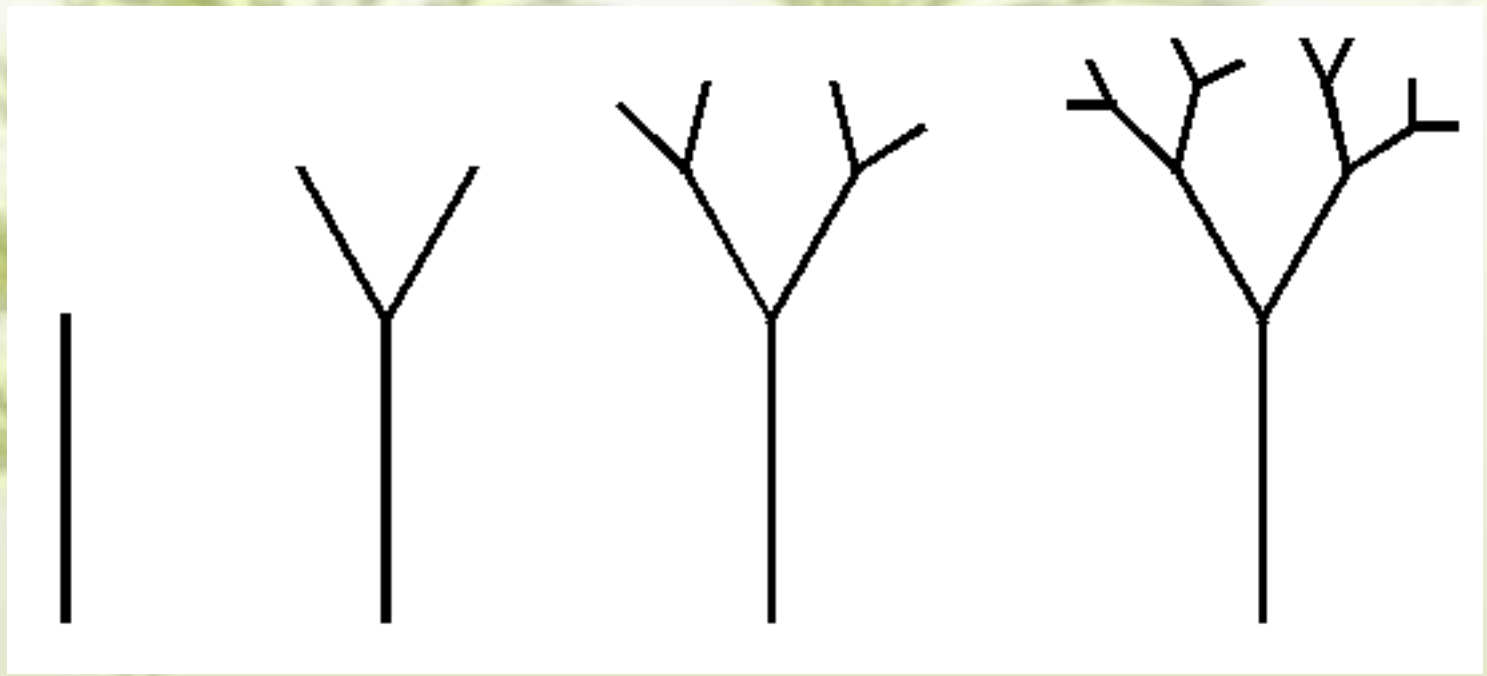
Recursive Graphics



Penn Engineering

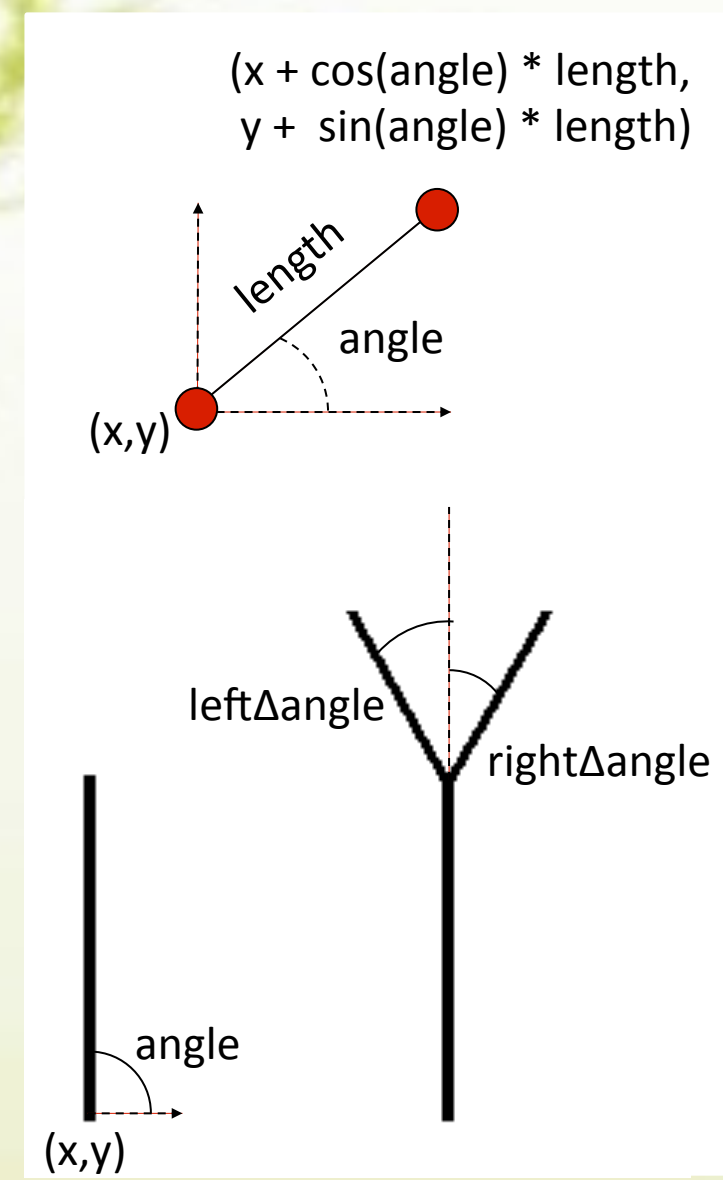
L-systems

L-systems



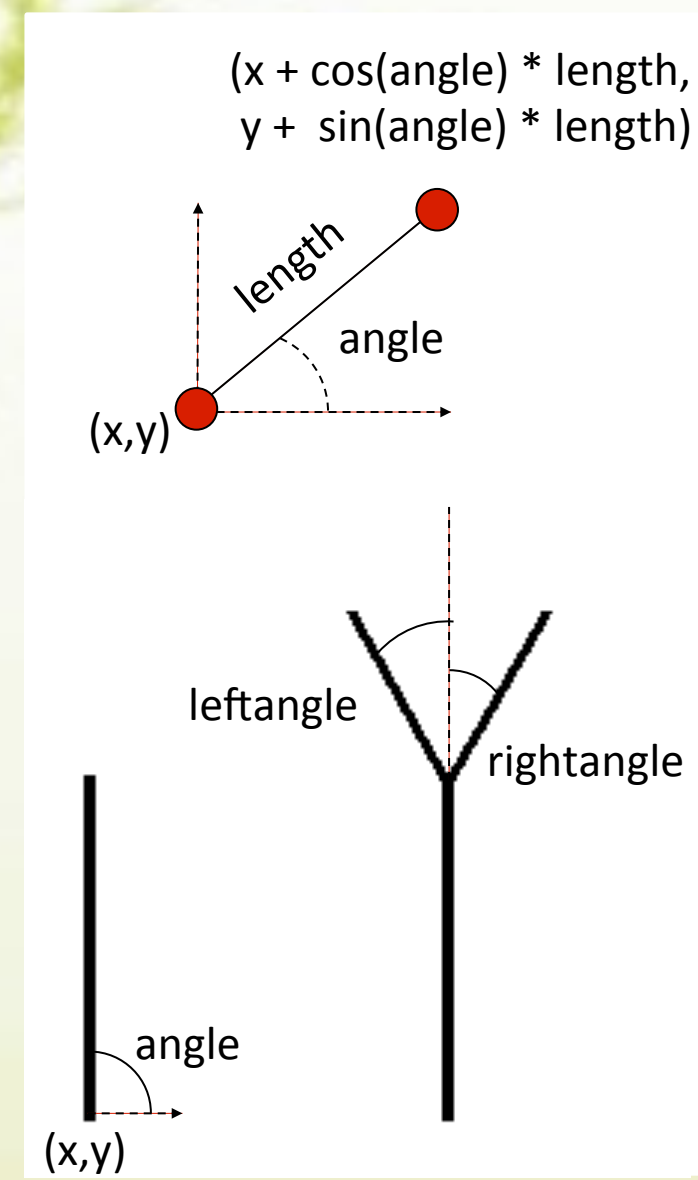
L-systems

```
void drawTree(int x, int y, double angle, int depth) {
   // base case

   // compute end coordinate (x2, y2)
   //     (with length = depth )

   // draw tree recursively


}
```



L-systems

```
void drawTree(int x, int y, double angle, int depth) {
   // base case

   // compute end coordinate (x2, y2)
   //     (with length = depth )

   // draw tree recursively
   StdDraw.line (x, y, x2, y2)
   drawTree(x2, y2, angle  + leftangle, depth –  0.01);
   drawTree(x2, y2, angle  - rightangle, depth – 0.01);
}
```



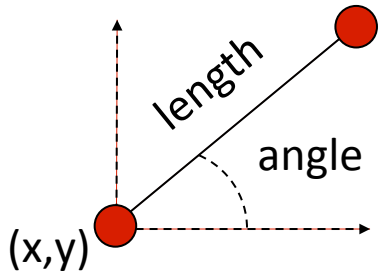
At each stage we want two ‘sub-trees’ to be created. One grows to the left and one grows to the right . Also, the depth controls the length of the branch

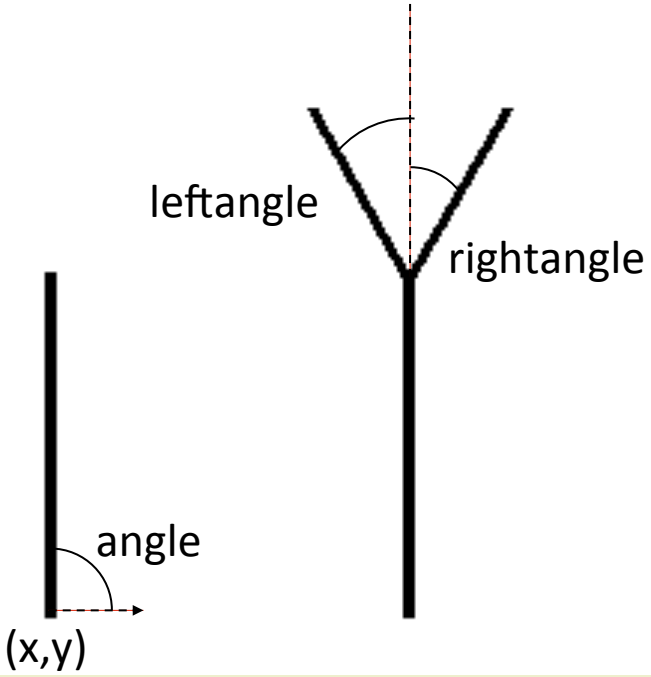
L-systems

```
public static void drawTree(int x, int y, double angle, int depth) {
   // base case
   if (depth == 0) return;

   // compute end coordinate
   double angleRadians = Math.toRadians(angle);
   double x2 = x + (Math.cos(angleRadians) *
                    depth);
   double y2 = y +  (Math.sin(angleRadians) *
                    depth);

   // draw tree recursively
   StdDraw.line(x, y, x2, y2);
   drawTree(x2, y2, angle - leftangle, depth - 1);
   drawTree(x2, y2, angle + rightangle, depth - 1);
}
```

For the complete program refer to the cis110 website

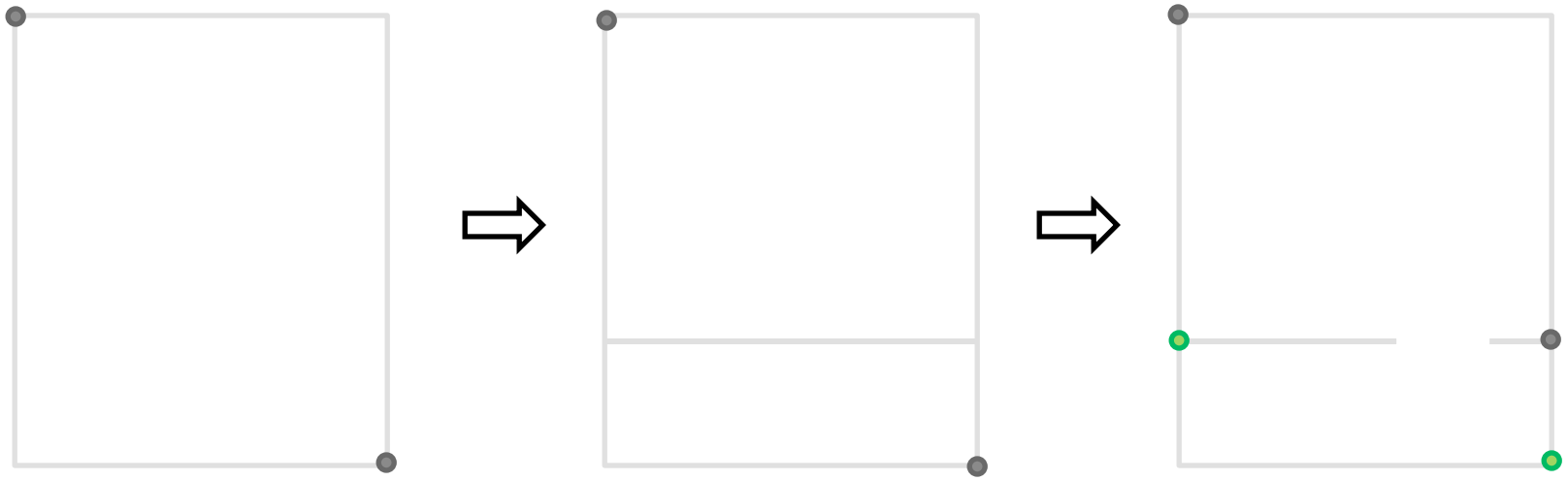
(x + cos(angle) * length,
y + sin(angle) * length)





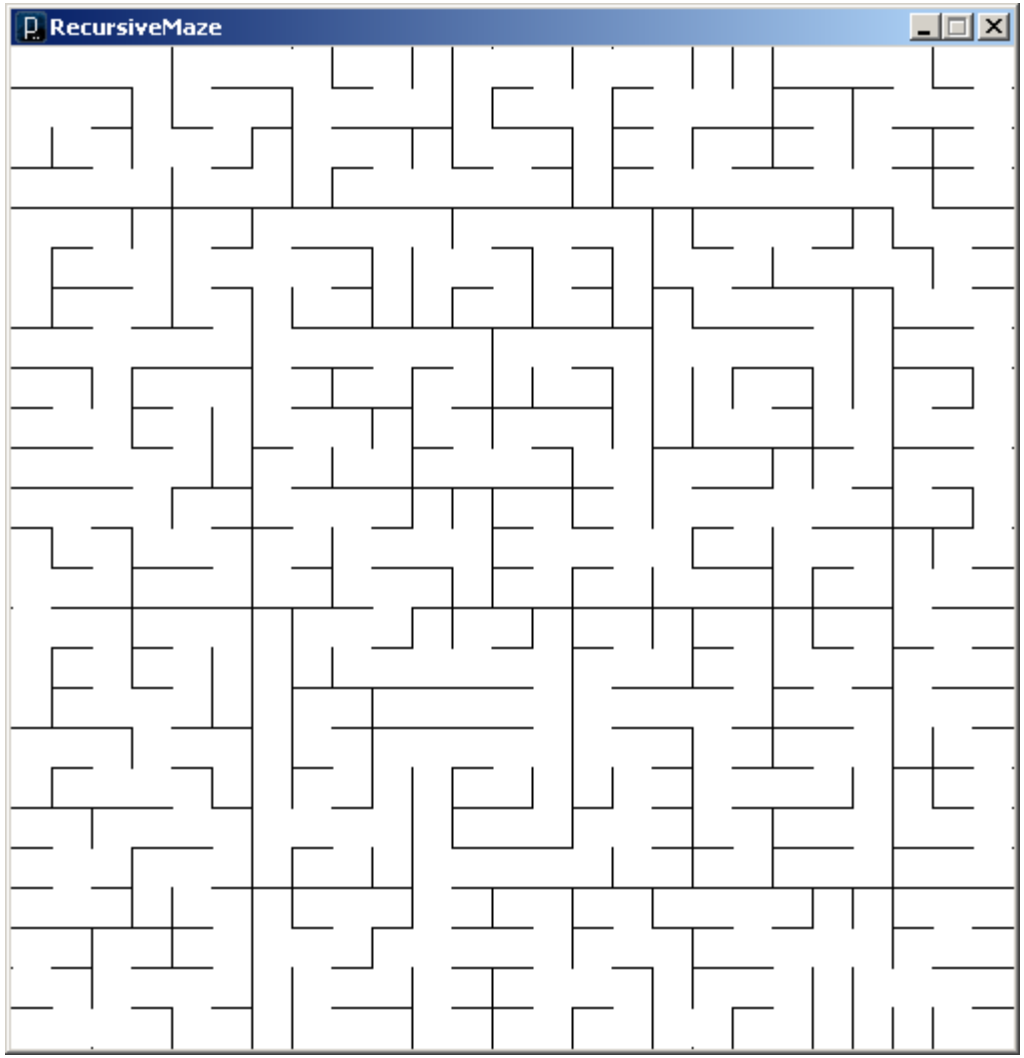
Creating a maze, recursively

1. Start with a rectangular region defined by its upper left and lower right corners
2. Divide the region at a random location through its more narrow dimension
3. Add an opening at a random location
4. Repeat on two rectangular subregions



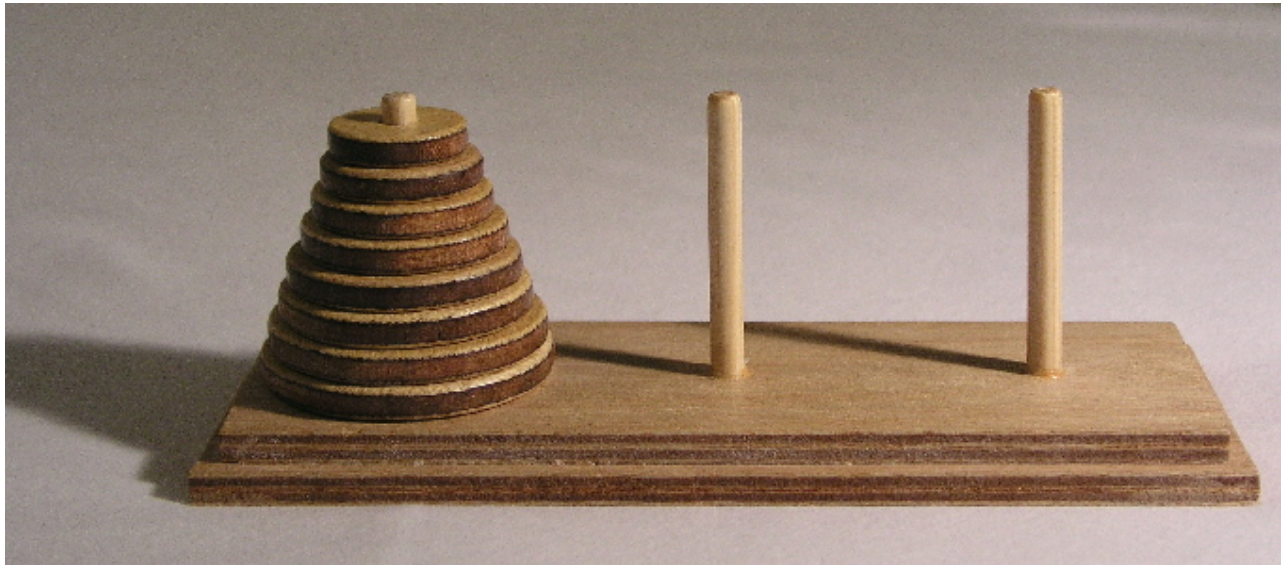
Penn Engineering

Inspired by http://weblog.jamisbuck.org/2011/1/12/maze-generation-recursive-division-algorithm



Penn Engineering

Towers of Hanoi



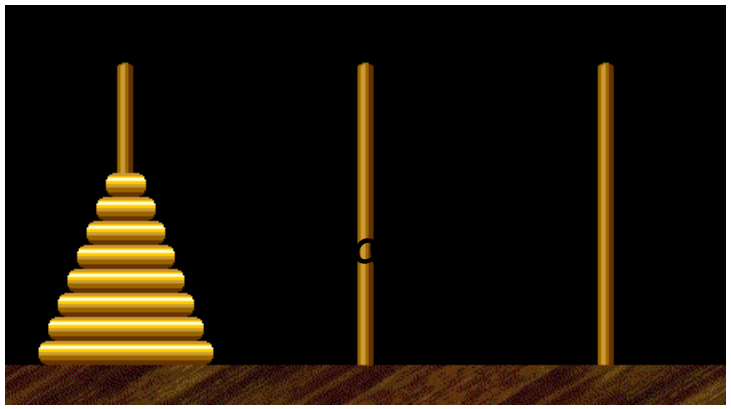
http://en.wikipedia.org/wiki/Image:Hanoiklein.jpg

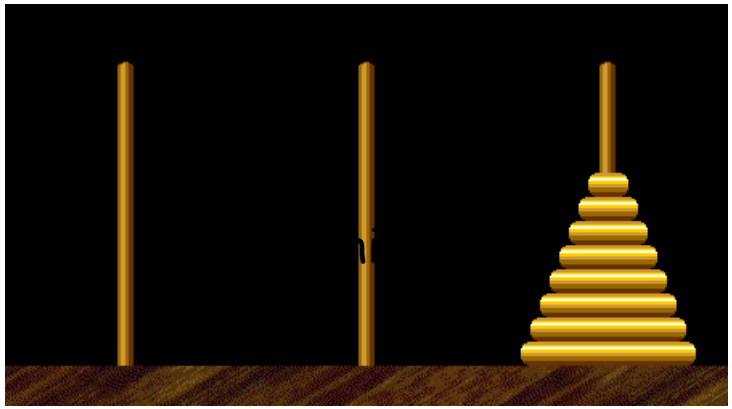
Penn Engineering

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.





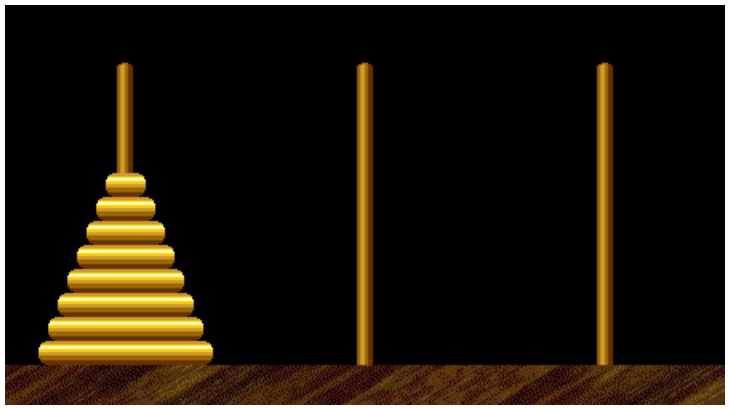
Towers of Hanoi Legend

Q. Is world going to end (according to legend)?

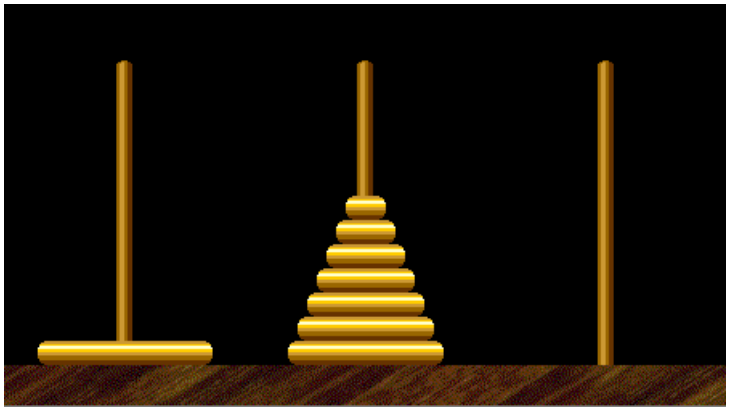
- 64 golden discs on 3 diamond pegs.
- World ends when certain group of monks accomplish task.

Q. Will computer algorithms help?

Towers of Hanoi: Recursive Solution

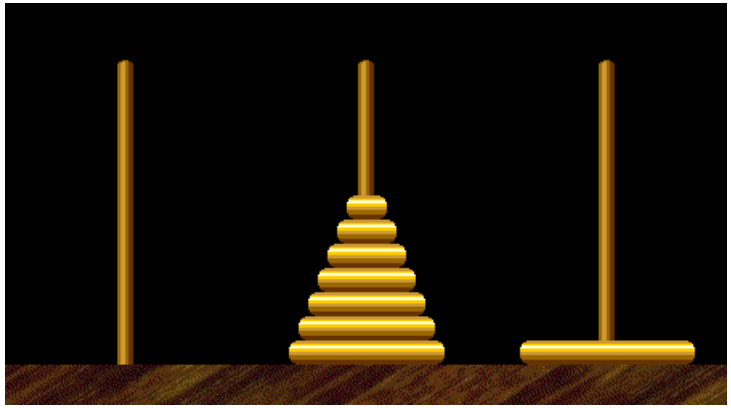


Move n-1 smallest discs right.

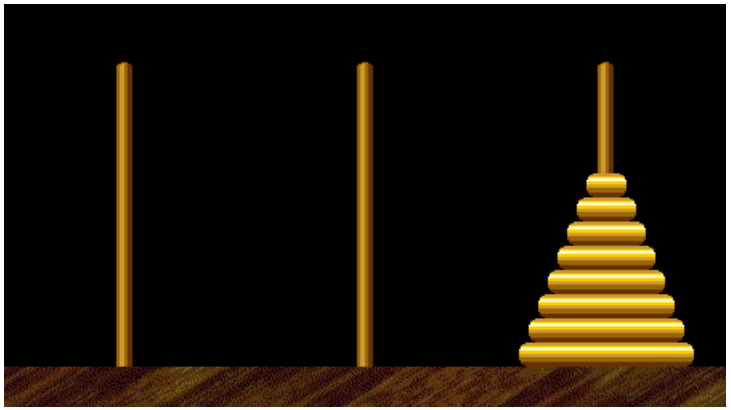


Move largest disc left.

cyclic wrap-around



Move n-1 smallest discs right.



Penn Engineering

Towers of Hanoi: Recursive Solution

```
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

Penn Engineering

Towers of Hanoi: Recursive Solution

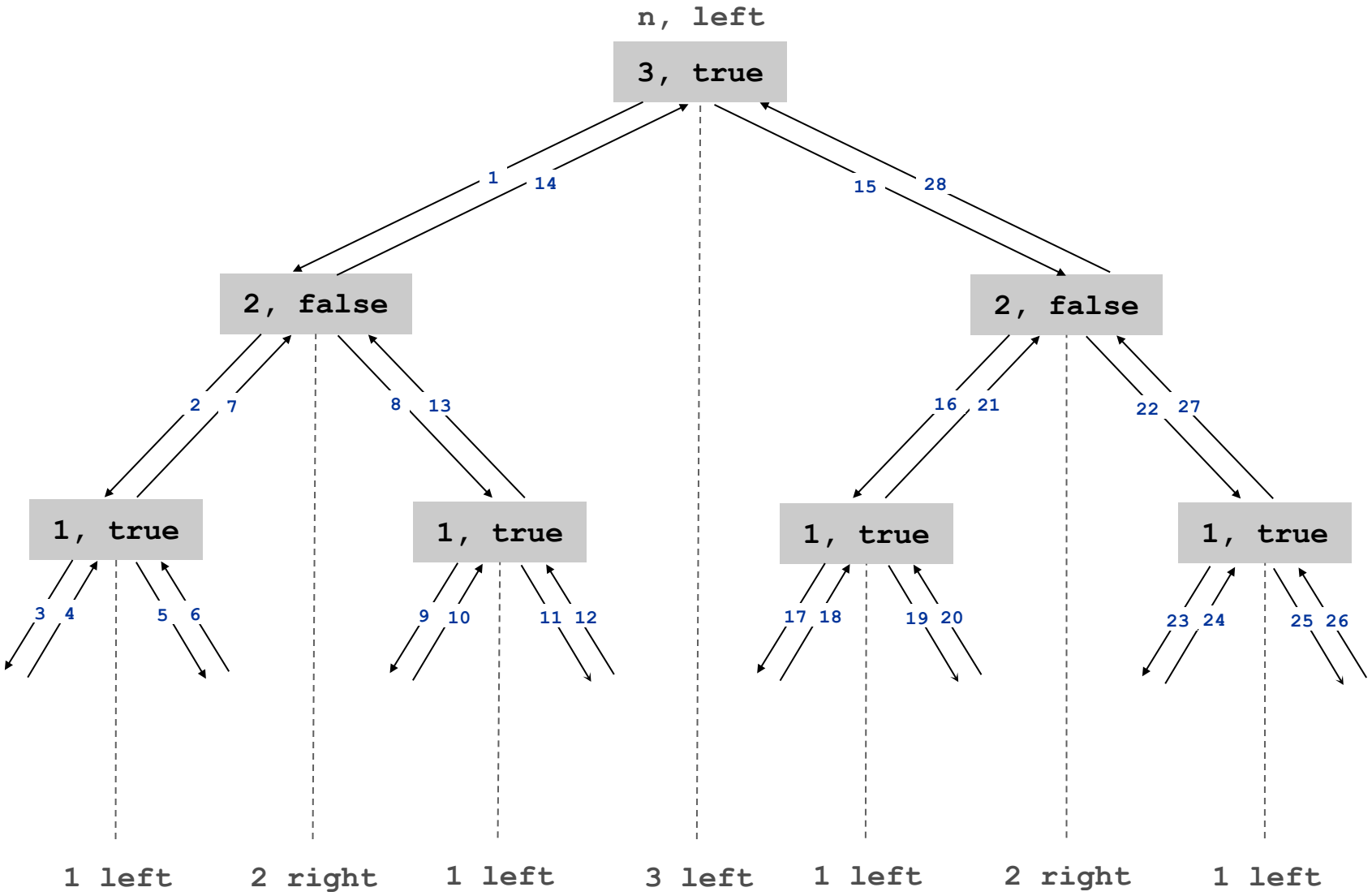
```
% java TowersOfHanoi 3
1 left
2 right
1 left
3 left
1 left
2 right
1 left
```

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

every other move is smallest disc

subdivisions of ruler

Towers of Hanoi: Recursion Tree



Penn Engineering

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.

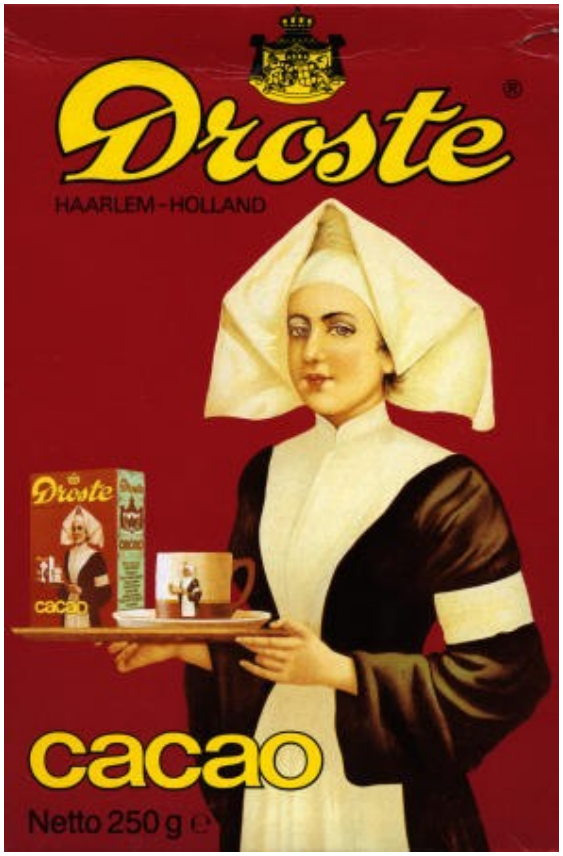
Recursive algorithm yields non-recursive solution!

- 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!

Penn Engineering



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

Fruits of Design, Certified Organic

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.

ROBERTA SMITH
ART REVIEW

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

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.

WILLIAM GRIMES
BOOKS

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 1,320 pages and 10 pounds 12 ounces, "New York

Continued on Page 46

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 Özkaya 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.

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

Mr. Özkaya, 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

Divine and Devotee Meet Across Hinges

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.

HOLLAND COTTER
ART REVIEW

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, comically 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.*

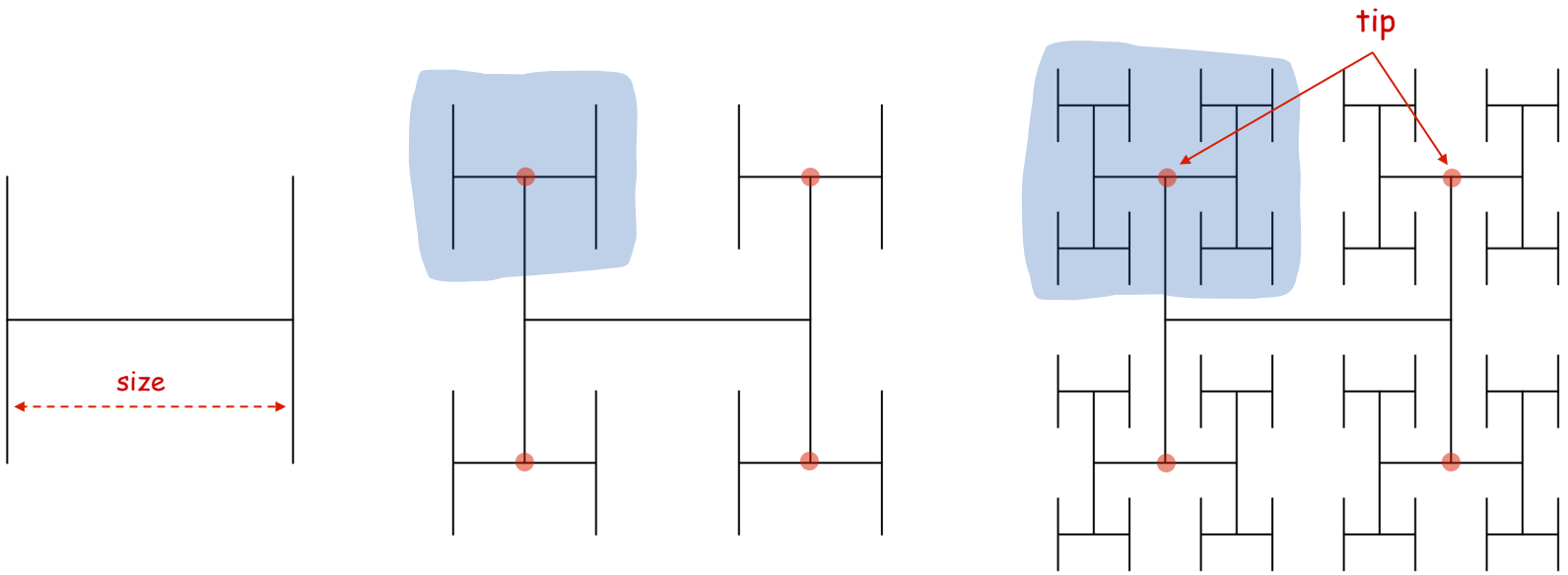
Penn Engineering

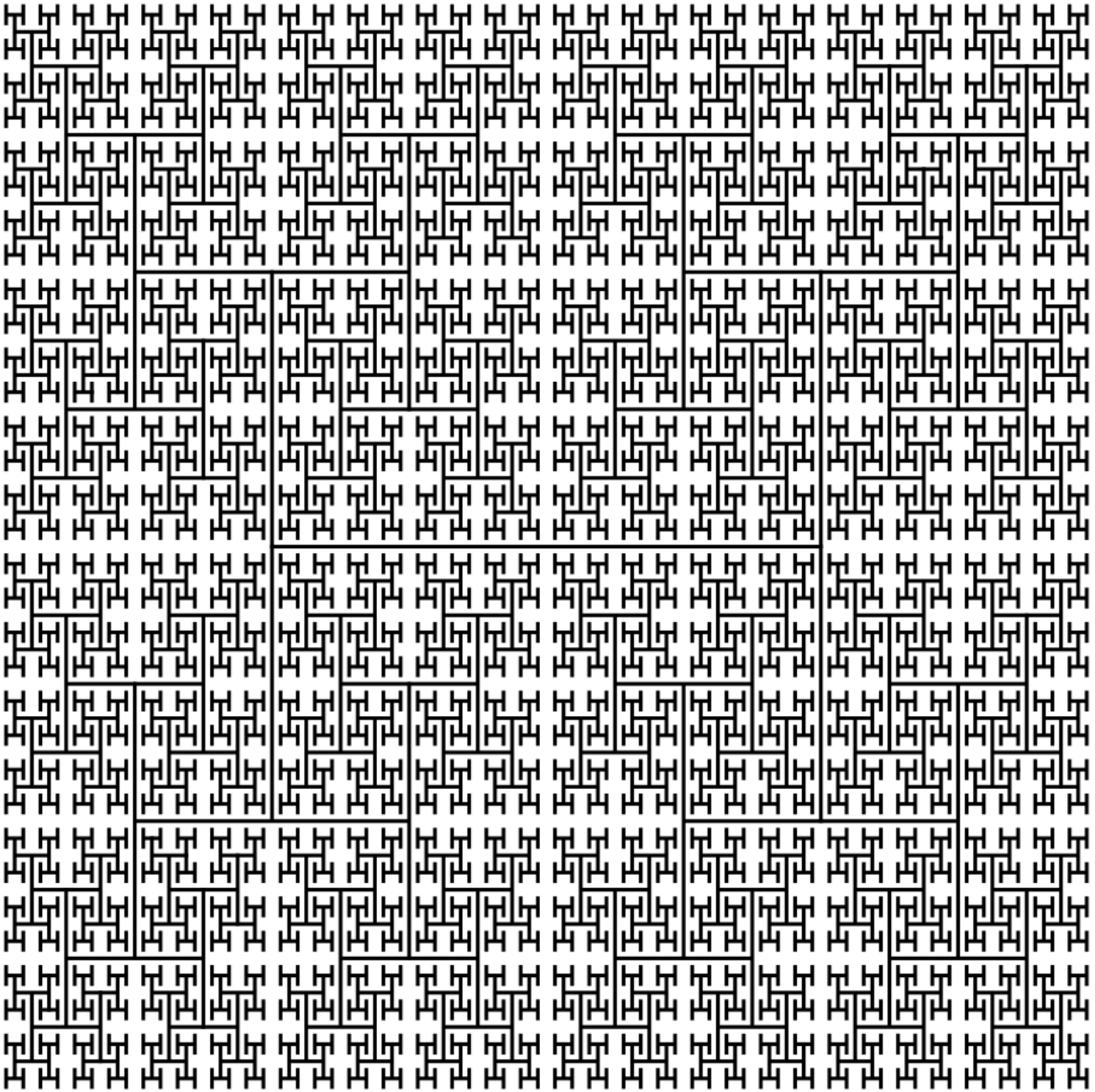
Htree

H-tree of order n.

and half the size

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

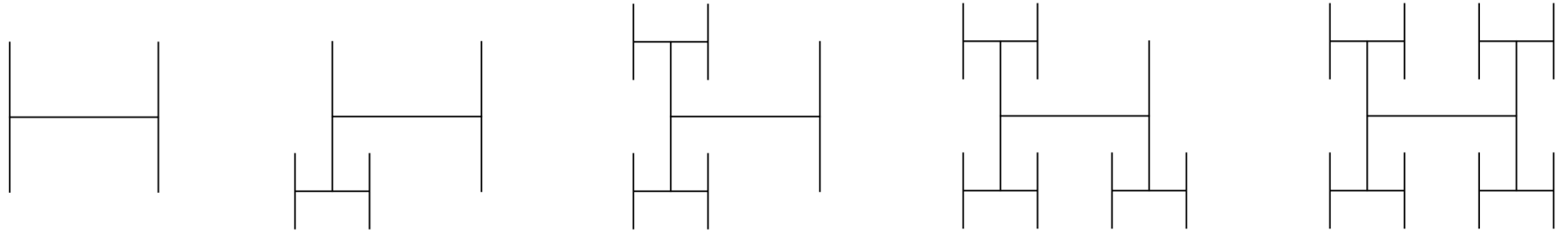




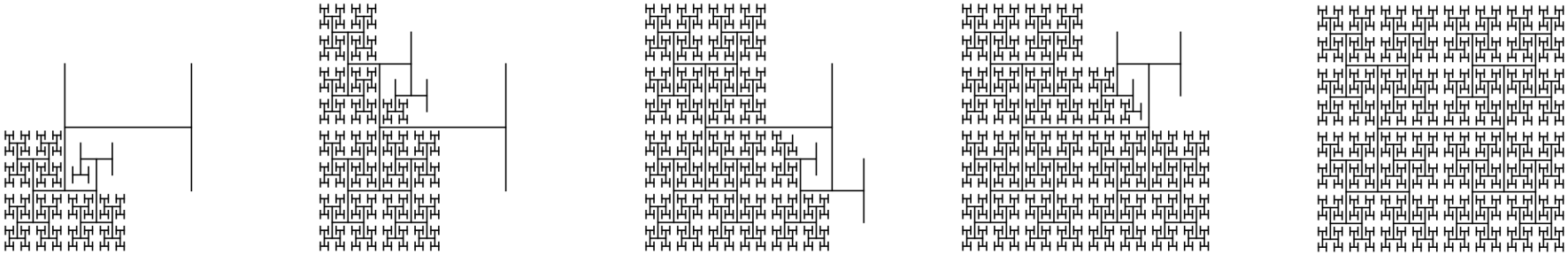
Penn Engineering

Animated H-tree

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



20% 40% 60% 80% 100%

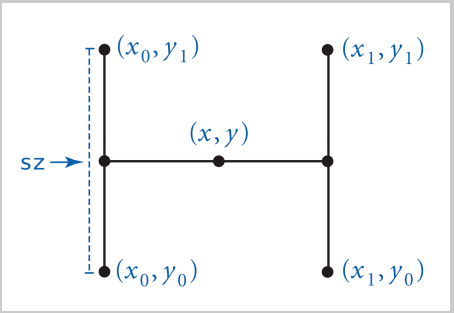


Htree in Java

```
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);
      StdDraw.line(x1, y0, x1, y1);

      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);
   }
}
```

← draw the H, centered on (x, y)

← recursively draw 4 half-size Hs



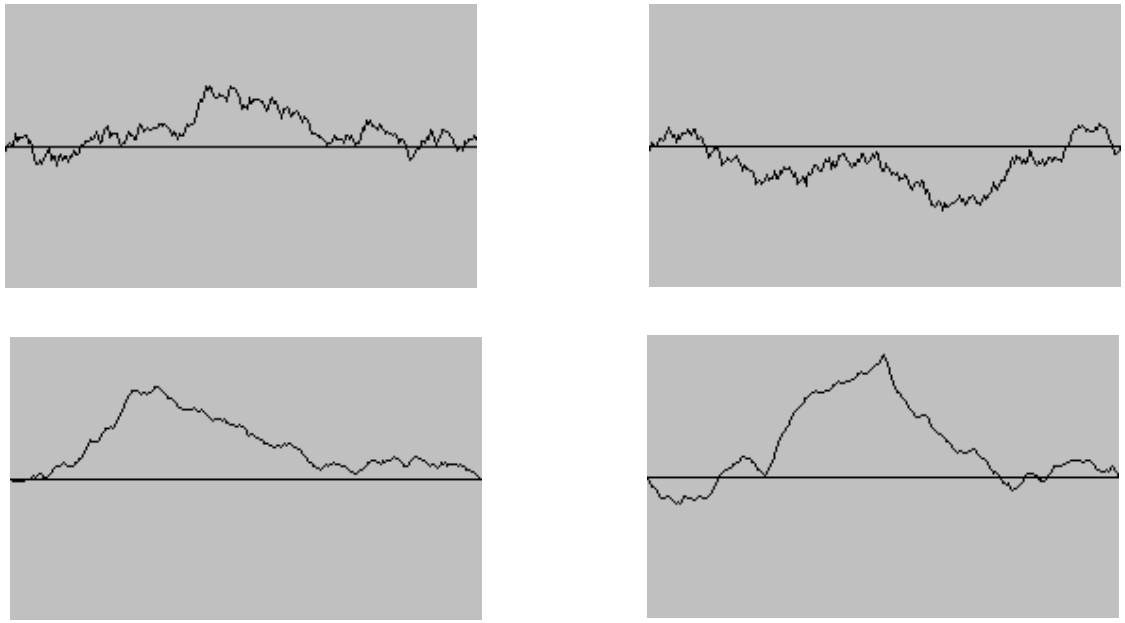
Penn Engineering

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.

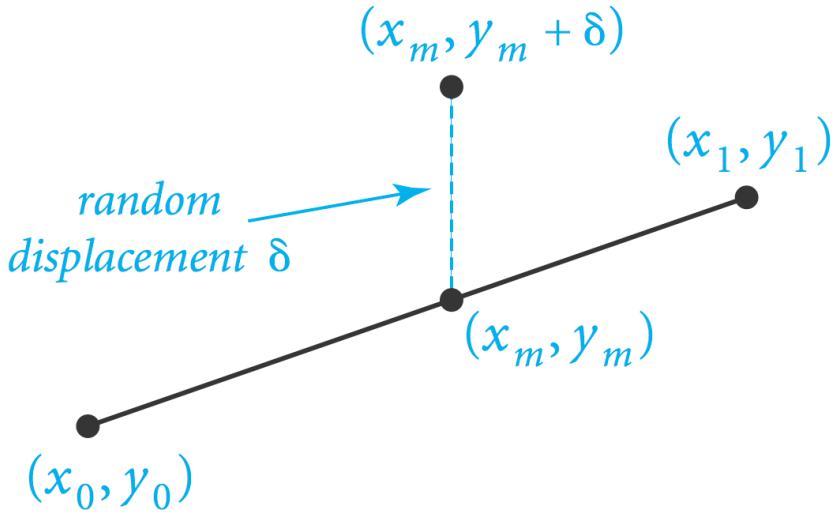


Penn Engineering

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 δ 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.



Penn Engineering

Simulating Brownian Motion: Java Implementation

Midpoint displacement method.

- Maintain an interval with endpoints (x_0, y_0) and (x_1, y_1).
- Divide the interval in half.
- Choose δ 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.

```
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

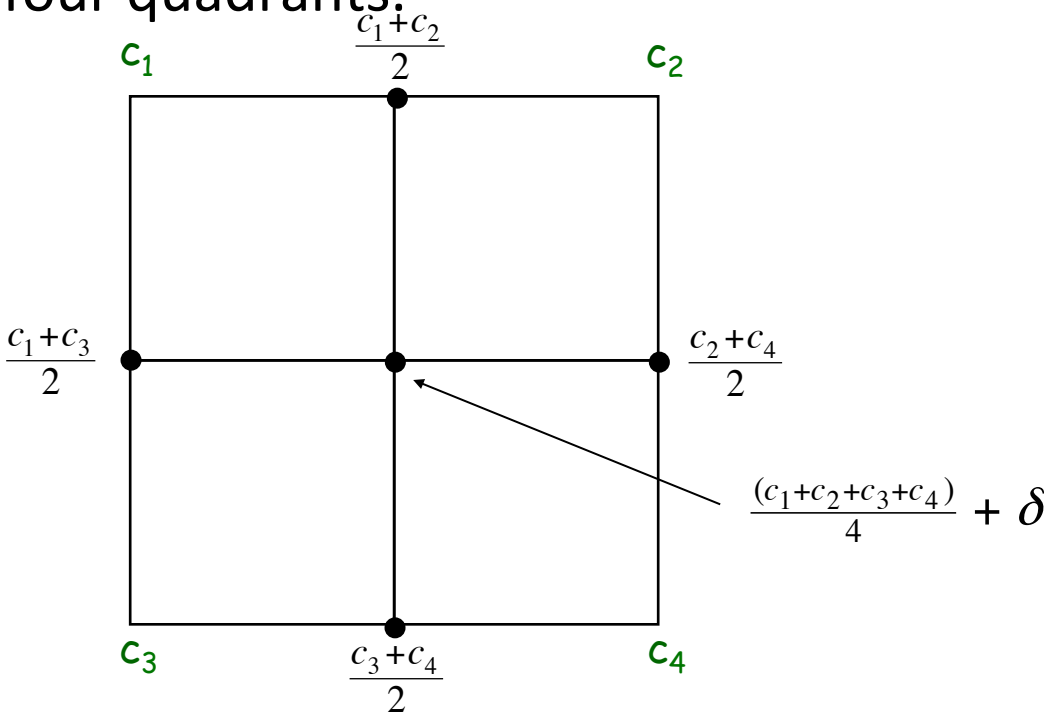
Recursive subdivision

- Divide your original area into smaller segments
- Recurse inside each of the segments
- With each recursive step add a little bit of randomness
- Patterns like Brownian motion, Plasma Clouds etc

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
- Recurse on the four quadrants.

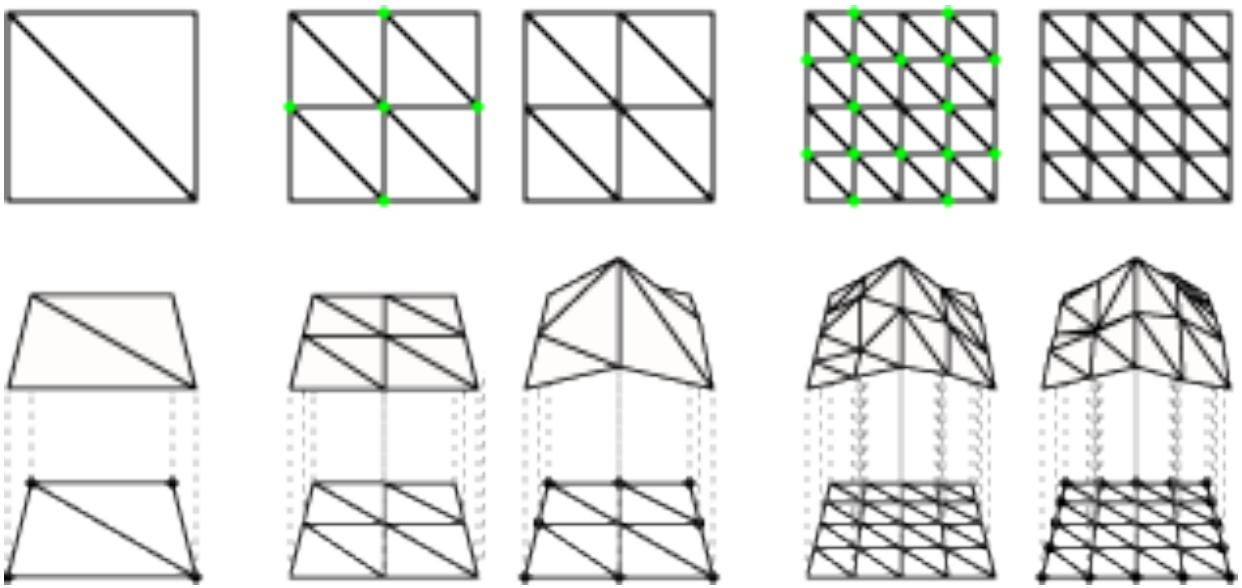


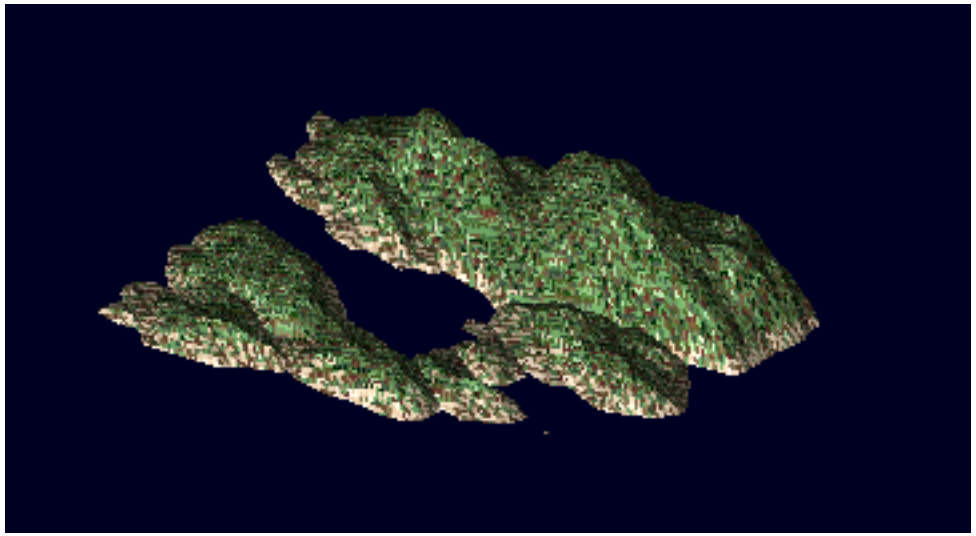
Penn Engineering

Plasma Cloud

Penn Engineering

Brownian Island





Penn Engineering

Brownian Landscape