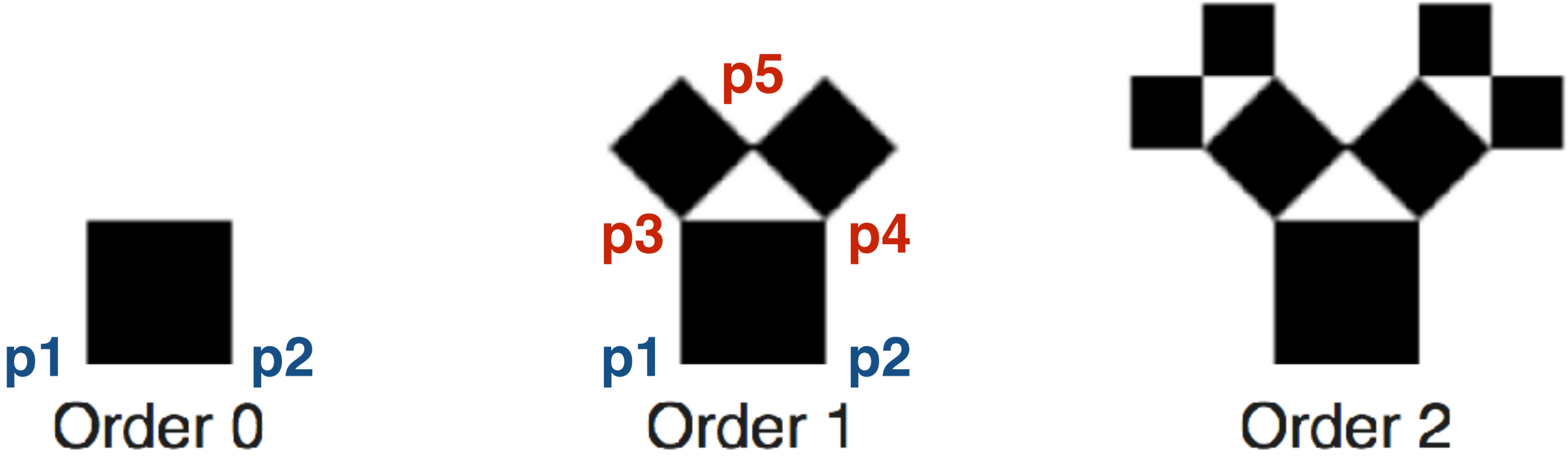
Recursion in 20”

Mircea F. Lungu
University of Groningen
Netherlands

@mircealungu

https://github.com/mircealungu/open_lectures/

Pythagoras' Tree



Code on GitHub

```
private void drawTree(Graphics2D g, Point2D p1, Point2D p2,
                      int depth) {

    if (depth == depthLimit)
        return;

    Point2D p3 = new Point2D.Double();
    Point2D p4 = new Point2D.Double();
    Point2D p5 = new Point2D.Double();

    computeNewPoints(p1, p2, p3, p4, p5);

    drawSquare(g, p1, p2, p3, p4, depth);

    drawTree(g, p4, p5, depth + 1);
    drawTree(g, p5, p3, depth + 1);
}
```

Code on GitHub

```
private void drawTree(Graphics2D g, Point2D p1, Point2D p2,
                      int depth) {

    if (depth == depthLimit)
        return;

    Point2D p3 = new Point2D.Double();
    Point2D p4 = new Point2D.Double();
    Point2D p5 = new Point2D.Double();

    computeNewPoints(p1, p2, p3, p4, p5);

    drawSquare(g, p1, p2, p3, p4, depth);

    drawTree(g, p4, p5, depth + 1);
    drawTree(g, p5, p3, depth + 1);
}
```

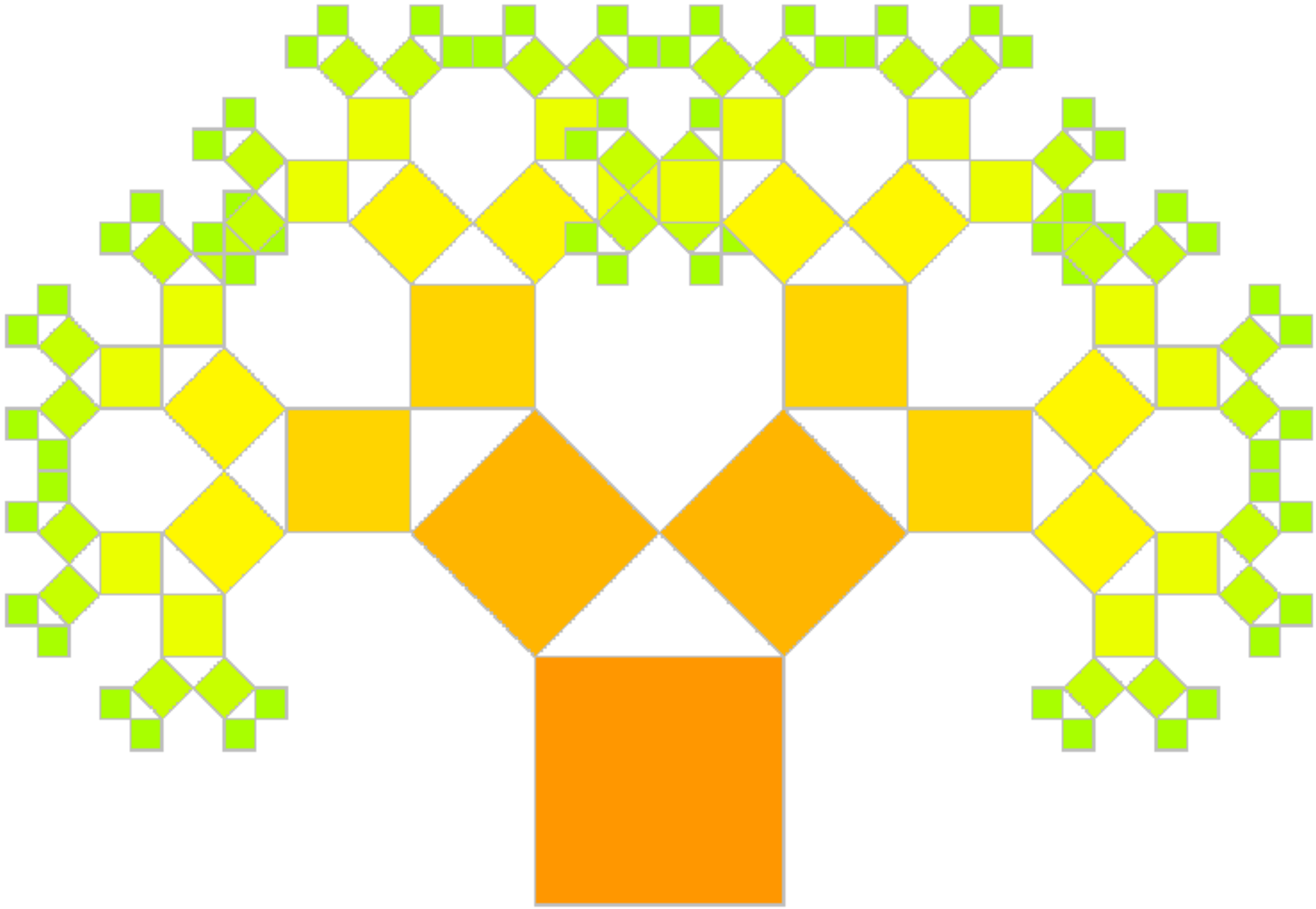
base case

when reached causes recursion to end

recursive == calling itself + converging to the base case

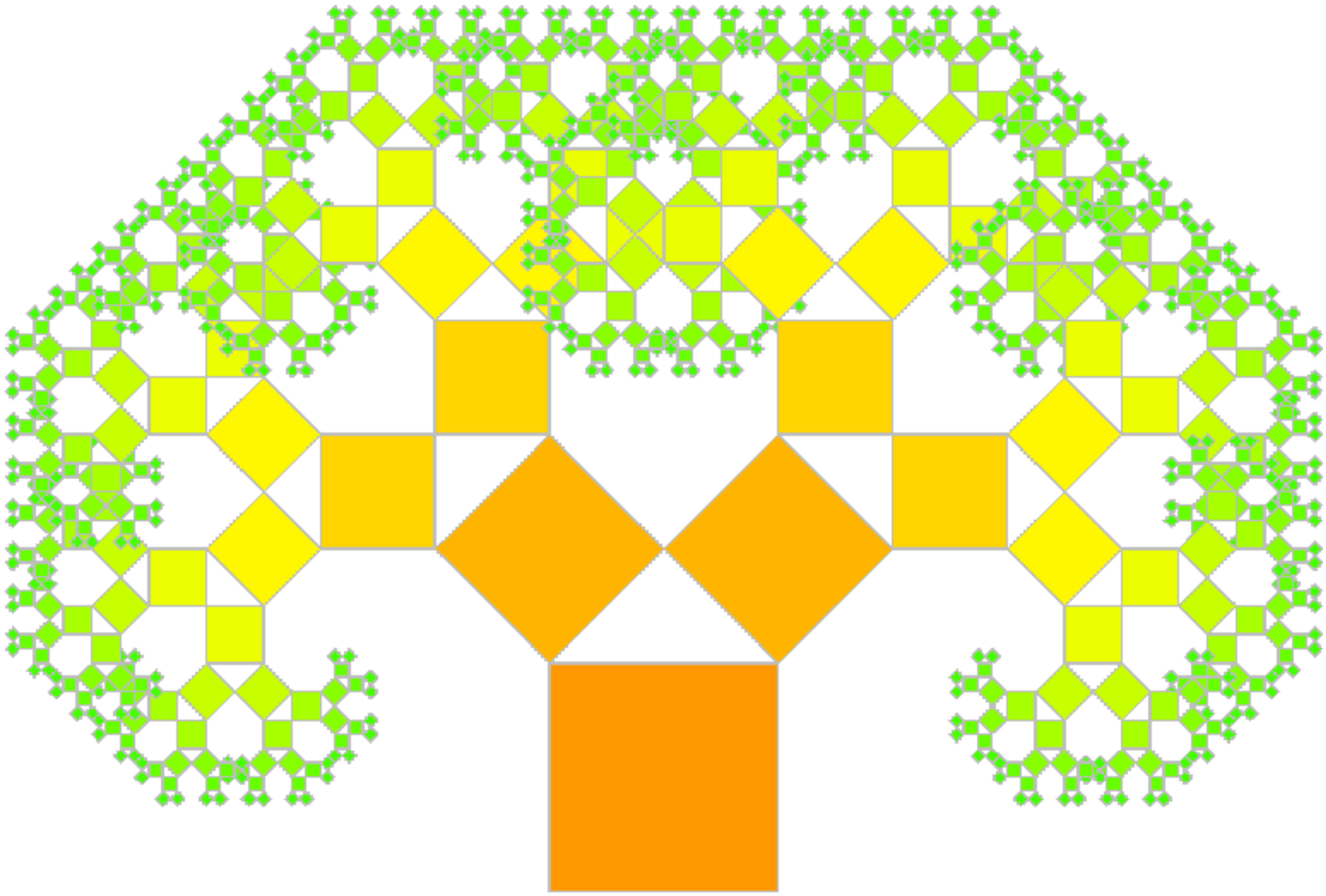
Code on GitHub

Order 7



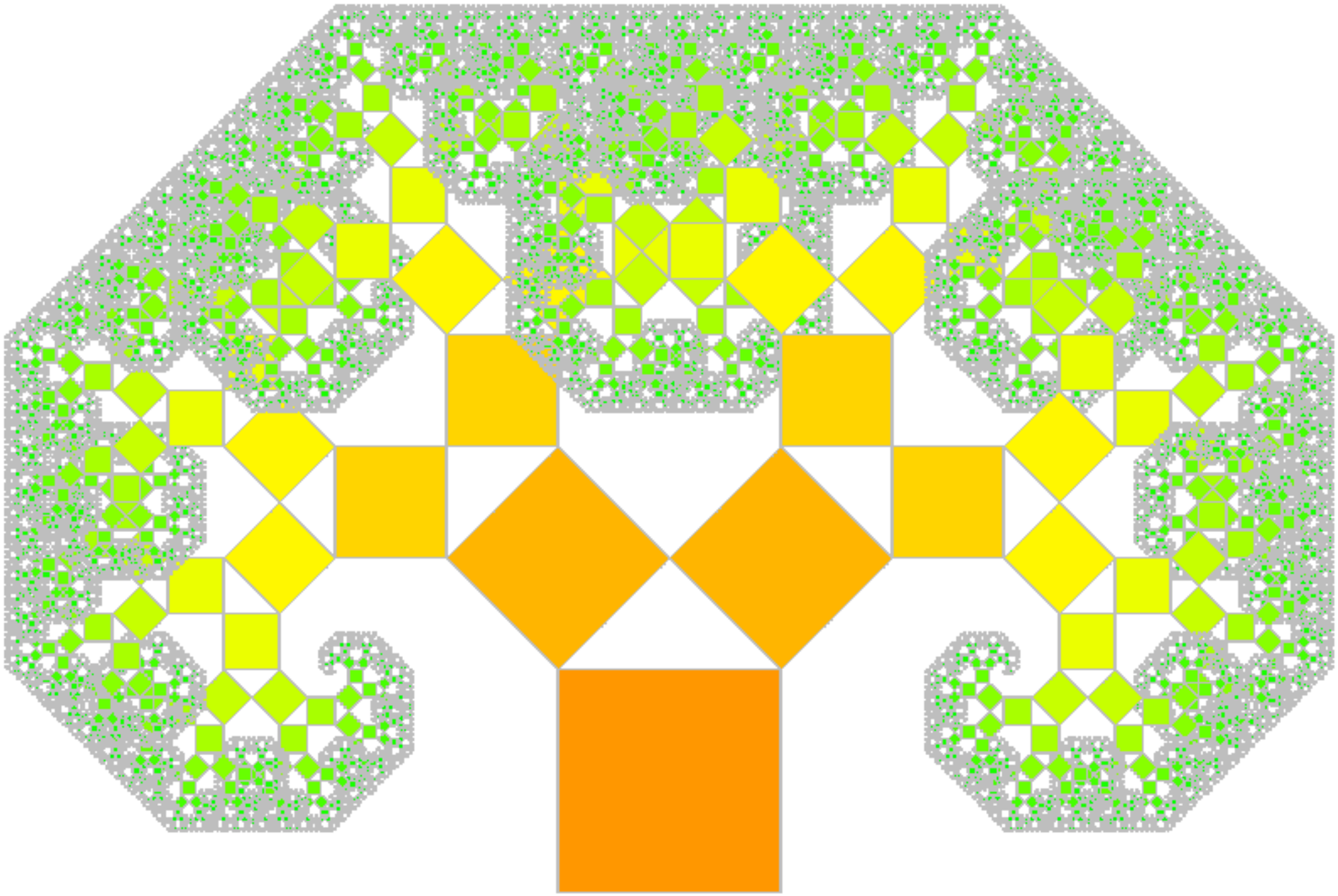
Code on GitHub

Order 10



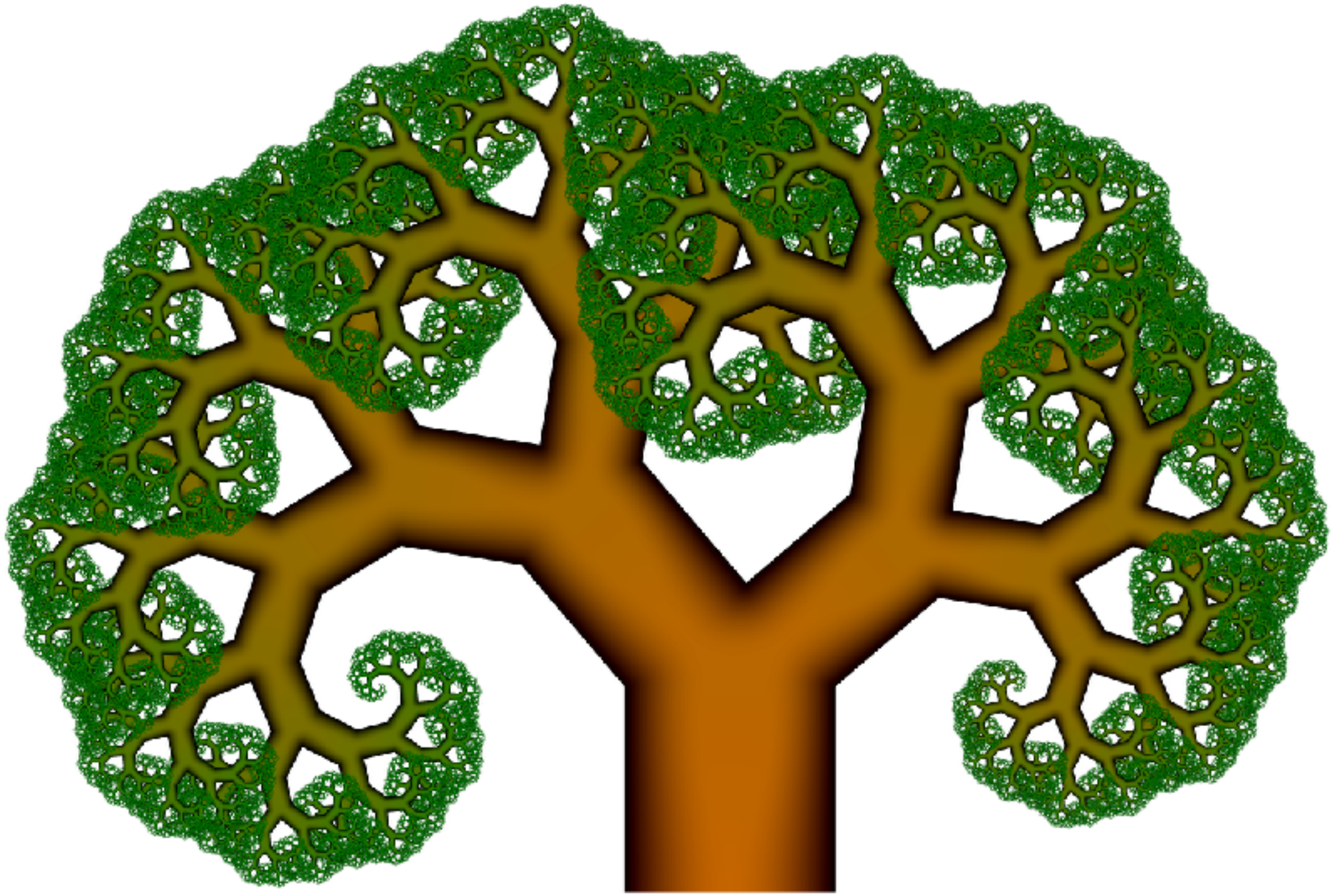
Code on GitHub

Order 15



Code on GitHub

Not on GitHub anymore!!!



By Atze van der Ploeg Atzecsse - Own work,
Attribution, https://commons.wikimedia.org/w/index.php?curid=28183362

The power of recursion lies in the possibility of defining

- **an infinite set of objects by a finite statement**

- **an infinite number of computations by a finite recursive program**

Algorithms + Data Structures = Programs,
Wirth, Niklaus (1976)

The power of recursion lies in the possibility of defining

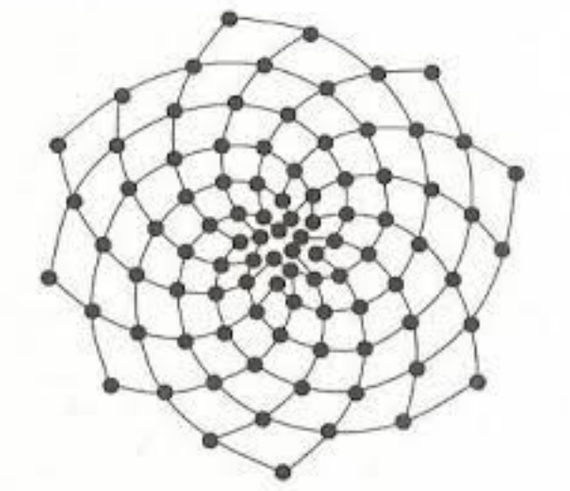
- **an infinite set of objects by a finite statement**

- **an infinite number of computations by a finite recursive program**

Algorithms + Data Structures = Programs,
Wirth, Niklaus (1976)

$$F_n = F_{n-1} + F_{n-2}$$

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610



http://www.sciencemag.org/news/2016/05/sunflowers-show-complex-fibonacci-sequences

Fibonacci Numbers in Java

```
public static long fibonacci(final int n)
{
    return (n < 2) ? n : fibonacci(n - 1) + fibonacci(n - 2);
}
```

base case

recursive calls

Exponential Growth

```
                F(n)
               /    \
          F(n-1)     F(n-2)
          /    \      /    \
      F(n-2) F(n-3) F(n-3)  F(n-4)
      /    \
  F(n-3) F(n-4)
```

Dynamic Programming - method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions

Demo — scalability of naïve Fibonacci

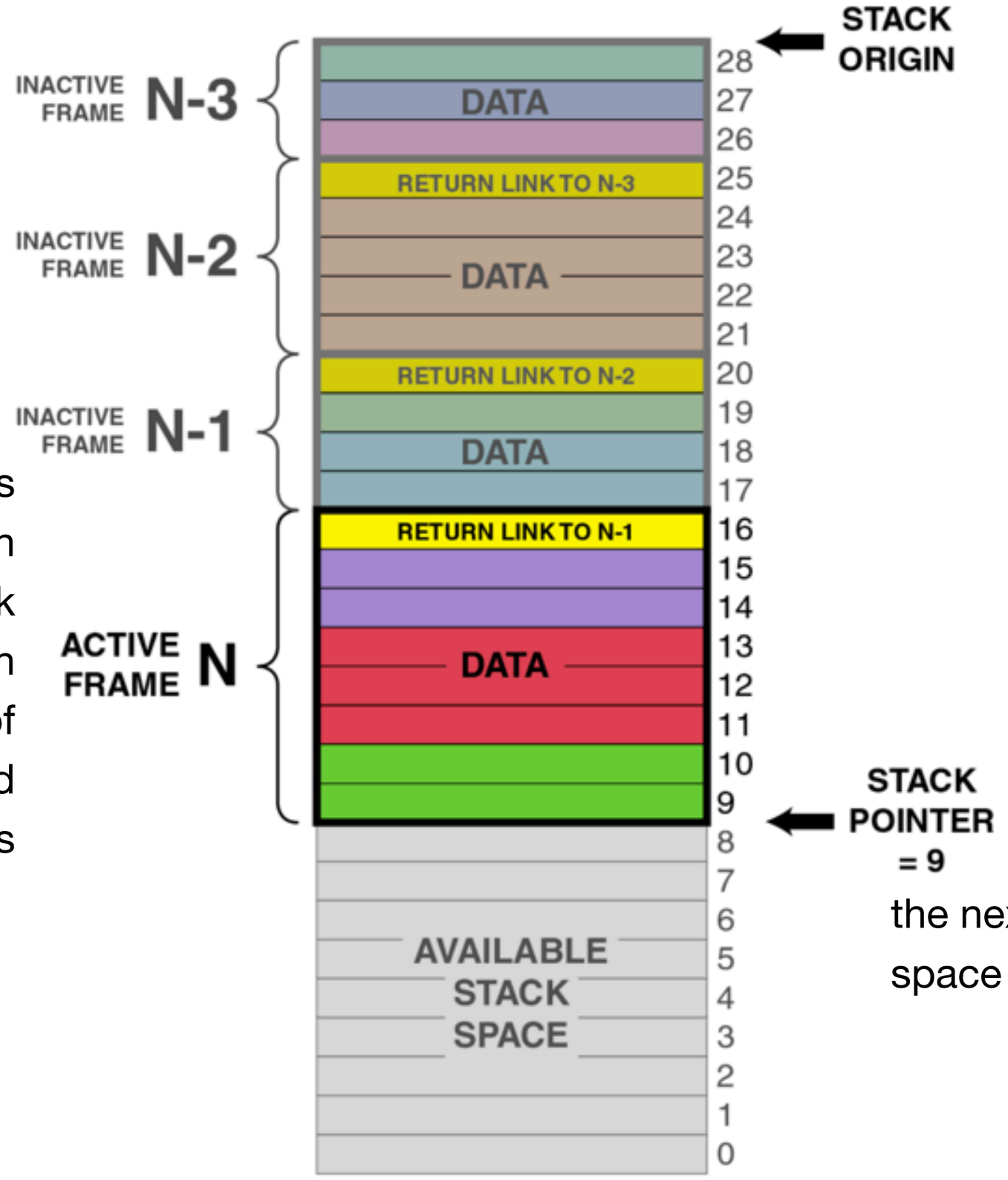
Faster Version!

```
public static long fasterFibonacci(final int n)
{
    if ( n <= 1 )
        return n;

    return fibonacciAcc(0, 1, n);
}
```

```
private static long fibonacciAcc(long prev, long curr, int n)
{
    if (n == 1)
        return curr;

    return fibonacciAcc(curr, prev + curr, n - 1);
}
```



frame pointers point to activation record, or stack frame, which contains copies of local variables and return address

the next available storage space on the stack

The Call Stack

language structure which allows interrupting and resuming a function


```java
public static long fasterFibonacci(final int n)
{
    if ( n == 1 )
        return n;

    return fibonacciAcc(0, 1, n);
}

public static void main(String[] args) {
    for ( int counter = 0; counter <= 50; counter++ )
        System.out.printf( "Fibo(%d) = %d\n", counter, fasterFibonacci(counter)
}
```

Fibonacci

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
Exception in thread "main" java.lang.StackOverflowError
    at Fibonacci.fibonacciAcc(Fibonacci.java:11)
    at Fibonacci.fibonacciAcc(Fibonacci.java:11)
    at Fibonacci.fibonacciAcc(Fibonacci.java:11)
```

Demo - Testing the depth of Java's Stack

Tail Recursion

```
private static long fibonacciAcc(long prev, long curr, int n)
{
    if (n == 1)
        return curr;

    return fibonacciAcc(curr, prev + curr, n - 1);
}
```

can be optimized with *tail call optimization*
easy to transform in iteration (challenge for you!)

The power of recursion lies in the possibility of defining

- **an infinite set of objects by a finite statement**

- **an infinite number of computations by a finite recursive program**

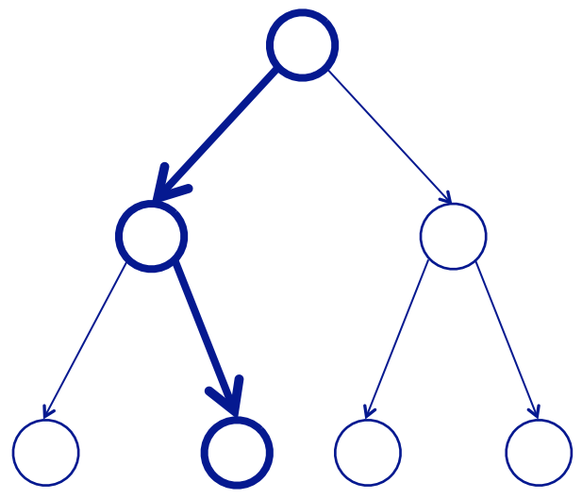
Algorithms + Data Structures = Programs,
Wirth, Niklaus (1976)

Modeling Binary Trees

```
class Node<T> {
    T value;
    Node<T> left;
    Node<T> right;

    Node(T value) {
        this.value = value;
    }

    void visit() {
        System.out.print(this.value + " ");
    }
}
```



Traversing Trees

recursive algos for recursive structures

```
static void traverse(Node<?> node, ORDER order) {
    if (node == null) {
        return;
    }
    switch (order) {
        case PREORDER:
            node.visit();
            traverse(node.left, order);
            traverse(node.right, order);
            break;
```

base case

recursive calls

"structural recursion" (HTDP, MIT Press)

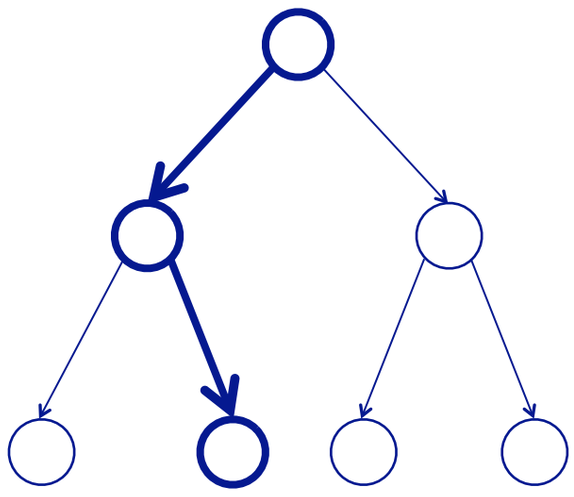
The power of recursion lies in the possibility of defining

- **an infinite set of objects by a finite statement**

- **an infinite number of computations by a finite recursive program**

Algorithms + Data Structures = Programs,
Wirth, Niklaus (1976)

Binary search



```
/*
      a - array with integer elements
      x - element we want to know whether present
      left - leftmost position where element could be
      right - rightmost position where element could be
 */

private int binarySearch(int[ ] a, int x, int left, int right) {
```

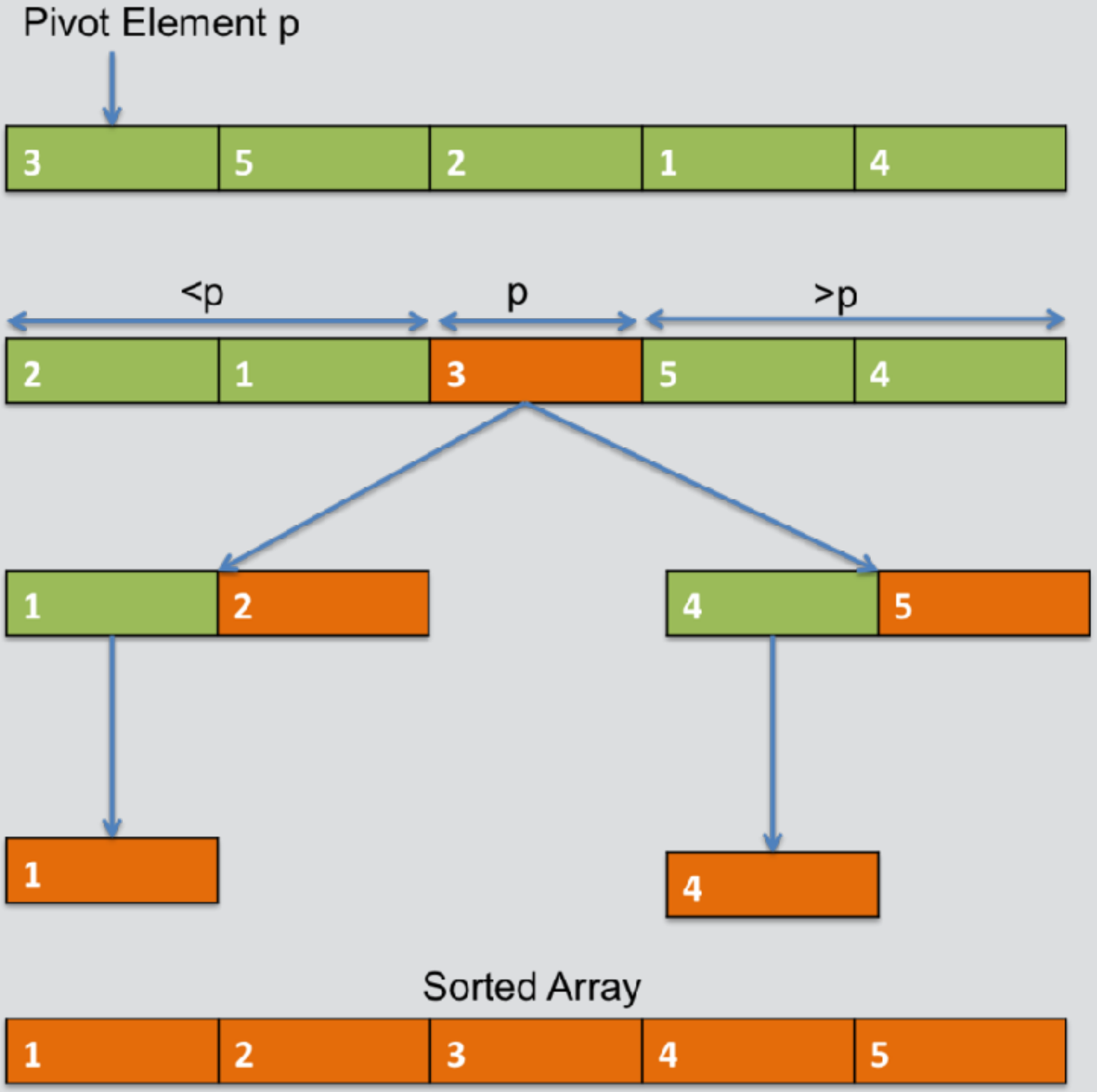
?

base case

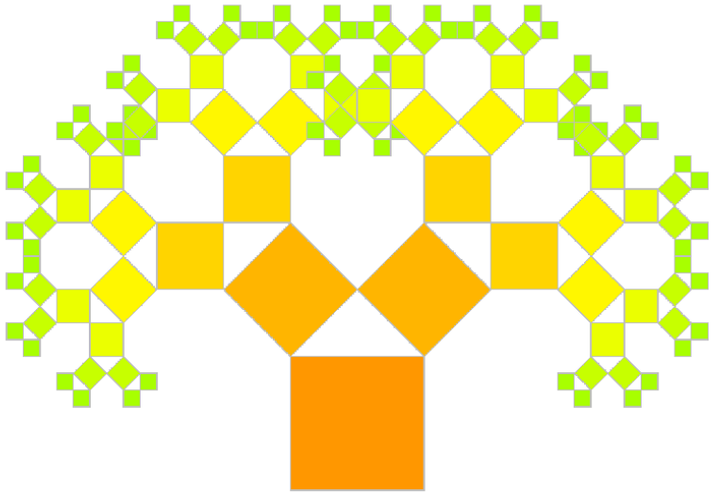
```
    else if (a[mid] < x)
        return binarySearch(a, x, mid+1, right);

    else
        return binarySearch(a, x, left, mid-1);
}
```

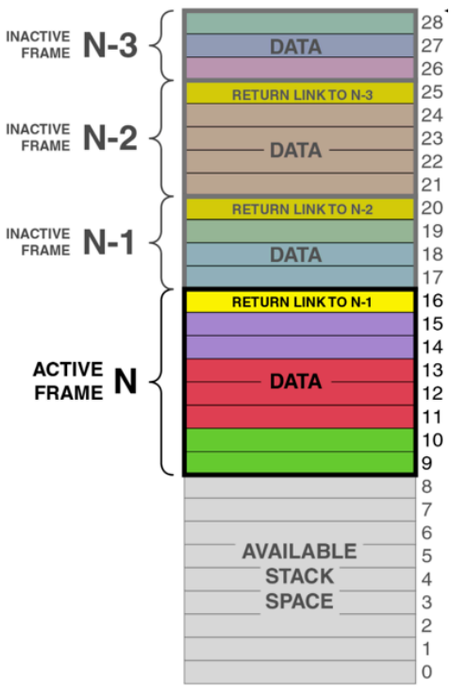
recursive calls

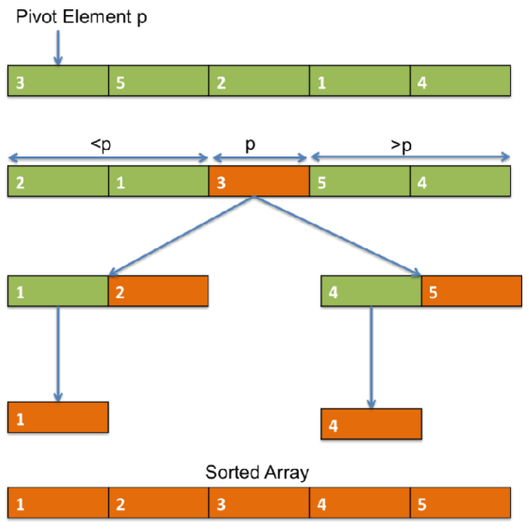


Quicksort



```
                F(n)
               /    \
          F(n-1)    F(n-2)
          /   \      /      \
     F(n-2) F(n-3) F(n-3)  F(n-4)
    /    \
F(n-3) F(n-4)
```





Code: https://github.com/mircealungu/open_lectures/recursion

Further Reading

- Slides on Recursion (C. Simpkins, Georgia Tech)
- Tail Call Optimization (DrDobbs)
- Replacing Recursion with Iteration (ThoughtWorks)
- Natural Language Parsing (NLTK)
- Dynamic Programming (CodeChef)

- Title Slide Image: http://qcc-art.deviantart.com/art/Tony-Monahan-Turtles-All-the-Way-Down-590318917

- Book image from Oscar Nierstrasz, U. Bern

- Pythagora Tree code adapted from rosettacode.org

https://github.com/mircealungu/open_lectures/

Backup Materials

Replacing Recursion with Iteration

```
Stack<Object> stack;
stack.push(first_object);
while( !stack.isEmpty() ) {
    // Do something
    my_object = stack.pop();

  // Push other objects on the stack.

}
```

```
void quicksort(int *array, int left, int right)
{
    int stack[1024];
    int i=0;

    stack[i++] = left;
    stack[i++] = right;

    while (i > 0)
    {
        right = stack[--i];
        left = stack[--i];

        if (left >= right)
            continue;

        int index = partition(array, left, right);
        stack[i++] = left;
        stack[i++] = index - 1;
        stack[i++] = index + 1;
        stack[i++] = right;
    }
}
```

iterative quicksort

Composite Design Pattern