Recursion

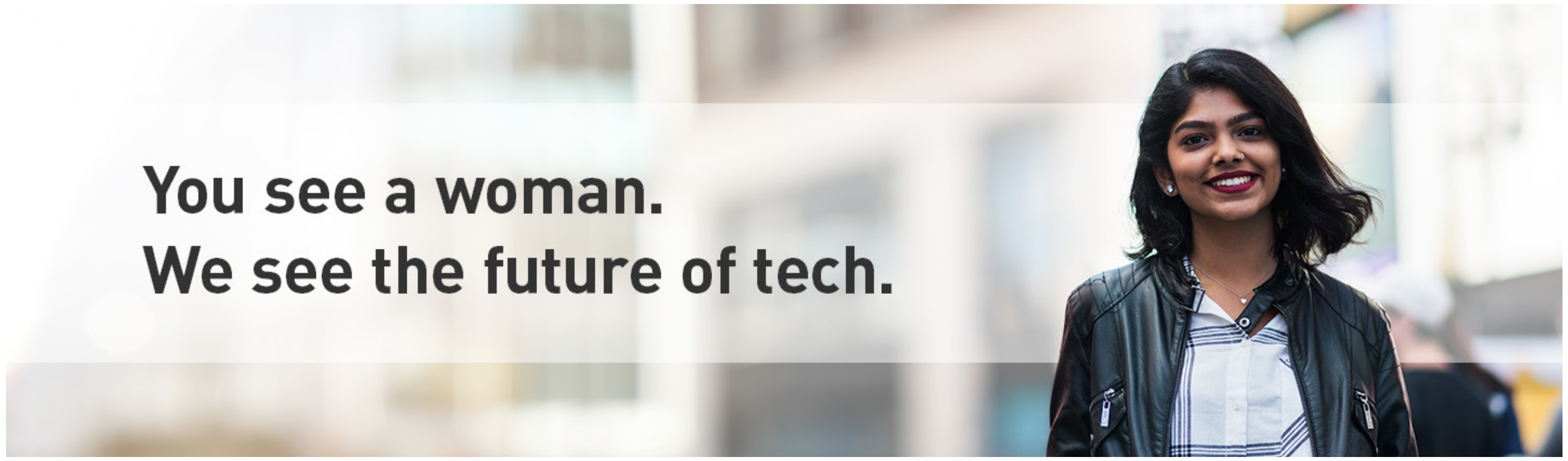
Tiziana Ligorio

tligorio@hunter.cuny.edu

Today's Plan

Announcements

Recursion



We Are Digital Enthusiasts

Jump start your future this January with a three-week Winternship.

What's a Winternship?
A Winternship is a PAID, three-week internship in NYC, open to first- and second-year women at CUNY during their January academic recess.

Why should I apply for a Winternship?
You'll learn more about job opportunities in tech and computing, build your resume, and expand your professional network.

Who can apply?
All first- and second-year women at CUNY who are interested in learning more about tech careers. You may be a computer science major, or you may not be. There are no academic requirements to apply. What are you waiting for?

Applications are now open for WiTNY Winternship!

APPLY TODAY!

witny.org/students

IMPORTANT DATES

- October 5, 2018: Applications due
- Mid-November: Placements announced
- January 7-24, 2019: Winternships take place in NYC and the surrounding tri-state area

Questions?

Maria DiKun, Program Coordinator, WiTNY | wit-ny@cornell.edu

WiTNY | Women in Technology and Entrepreneurship in New York

CUNY The City University of New York

CORNELL TECH

Announcements and Syllabus Check

Still running 1 lecture behind!

Let’s talk 10 minutes about Project2!

Write String Backwards

"Hello"

Write String Backwards String

"Hello"

<u>Procedure:</u>

Write the last character and reverse the rest

Write String Backwards

Hello

o

Write String Backwards

Hello

o

Hell

Write String Backwards

Hello

o

    Hell

    o l

Write String Backwards

Hello

o

Hell

o l

Hel

Write String Backwards

Hello

o

Hell

o l

Hel

o l l

Write String Backwards

Hello

o

Hell

o l

Hel

o l l

He

Write String Backwards

Hello

o

Hell

o l

Hel

o l l

He

o l l e

Write String Backwards

Hello

o

Hell

o l

Hel

o l l

He

o l l e

H

Write String Backwards

Hello

o

Hell

o l

Hel

o l l

He

o l l e

H

o l l e H

Write String Backwards

Hello

o

Hell

o l

Hel

o l l

He

o l l e

H

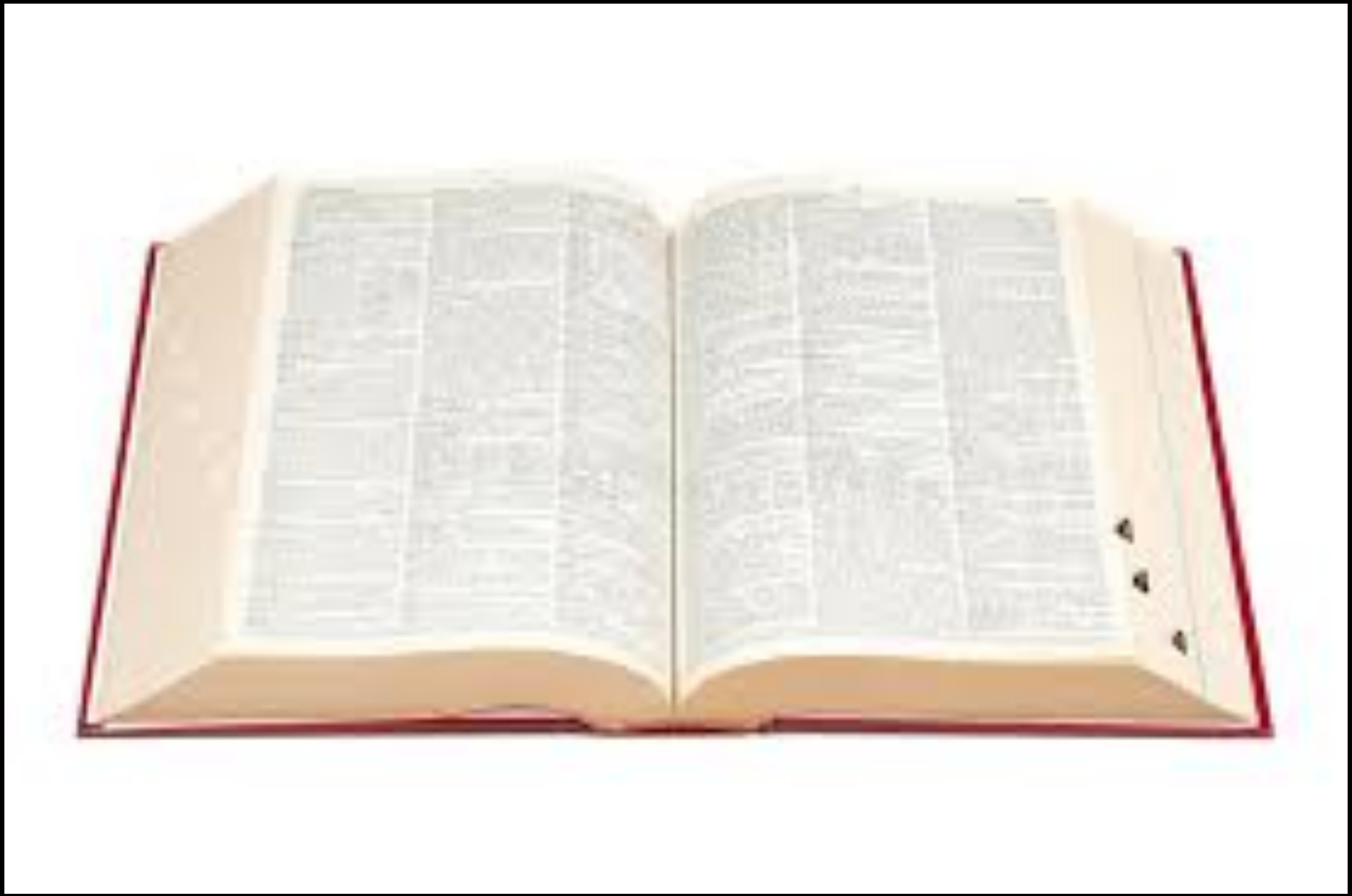
o l l e H

BASE CASE

In-Class Task

If I hand you a printed dictionary and ask you to find the word “Kalimba”, what do you do?

Write down precise steps (a procedure) as if someone who has never seen a dictionary before must follow your instructions.

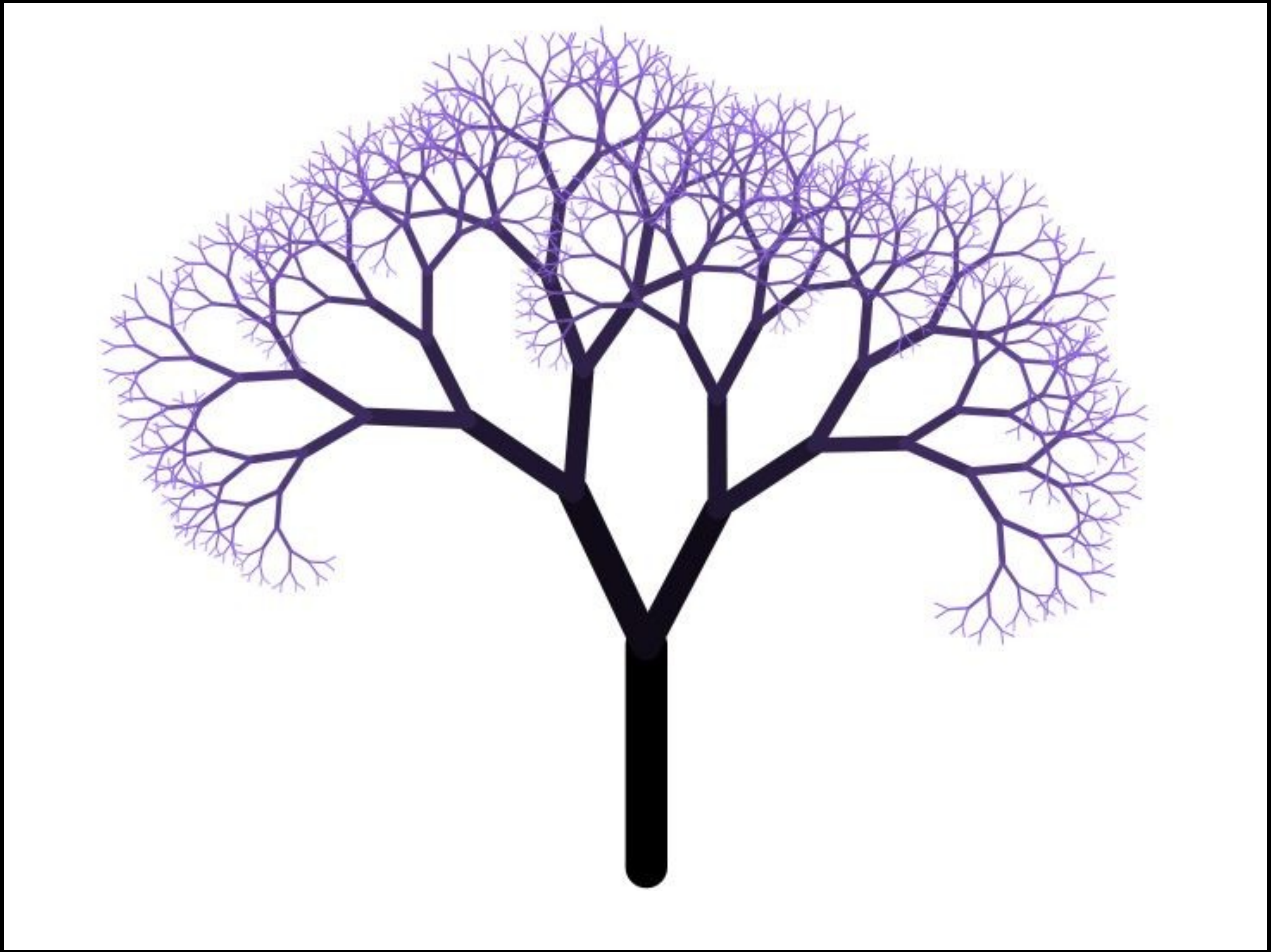


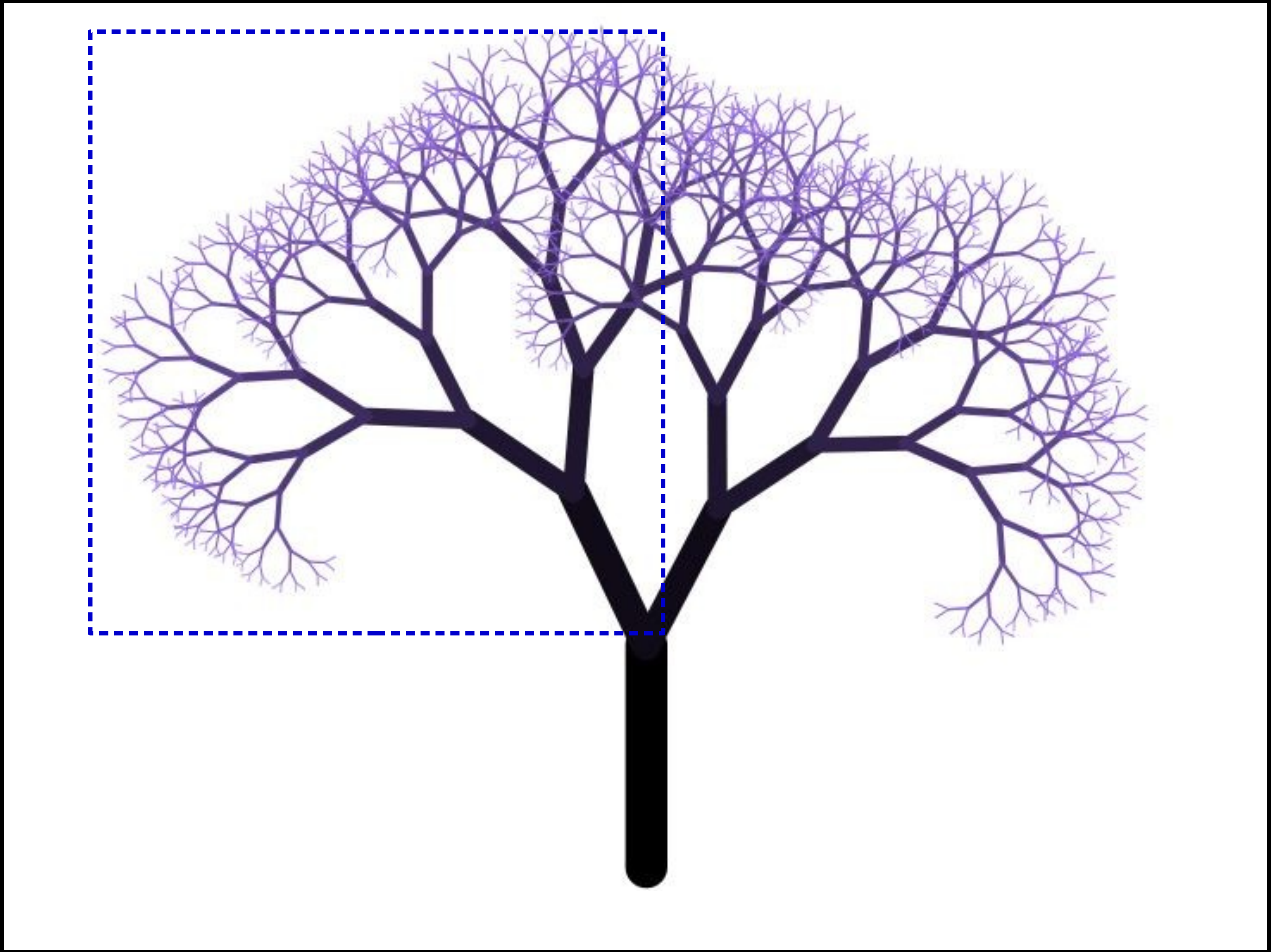
Look in ?

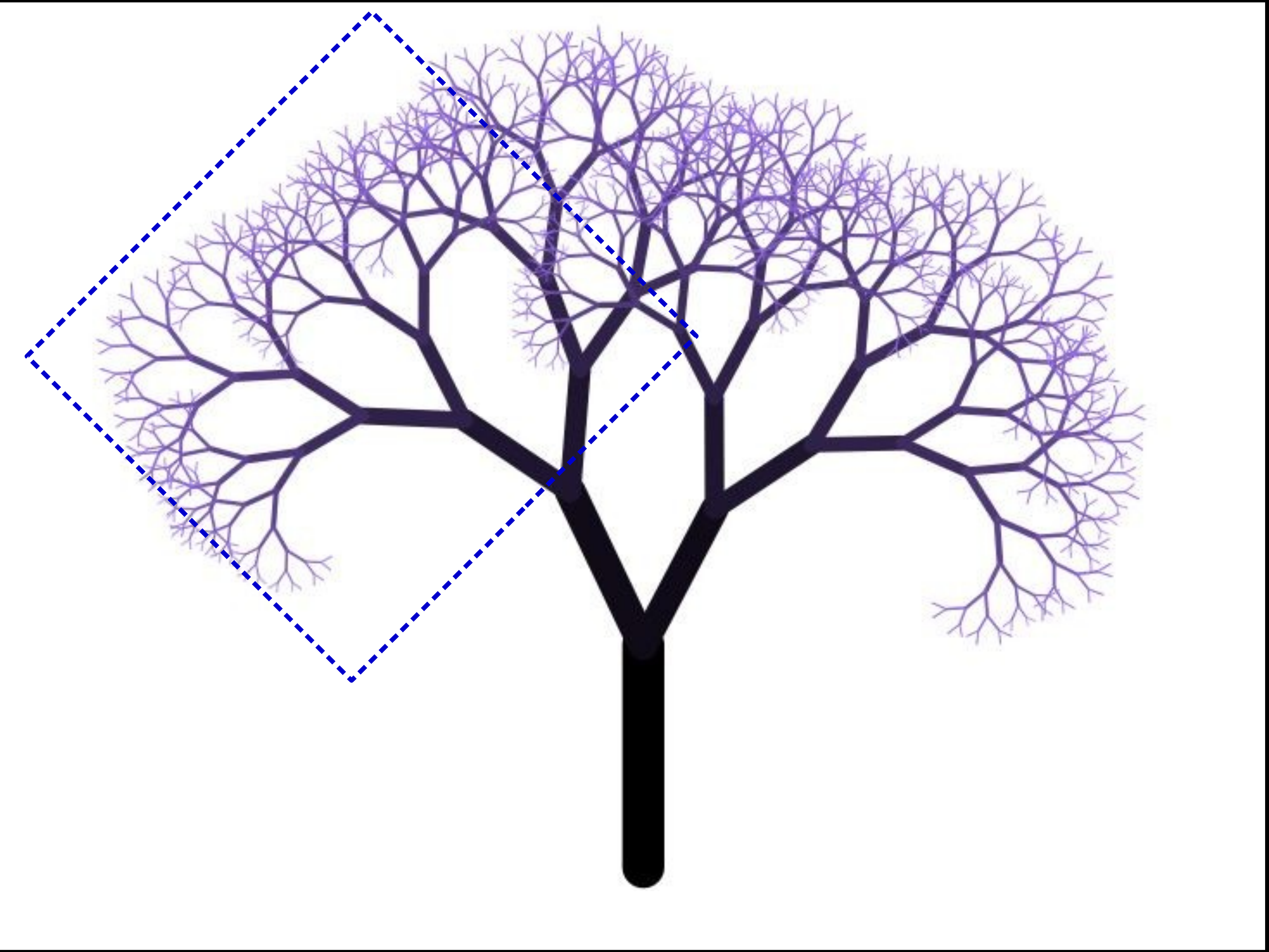
The images in the next slides were borrowed from Keith Schwarz

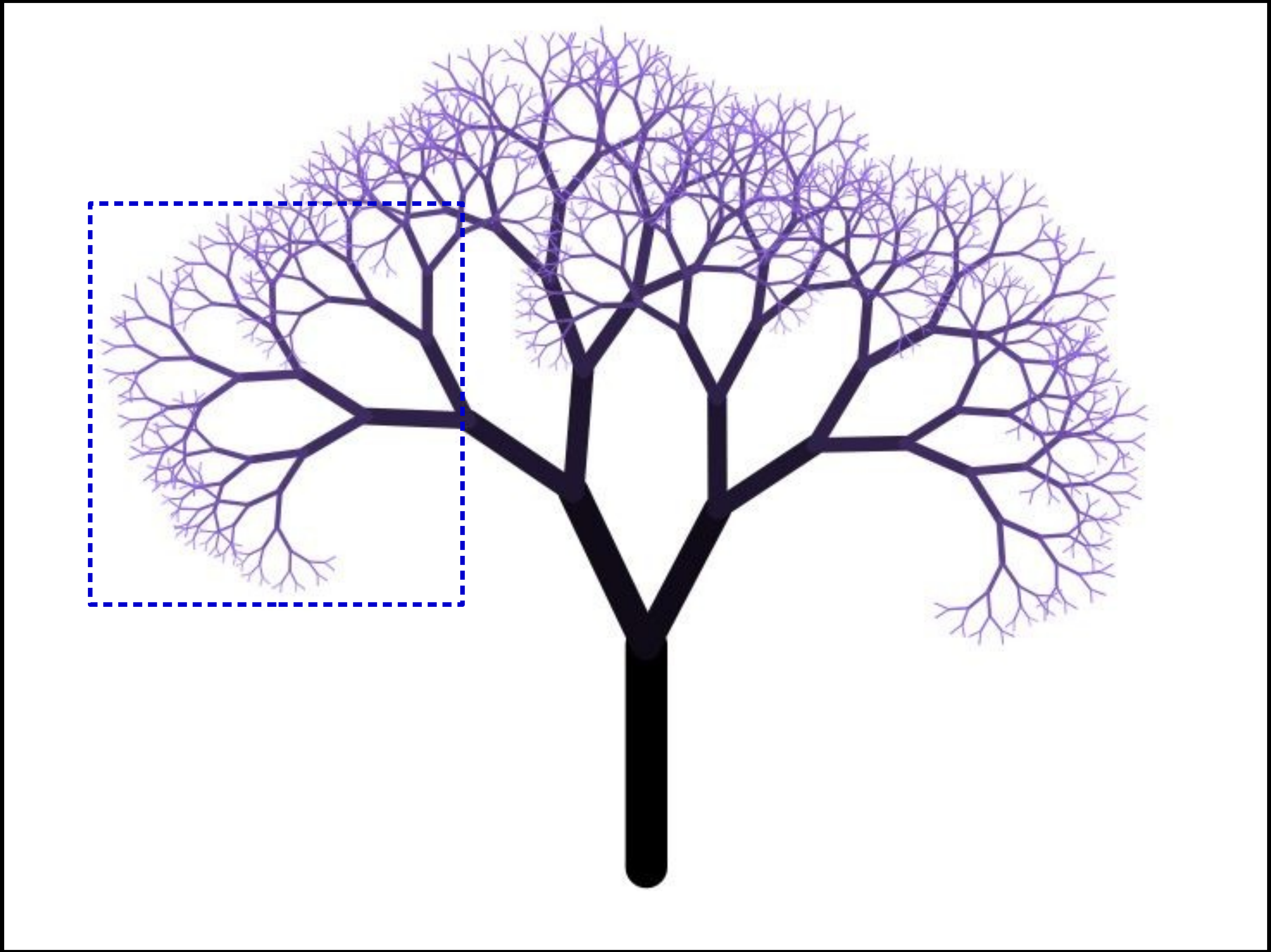
A ***fractal*** is an object that contains a smaller copy of itself.

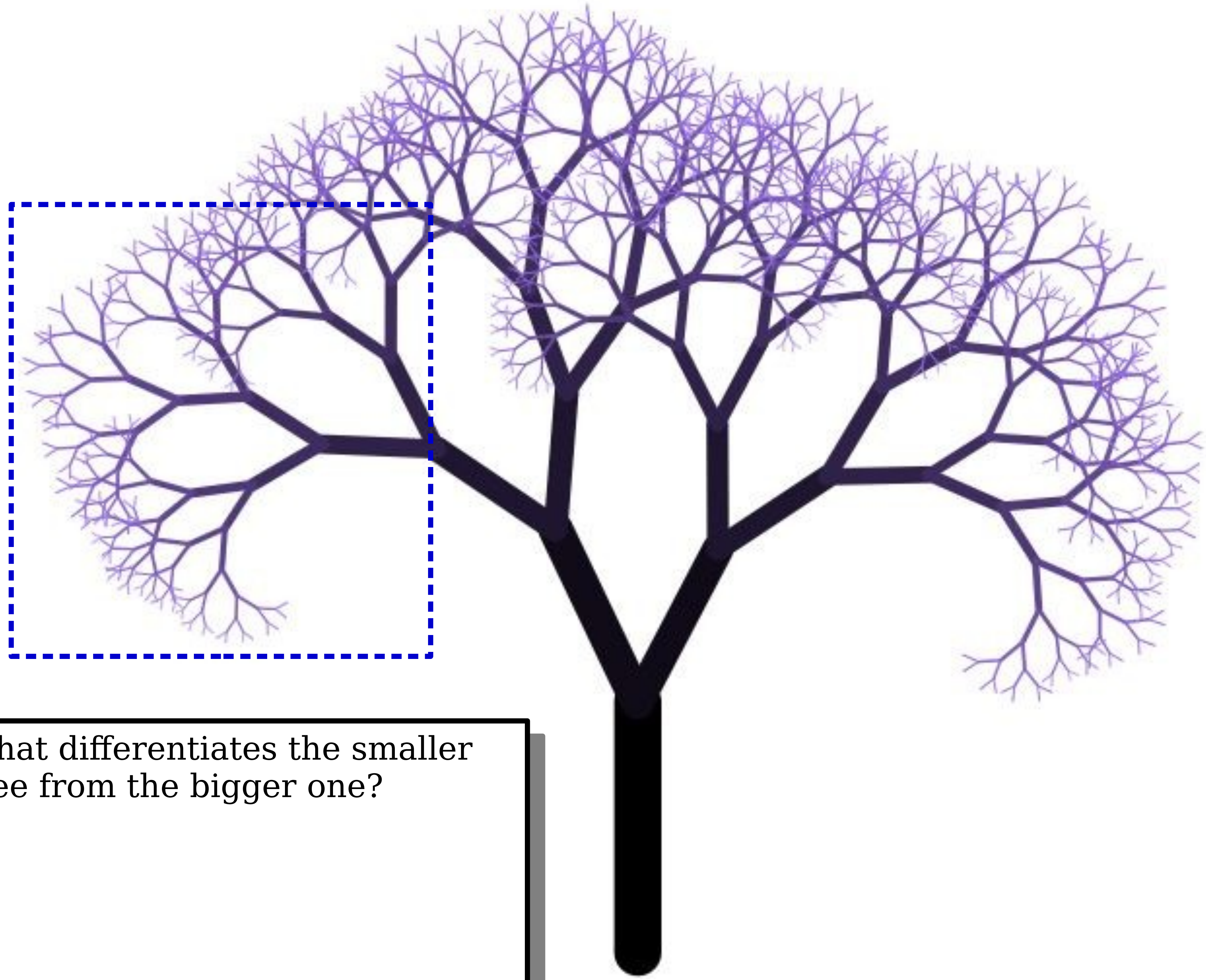
A ***fractal*** is an object that contains a smaller copy of itself.



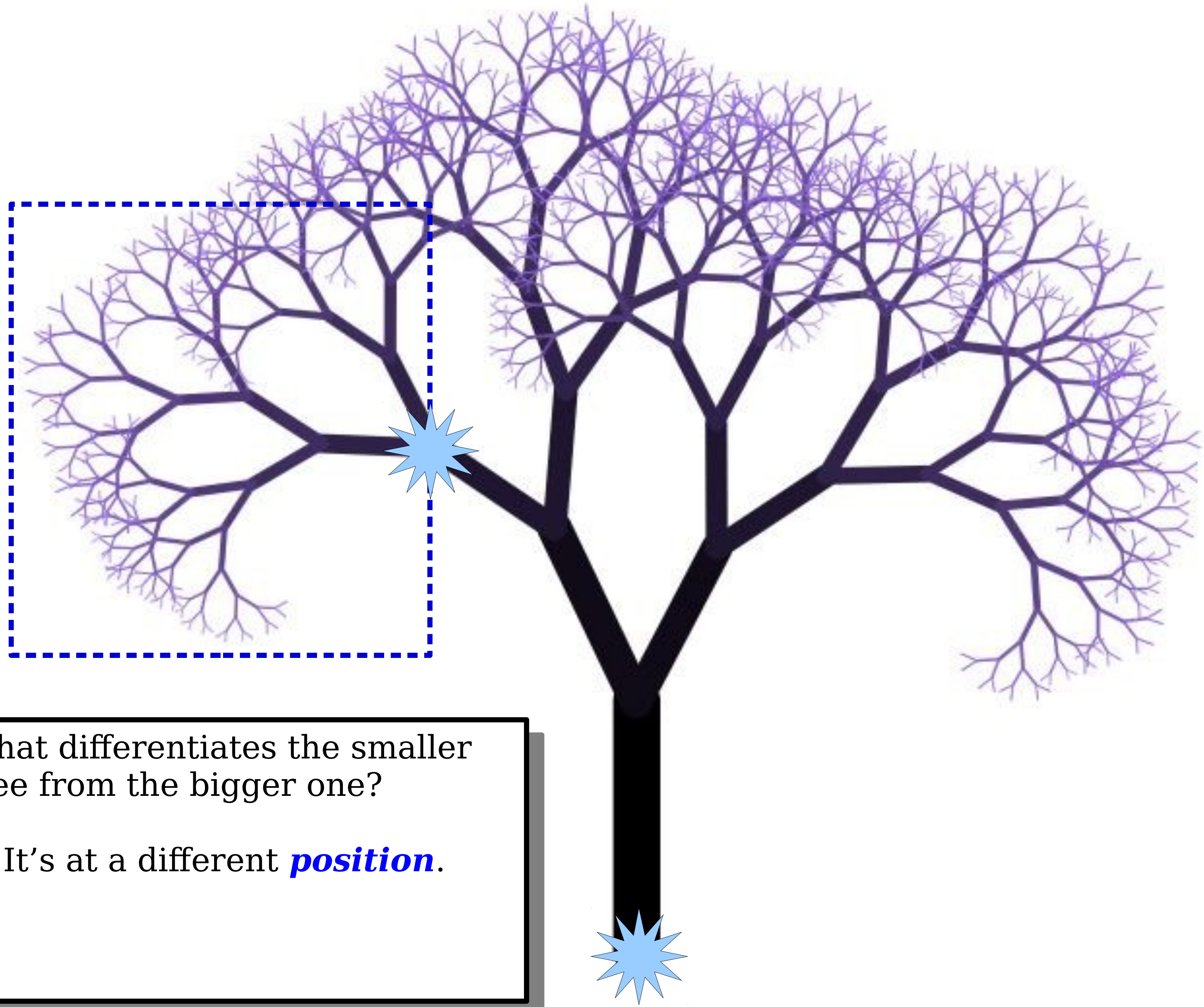






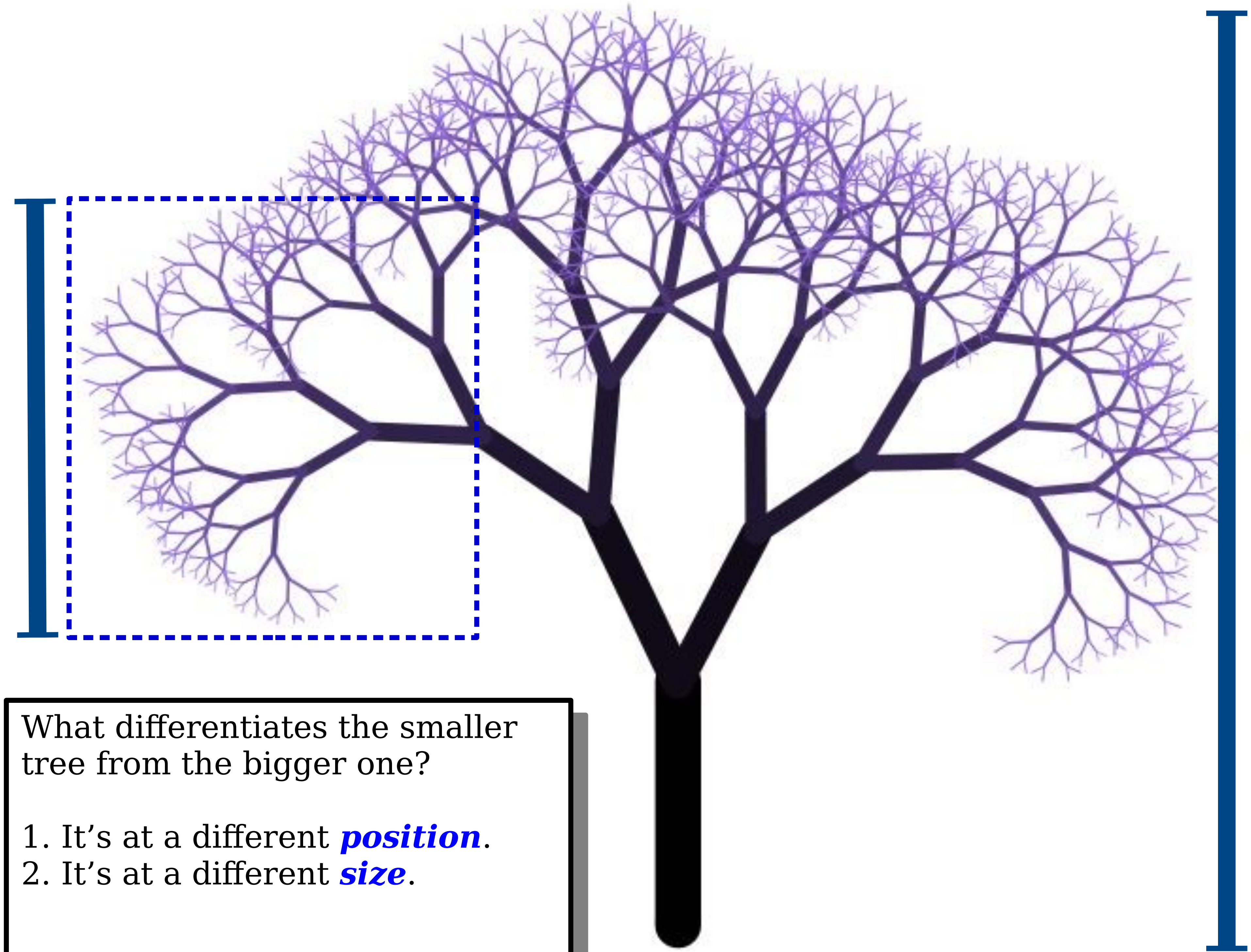


What differentiates the smaller tree from the bigger one?



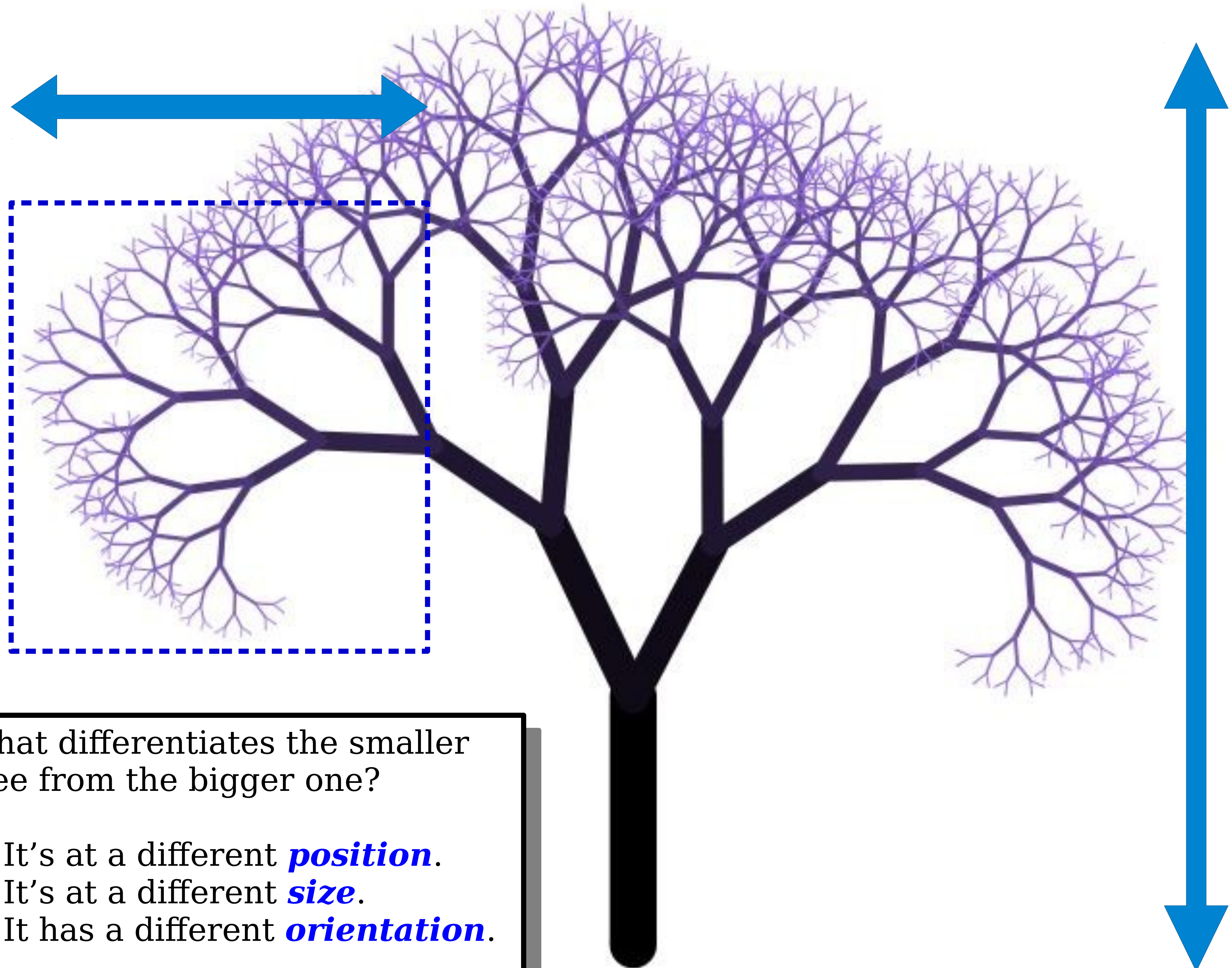
What differentiates the smaller tree from the bigger one?

1. It’s at a different ***position***.



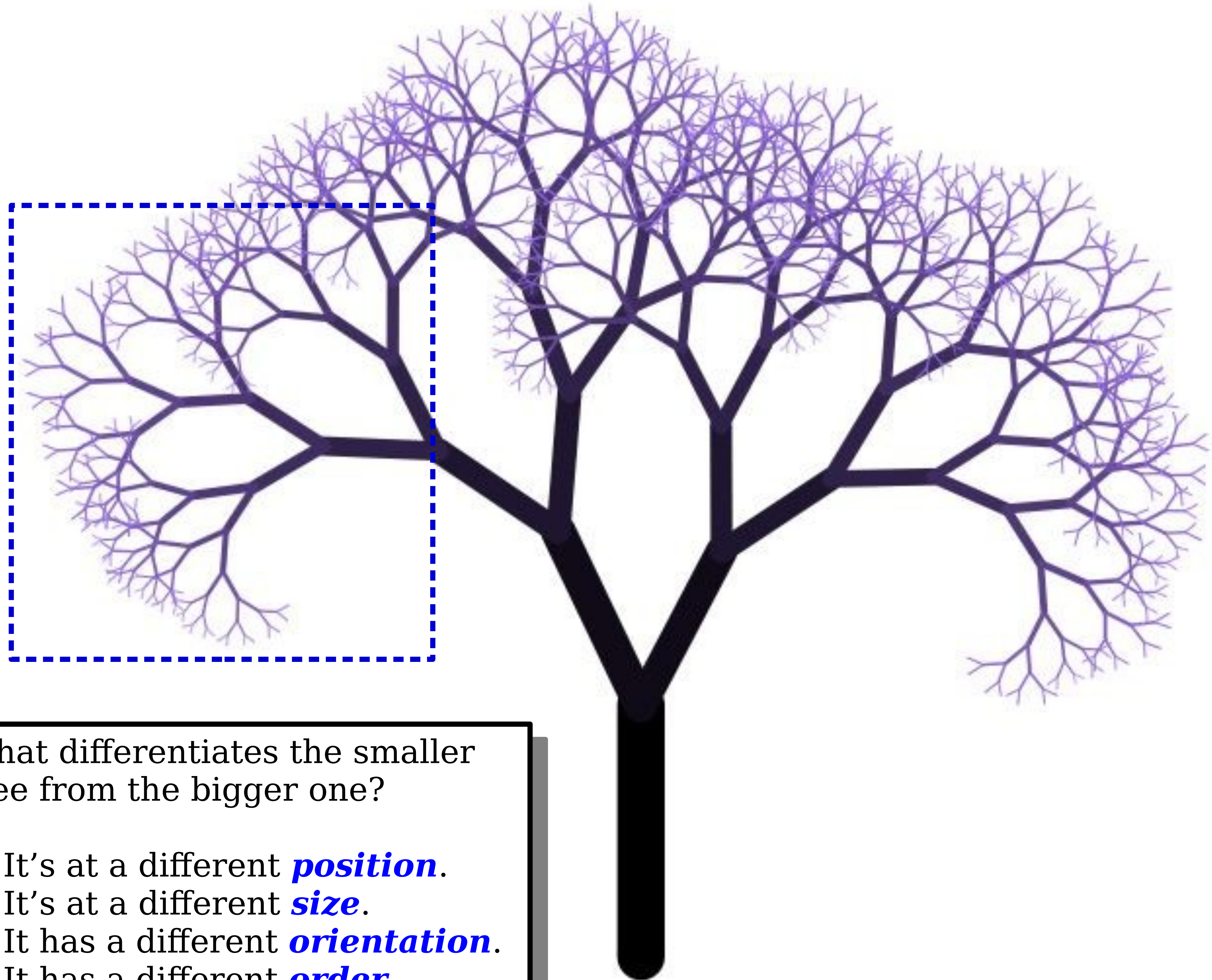
What differentiates the smaller tree from the bigger one?

1. It's at a different ***position***.
2. It's at a different ***size***.



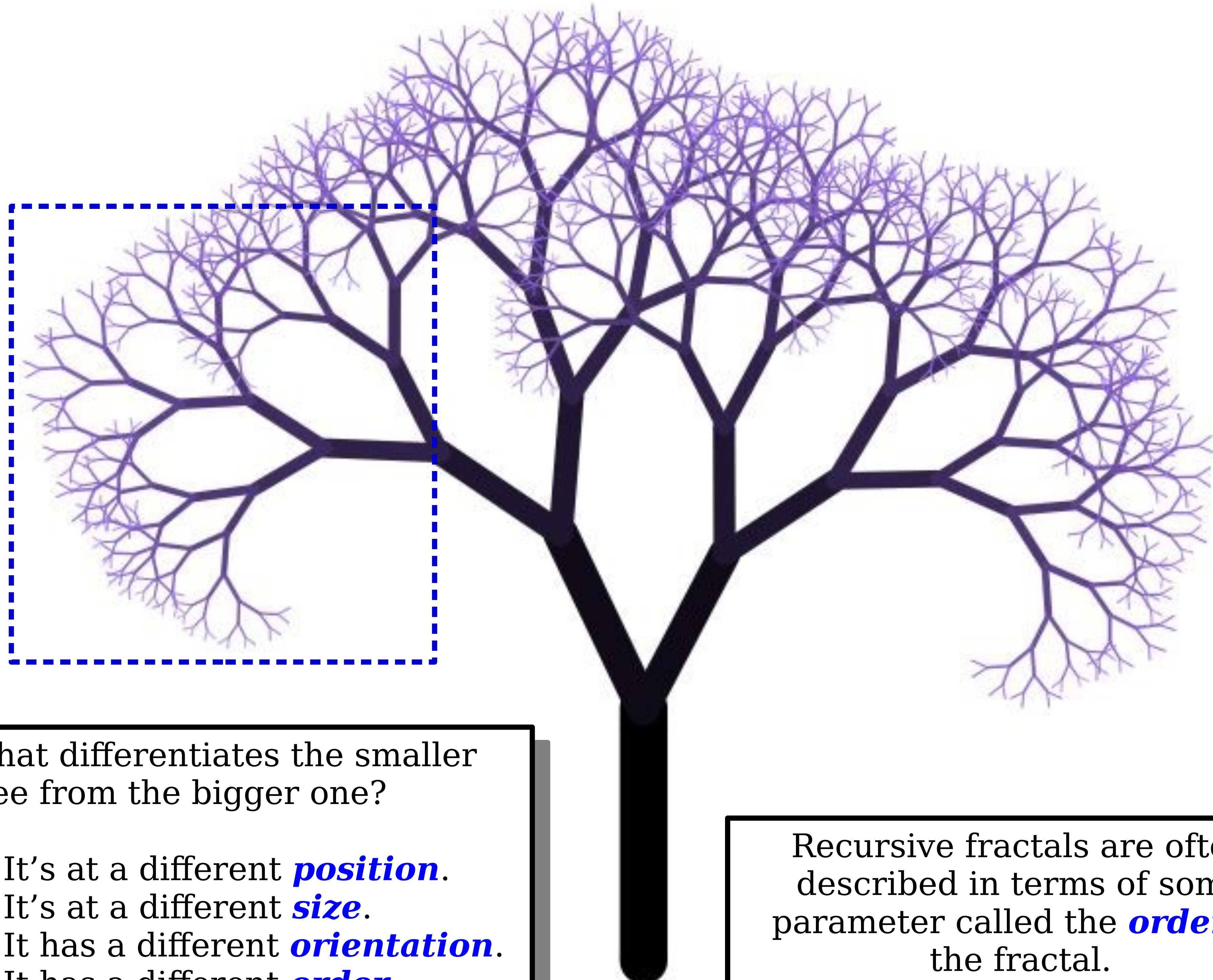
What differentiates the smaller tree from the bigger one?

1. It’s at a different ***position***.
2. It’s at a different ***size***.
3. It has a different ***orientation***.



What differentiates the smaller tree from the bigger one?

1. It’s at a different ***position***.
2. It’s at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.



What differentiates the smaller tree from the bigger one?

1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

Recursive fractals are often described in terms of some parameter called the ***order*** of the fractal.

An order-0 tree.

What differentiates the smaller tree from the bigger one?

1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

Recursive fractals are often described in terms of some parameter called the ***order*** of the fractal.

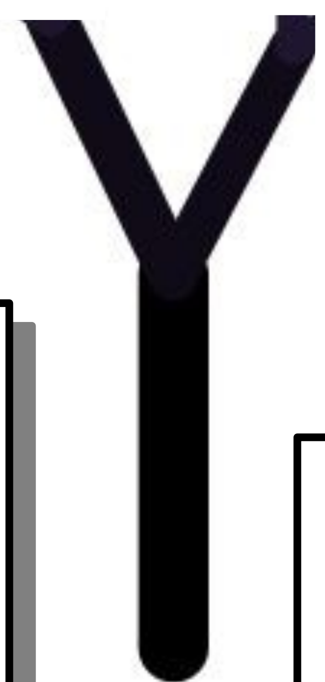
An order-1 tree.

What differentiates the smaller tree from the bigger one?

1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

Recursive fractals are often described in terms of some parameter called the ***order*** of the fractal.

An order-2 tree.

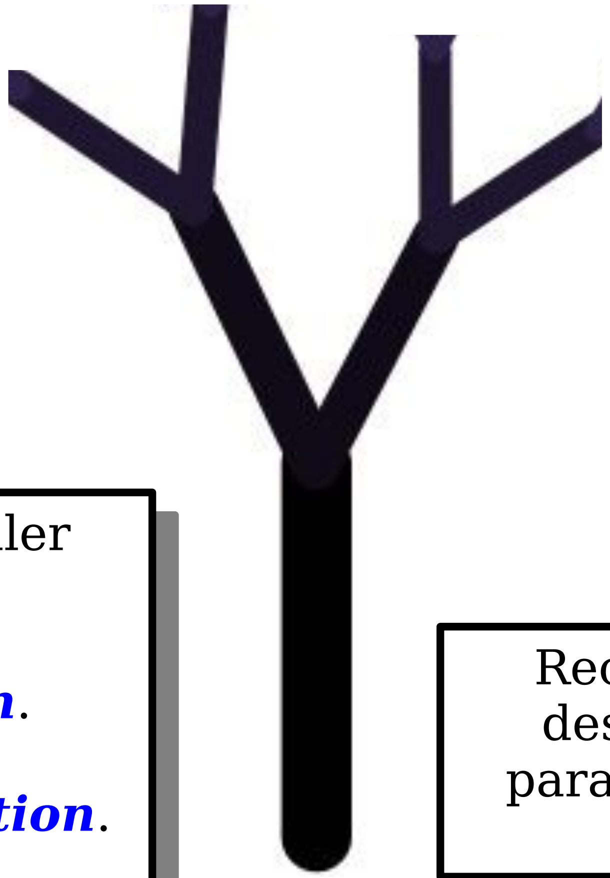


What differentiates the smaller tree from the bigger one?

1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

Recursive fractals are often described in terms of some parameter called the ***order*** of the fractal.

An order-3 tree.

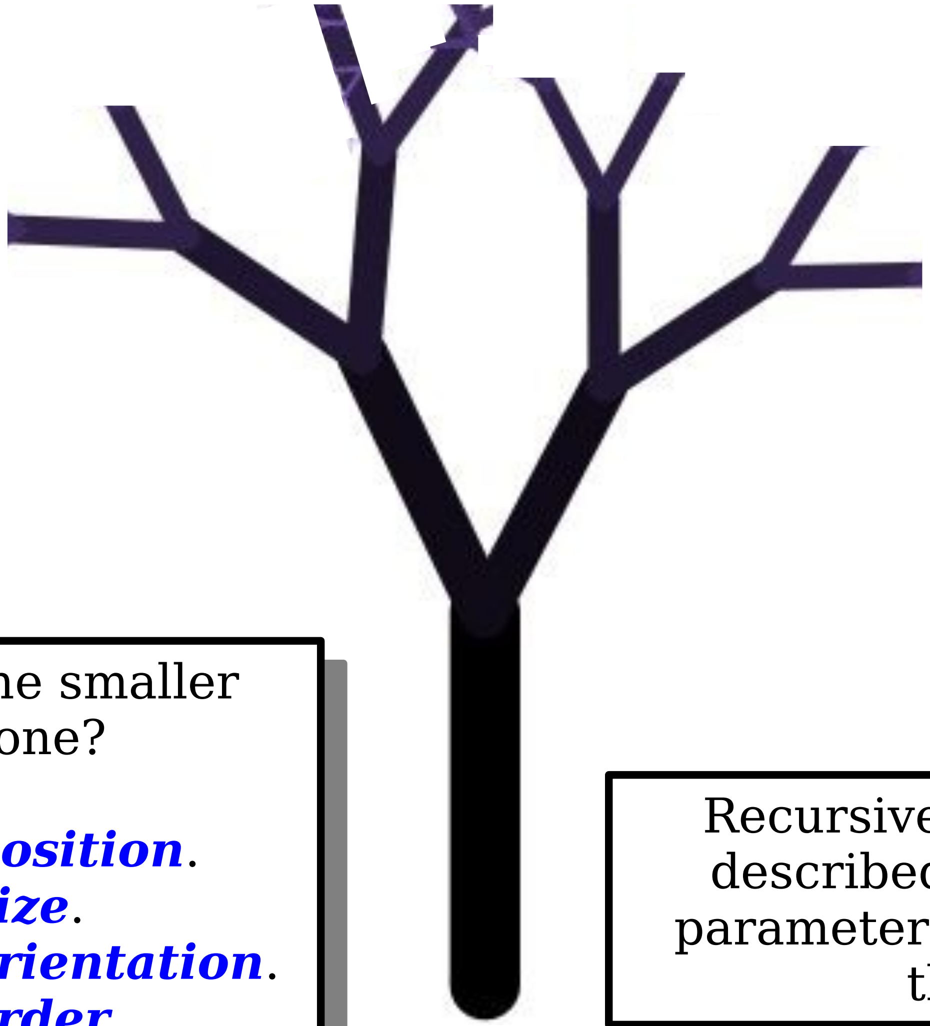


What differentiates the smaller tree from the bigger one?

1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

Recursive fractals are often described in terms of some parameter called the ***order*** of the fractal.

An order-4 tree.

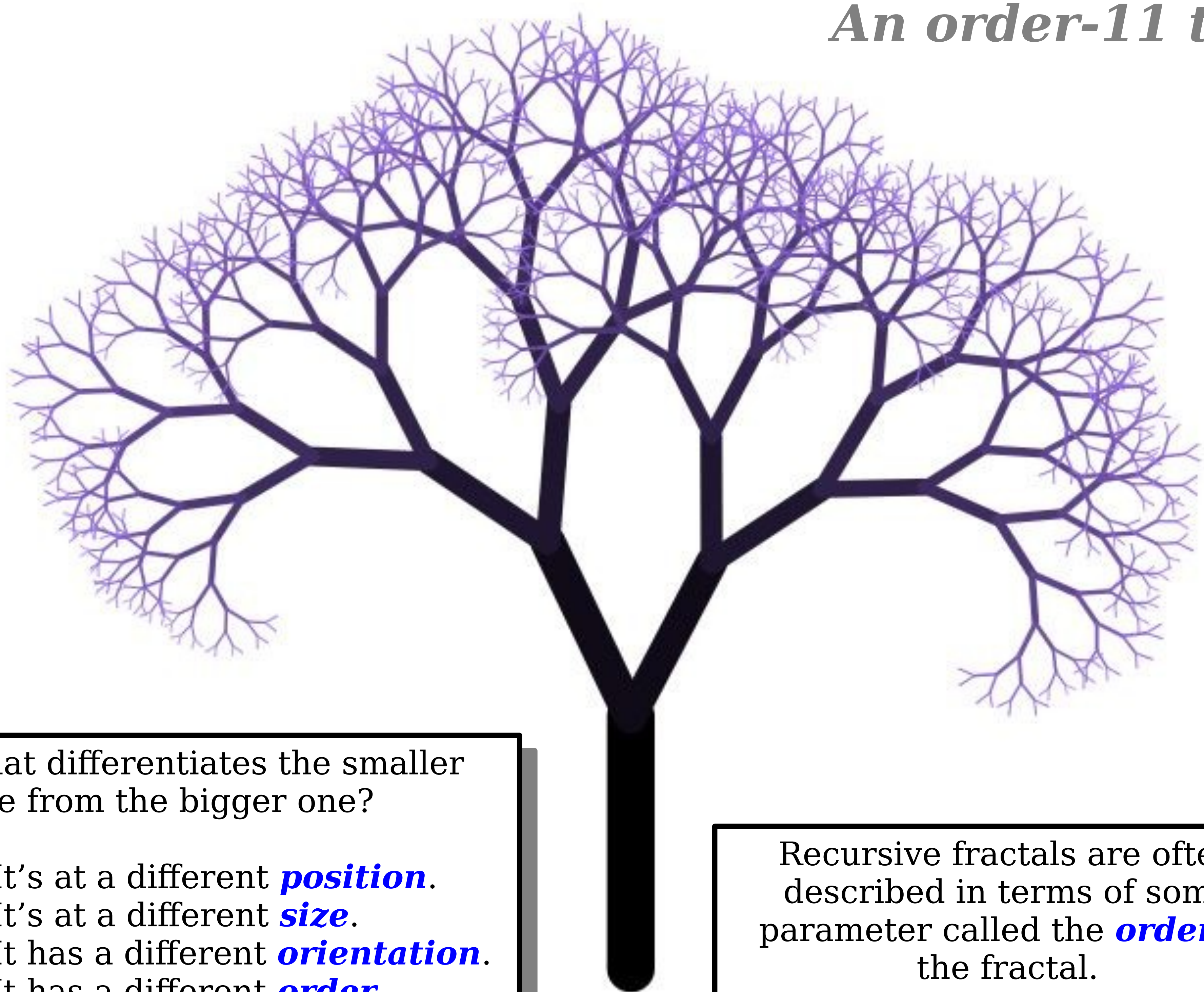


What differentiates the smaller tree from the bigger one?

1. It’s at a different ***position***.
2. It’s at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

Recursive fractals are often described in terms of some parameter called the ***order*** of the fractal.

An order-11 tree.

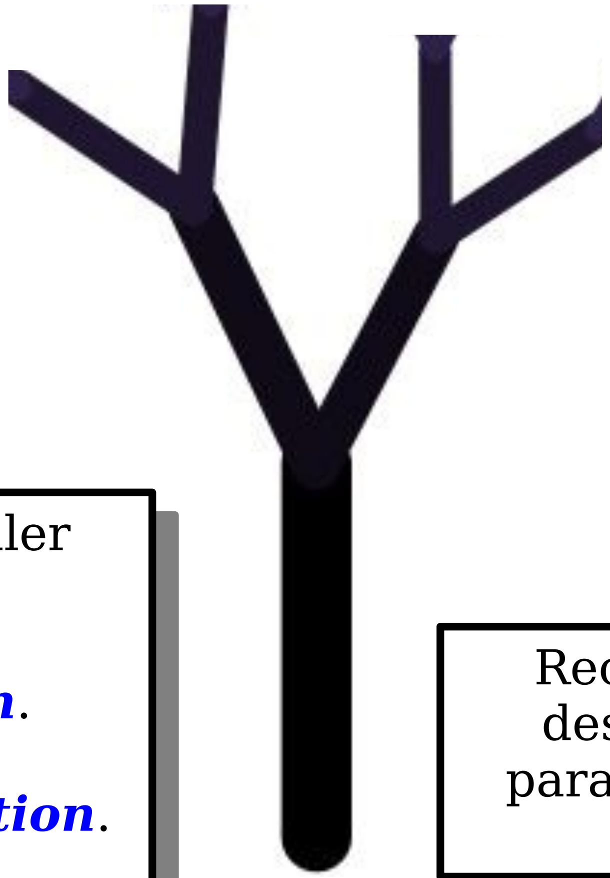


What differentiates the smaller tree from the bigger one?

1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

Recursive fractals are often described in terms of some parameter called the ***order*** of the fractal.

An order-3 tree.



What differentiates the smaller tree from the bigger one?

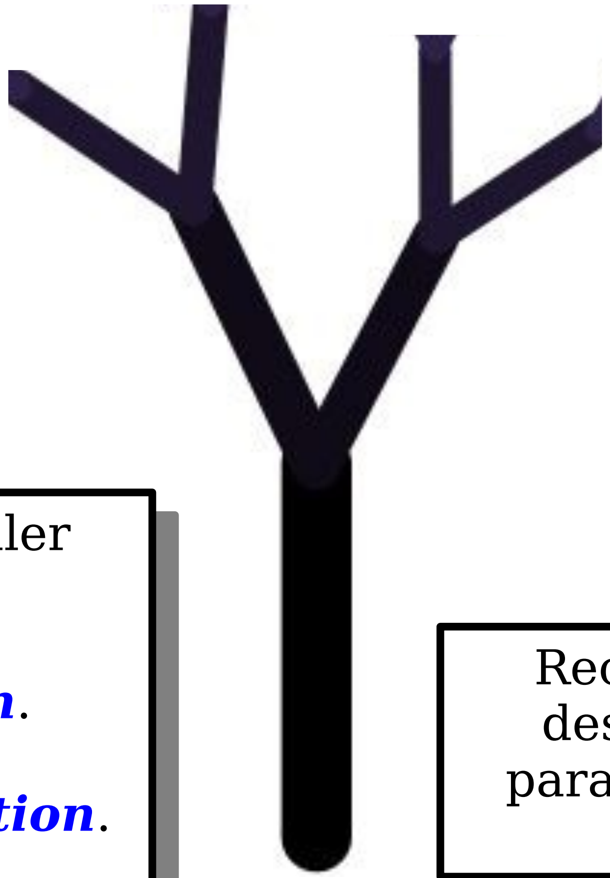
1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

Recursive fractals are often described in terms of some parameter called the ***order*** of the fractal.

An order-3 tree.

An order-0 tree is nothing at all.

An order-*n* tree is a line with two smaller order-(*n*-1) trees starting at the end of that line.



What differentiates the smaller tree from the bigger one?

1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

Recursive fractals are often described in terms of some parameter called the ***order*** of the fractal.

In-Class Task

Write a procedure (steps in english/pseudocode) to generate an order-3 fractal tree

In-Class Task

- draw a line

- tilt the canvas 45˚ left and draw an order-2 tree

- tilt the canvas 45˚ right and draw an order-2 tree

In-Class Task

- draw a line

- tilt the canvas 45˚ left and draw an order-2 tree

- tilt the canvas 45˚ right and draw an order-2 tree

In-Class Task

- draw a line
- tilt the canvas 45° left and draw an order-2 tree
    - draw a line
    - tilt the canvas 45° left and draw an order-1 tree
    - tilt the canvas 45° right and draw an order-1 tree
- tilt the canvas 45° right and draw an order-2 tree
    - draw a line
    - tilt the canvas 45° left and draw an order-1 tree
    - tilt the canvas 45° right and draw an order-1 tree

In-Class Task

- draw a line
- tilt the canvas 45° left and draw an order-2 tree
    - draw a line
    - tilt the canvas 45° left and draw an order-1 tree
    - tilt the canvas 45° right and draw an order-1 tree
- tilt the canvas 45° right and draw an order-2 tree
    - draw a line
    - tilt the canvas 45° left and draw an order-1 tree
    - tilt the canvas 45° right and draw an order-1 tree

In-Class Task

- draw a line
- tilt the canvas 45˚ left and draw an order-2 tree
    - draw a line
    - tilt the canvas 45˚ left and draw an order-1 tree
        - draw a line
        - tilt the canvas 45˚ left and draw an order-0 tree
        - tilt the canvas 45˚ right and draw an order-0 tree
    - tilt the canvas 45˚ right and draw an order-1 tree
        - draw a line
        - tilt the canvas 45˚ left and draw an order-0 tree
        - tilt the canvas 45˚ right and draw an order-0 tree
- tilt the canvas 45˚ right and draw an order-2 tree
    - draw a line
    - tilt the canvas 45˚ left and draw an order-1 tree
        - draw a line
        - tilt the canvas 45˚ left and draw an order-0 tree
        - tilt the canvas 45˚ right and draw an order-0 tree
    - tilt the canvas 45˚ right and draw an order-1 tree
        - draw a line
        - tilt the canvas 45˚ left and draw an order-0 tree
        - tilt the canvas 45˚ right and draw an order-0 tree

In-Class Task

- draw a line
- tilt the canvas 45˚ left and draw an order-2 tree
    - draw a line
    - tilt the canvas 45˚ left and draw an order-1 tree
        - draw a line
        - tilt the canvas 45˚ left and draw an order-0 tree
        - tilt the canvas 45˚ right and draw an order-0 tree
    - tilt the canvas 45˚ right and draw an order-1 tree
        - draw a line
        - tilt the canvas 45˚ left and draw an order-0 tree
        - tilt the canvas 45˚ right and draw an order-0 tree
- tilt the canvas 45˚ right and draw an order-2 tree
    - draw a line
    - tilt the canvas 45˚ left and draw an order-1 tree
        - draw a line
        - tilt the canvas 45˚ left and draw an order-0 tree
        - tilt the canvas 45˚ right and draw an order-0 tree
    - tilt the canvas 45˚ right and draw an order-1 tree
        - draw a line
        - tilt the canvas 45˚ left and draw an order-0 tree
        - tilt the canvas 45˚ right and draw an order-0 tree

Nothing to draw at order 0
We stop!

BASE CASE

In general for n

- draw a line

- tilt the canvas 45° left and draw and order-(n-1) tree

- tilt the canvas 45° right and draw and order-(n-1) tree

Check This Out!!!

http://recursivedrawing.com/

Different Flavors of Recursion

Reverse String: write first character, reverse the remaining single smaller string

Dictionary: either inspect upper-half or lower-half

Fractal Tree: draw both the left order-(n-1) and right order-(n-1) trees

All solve a problem by breaking it up into one or more smaller "*similar*" problems

Recursive Problem-Solving

```
if(problem is sufficiently simple){

     directly solve the problem
     i.e. do something and/or return the solution

} else{

     split problem up into one or more smaller
     problems with the same structure as the original

     solve some or all of those smaller problems

     combine results to get overall solution

     return overall solution
}
```

Recursive Problem-Solving

```
if(problem is sufficiently simple){

     directly solve the problem
     i.e. do something and/or return the solution

} else{

     split problem up into one or more smaller
     problems with the same structure as the original

     solve some or all of those smaller problems

     combine results to get overall solution

     return overall solution
}
```

BASE CASE

Why Recursion

An alternative to iteration

Not always practical

Elegant and intuitive solution for some problems

Factorial

1 x 2 x 3 x … x n

$$n! = \prod_{k=1}^{n} k$$

For example:

0!=1, **1!**=1, **2!**=2, **3!**=6, **4!**=24, **5!**=120

The empty product

But what if we start from `n`?

n!=

But what if we start from `n`?

n! = n x (n-1) x (n-2) x (n - 3) x … …. …. … 2 x 1

What is this?

But what if we start from `n`?

$$\mathbf{n!} = n \times \underbrace{(n-1) \times (n-2) \times (n-3) \times \ldots \quad \ldots. \quad \ldots. \quad \ldots 2 \times 1}_{\mathbf{(n-1)!}}$$

But what if we start from `n`?

$$\mathbf{n!} = n \times \underbrace{(n-1) \times (n-2) \times (n - 3) \times \ldots \quad \ldots. \quad \ldots. \quad \ldots\ 2 \times 1}_{\mathbf{(n-1)!}}$$

$$\mathbf{(n-1)!} = (n-1) \times \underbrace{(n-2) \times (n - 3) \times \ldots \quad \ldots. \quad \ldots. \quad \ldots\ 2 \times 1}_{\textbf{What is this?}}$$

But what if we start from `n`?

$$\textbf{n!} = n \times \underbrace{(n-1) \times (n-2) \times (n-3) \times \ldots \quad \ldots. \quad \ldots. \quad \ldots 2 \times 1}_{\textbf{(n-1)!}}$$

$$\textbf{(n-1)!} = (n-1) \times \underbrace{(n-2) \times (n-3) \times \ldots \quad \ldots. \quad \ldots. \quad \ldots 2 \times 1}_{\textbf{(n-2)!}}$$

Recursion that Returns a Value

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

Same function being called within solution

Recursion that Returns a Value

n! = n x (n-1)!

```
/** Computes the factorial of the nonnegative integer n.
 pre:  n must be greater than or equal to 0.
 post:  None.
 return:  The factorial of n; n is unchanged. */
int factorial(int n)
{
   if (n == 0)
      return 1;
   else // n > 0, so n-1 >= 0. Thus, fact(n-1) returns (n-1)!
      return n * factorial(n - 1); // n * (n-1)! is n!
}  // end fact
```

Recursion that Returns a Value

n! = n x (n-1)!

```
/** Computes the factorial of the nonnegative integer n.
 pre:  n must be greater than or equal to 0.
 post:  None.
 return:  The factorial of n; n is unchanged. */
int factorial(int n)
{
   if (n == 0)
      return 1;
   else // n > 0, so n-1 >= 0. Thus, fact(n-1) returns (n-1)!
      return n * factorial(n - 1); // n * (n-1)! is n!
}  // end fact
```

BASE CASE

Recursion that Returns a Value

n! = n x (n-1)!

```
/** Computes the factorial of the nonnegative integer n.
 pre:  n must be greater than or equal to 0.
 post:  None.
 return:  The factorial of n; n is unchanged. */
int factorial(int n)
{
   if (n == 0)
      return 1;
   else // n > 0, so n-1 >= 0. Thus, fact(n-1) returns (n-1)!
      return n * factorial(n - 1); // n * (n-1)! is n!
}  // end fact
```

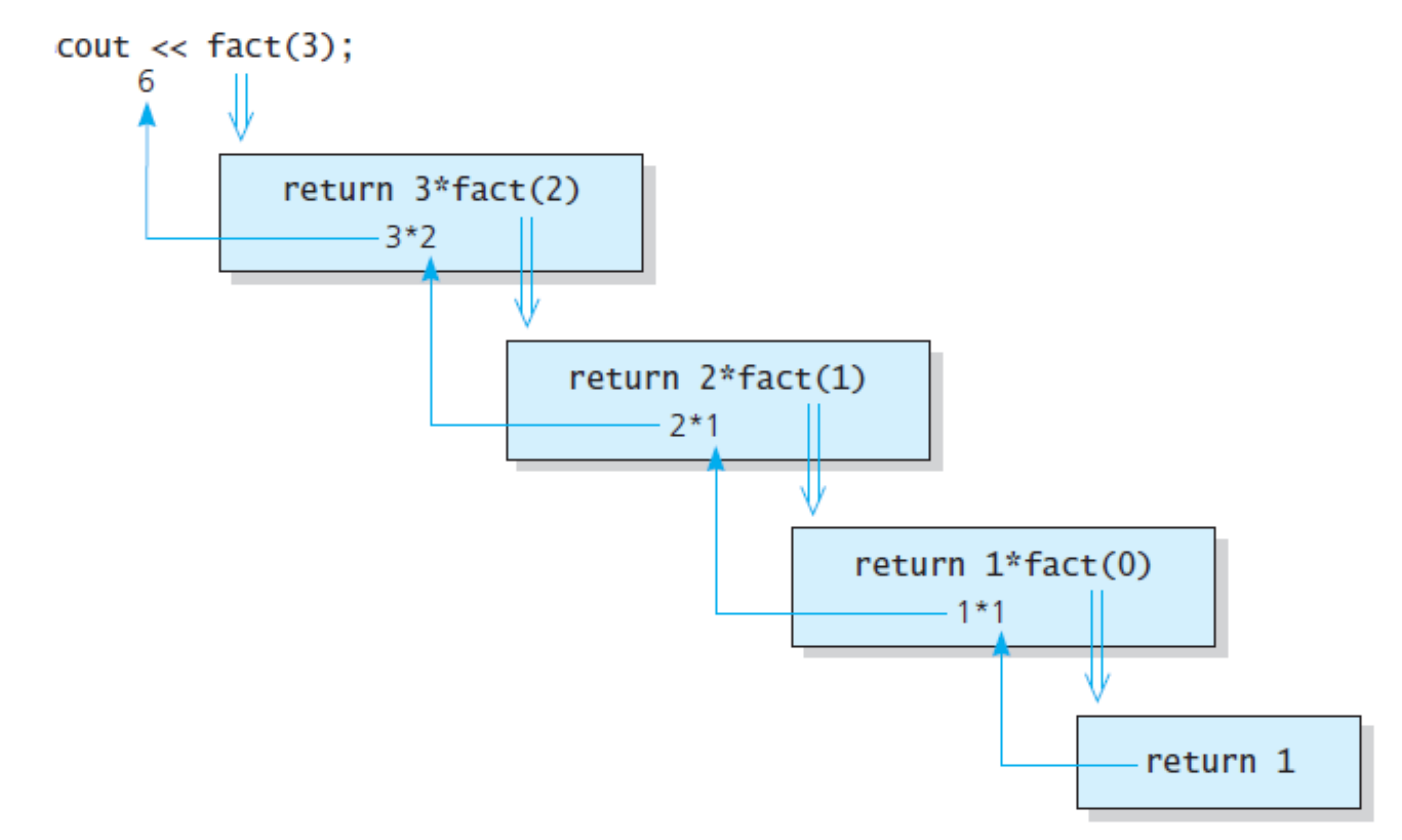
Recursion that Returns a Value

n! = n x (n-1)!

```
/** Computes the factorial of the nonnegative integer n.
 pre:  n must be greater than or equal to 0.
 post:  None.
 return:  The factorial of n; n is unchanged. */
int factorial(int n)
{
   if (n == 0)
      return 1;
   else // n > 0, so n-1 >= 0. Thus, fact(n-1) returns (n-1)!
      return n * factorial(n - 1); // n * (n-1)! is n!
}  // end fact
```

BASE CASE

WILL LEAD TO BASE CASE


cout << fact(3);
6
return 3*fact(2)
3*2
return 2*fact(1)
2*1
return 1*fact(0)
1*1
return 1

Writing a String Backwards

```
writeBackward(string s)
{
   if(the string is empty)
      Do nothing - this is the base case
   else
      Write the last character of s
      writeBackward(s minus the last char)
}
```

Recursion that Performs an Action

```
/** Writes a character string backward.
 pre:  The string s to write backward.
 post:  None.
 param: s  The string to write backward. */

void writeBackward(std::string s)
{
   size_t length = s.size(); // Length of string
   if (length > 0)
   {
      // Write the last character
      std::cout << s.substr(length - 1, 1);


      // Write the rest of the string backward
      writeBackward(s.substr(0, length - 1));
   } // end if

     // length == 0 is the base case - do nothing
} // end writeBackward
```

Recursion that Performs an Action

```
/** Writes a character string backward.
 pre:  The string s to write backward.
 post:  None.
 param: s  The string to write backward. */

void writeBackward(std::string s)
{
   size_t length = s.size(); // Length of string
   if (length > 0)
   {
      // Write the last character
      std::cout << s.substr(length - 1, 1);


      // Write the rest of the string backward
      writeBackward(s.substr(0, length - 1));
   } // end if

     // length == 0 is the base case - do nothing
} // end writeBackward
```

WILL LEAD TO BASE CASE

Write String Backwards

Hello

o

Hell

o l

Hel

o l l

He

o l l e

H

o l l e H

BASE CASE