

Thinking Recursively

Part I

Outline for Today

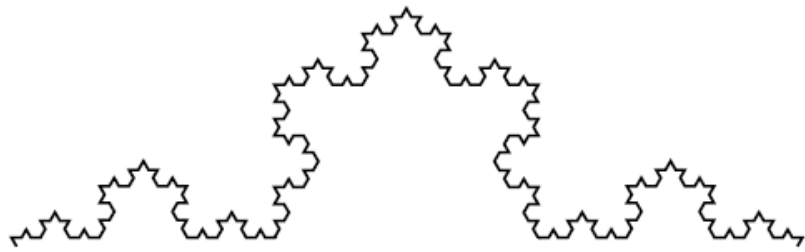
- ***Self-Similarity***
 - Recursive patterns are everywhere!
- ***Recursive Trees***
 - Elegant structures from simple code.
- ***Information Flow***
 - How to send information around in recursion.

Self-Similarity



An object is *self-similar* if it contains a smaller copy of itself.

This is called the
Koch curve.

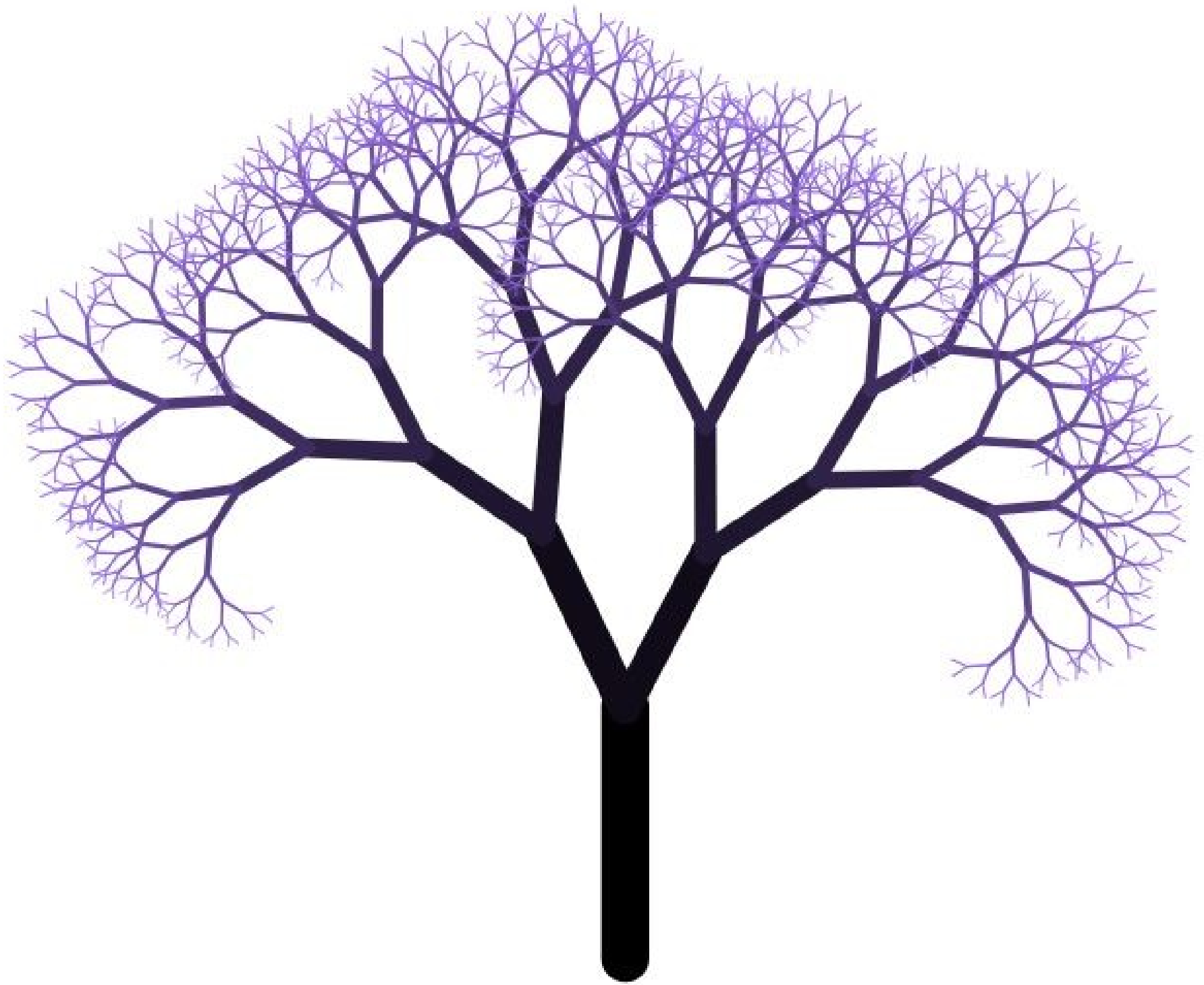


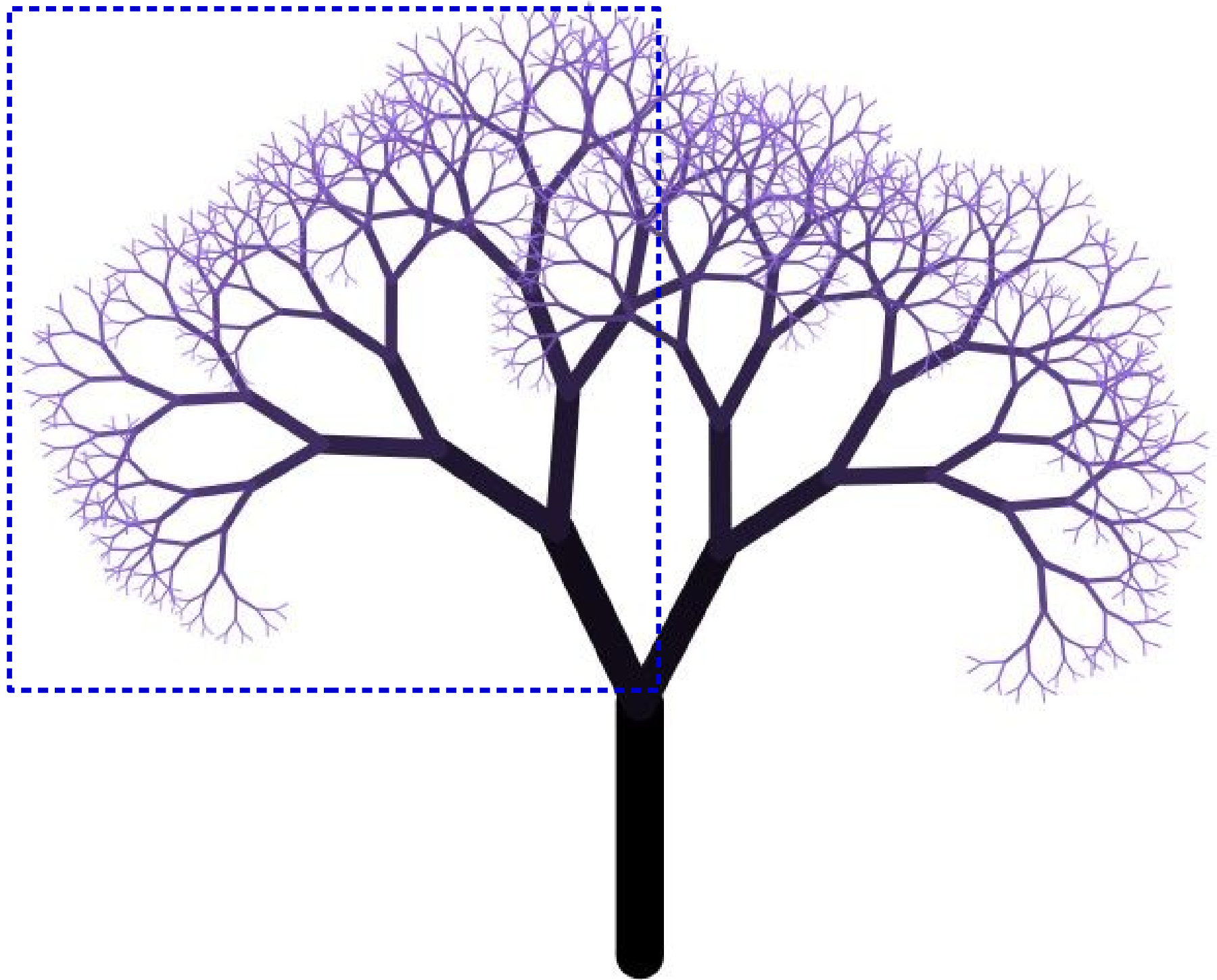
happy
holidays
from
wics

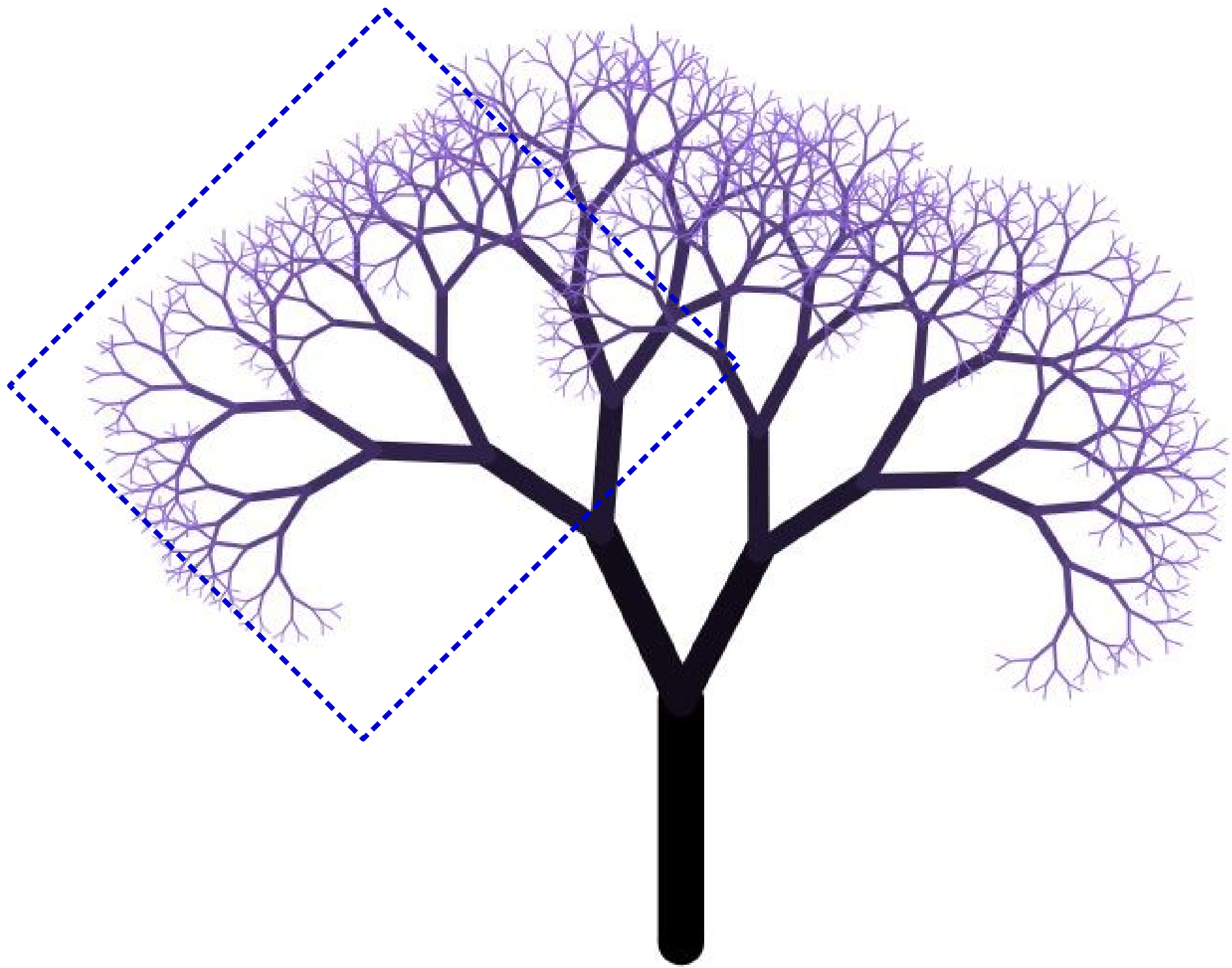
{w}

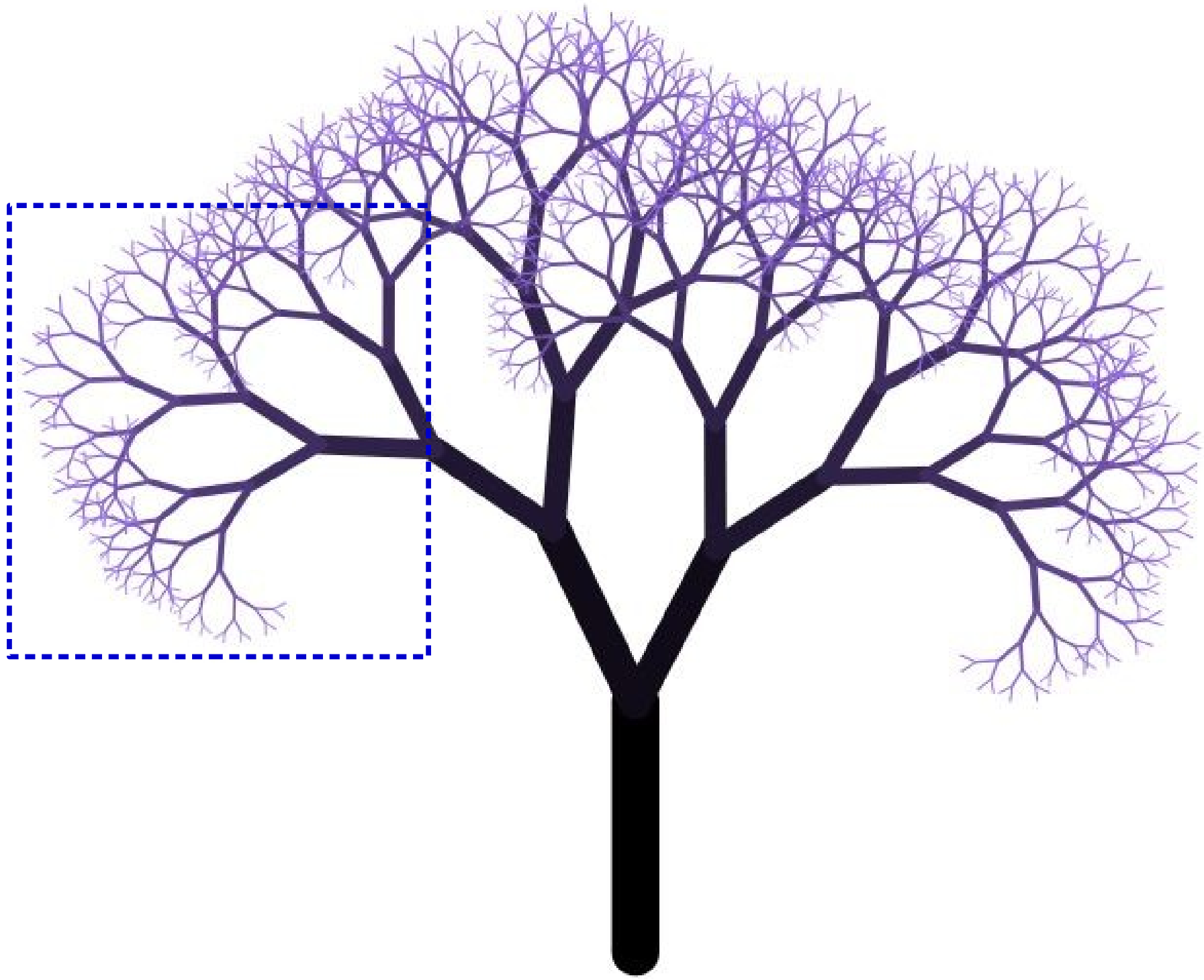
An object is ***self-similar*** if it contains a smaller copy of itself.

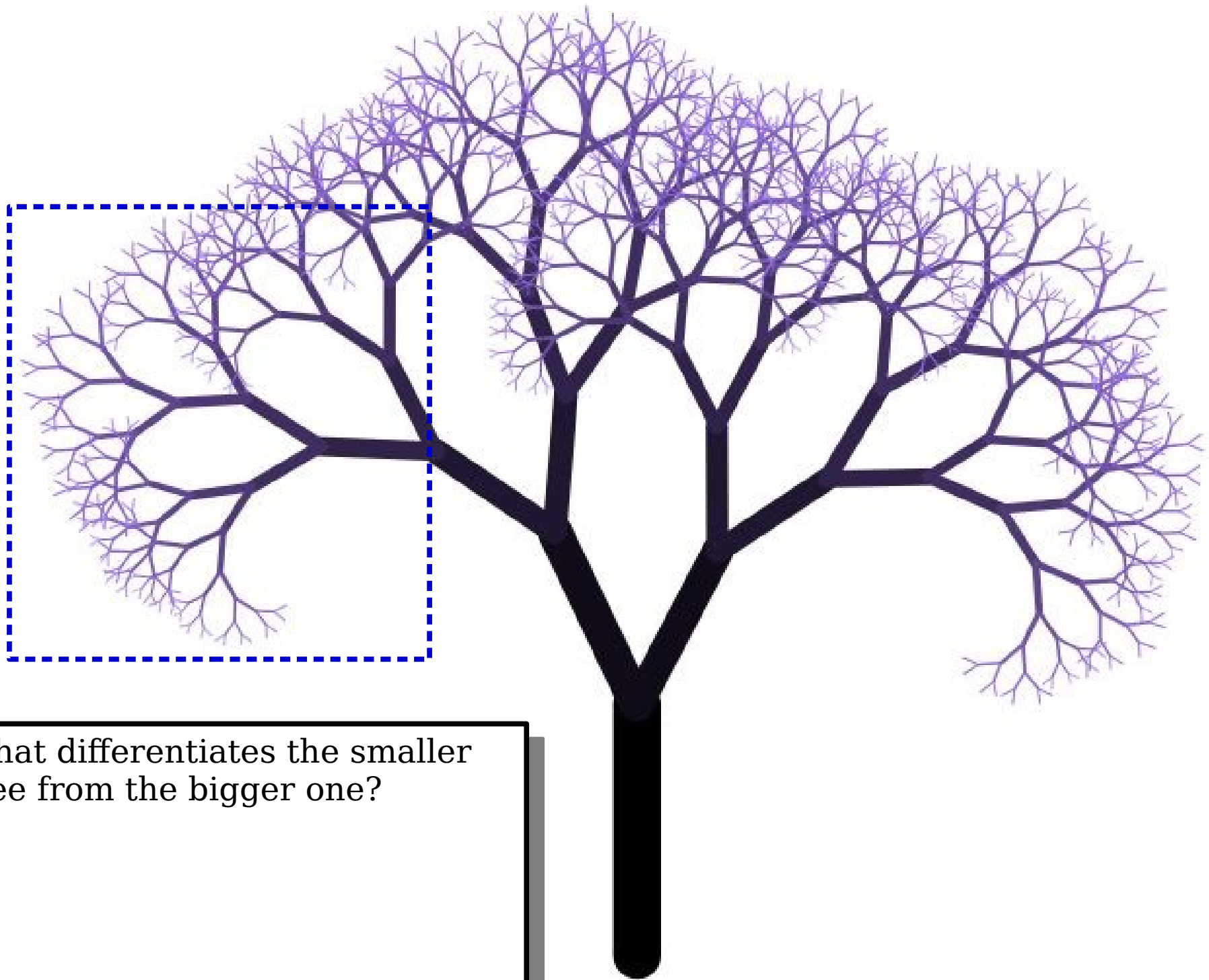
Drawing Self-Similar Shapes



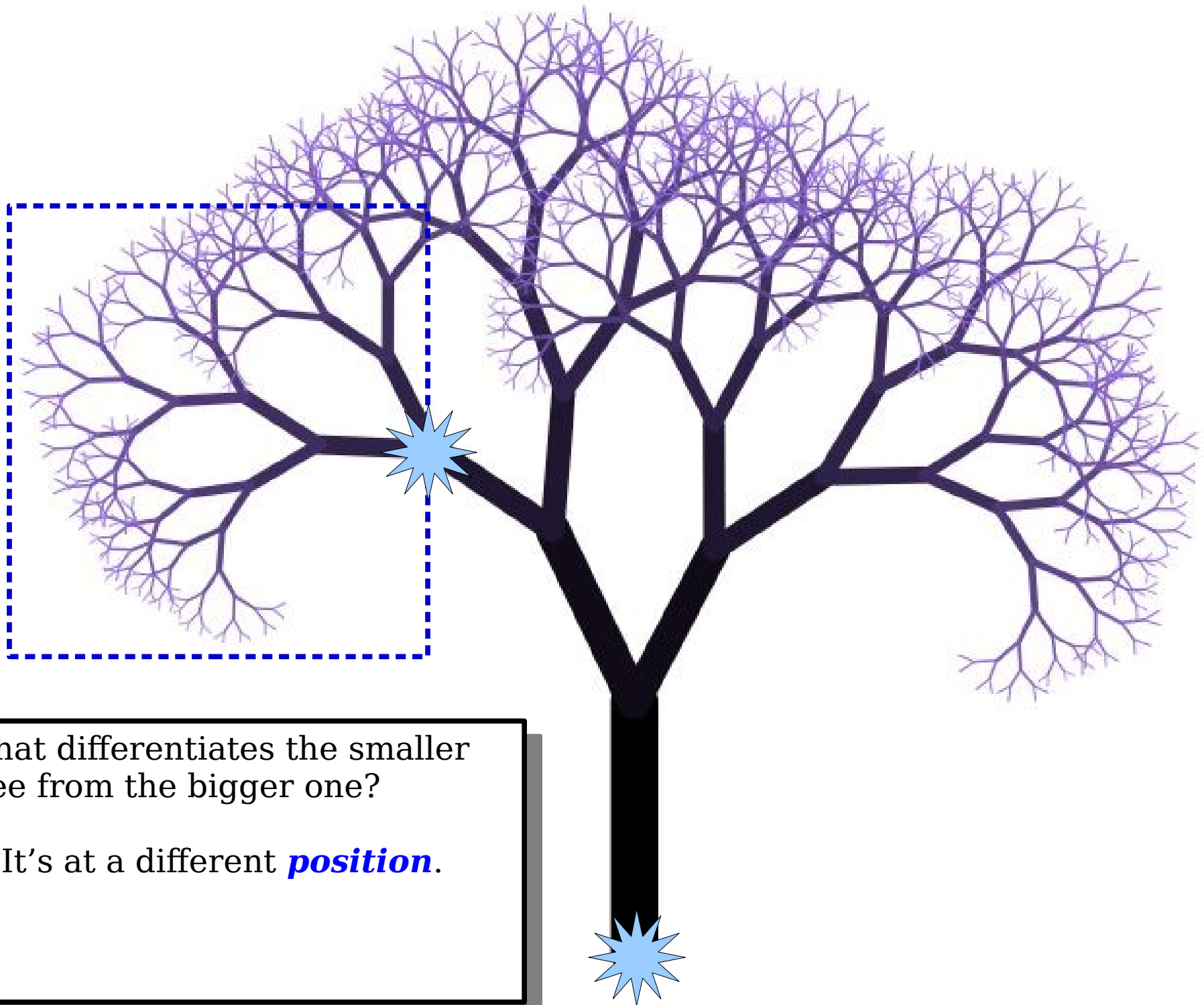






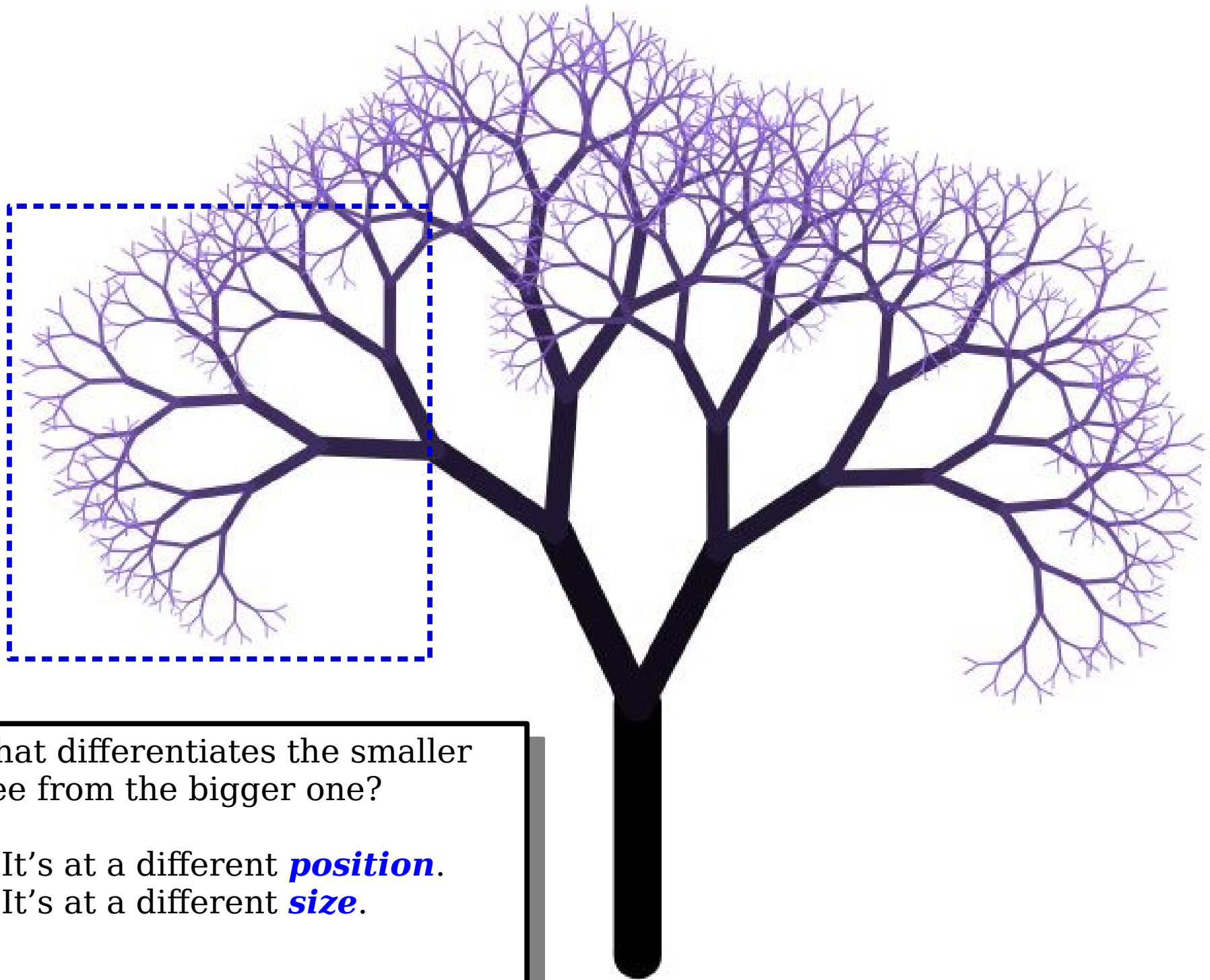


What differentiates the smaller tree from the bigger one?



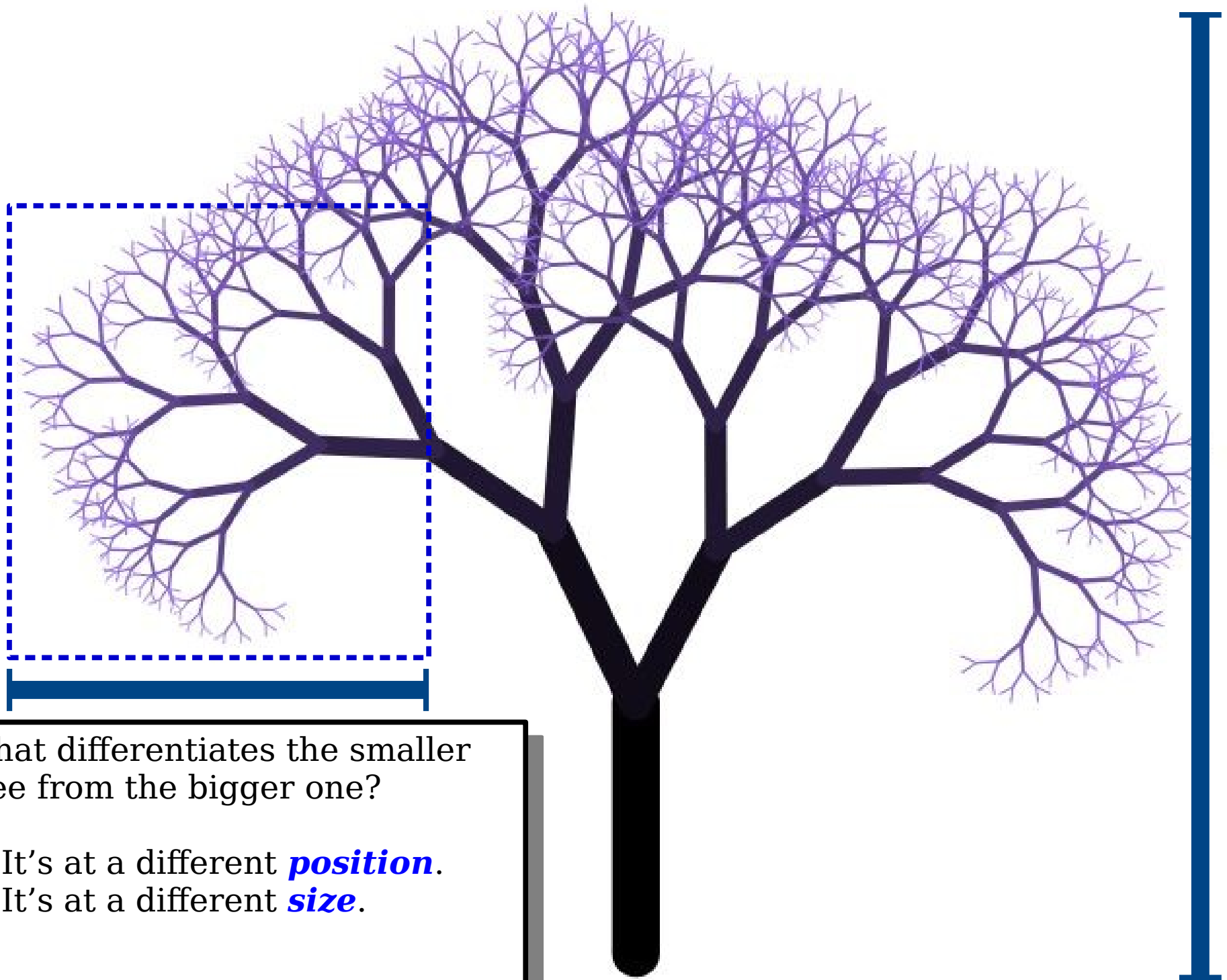
What differentiates the smaller tree from the bigger one?

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



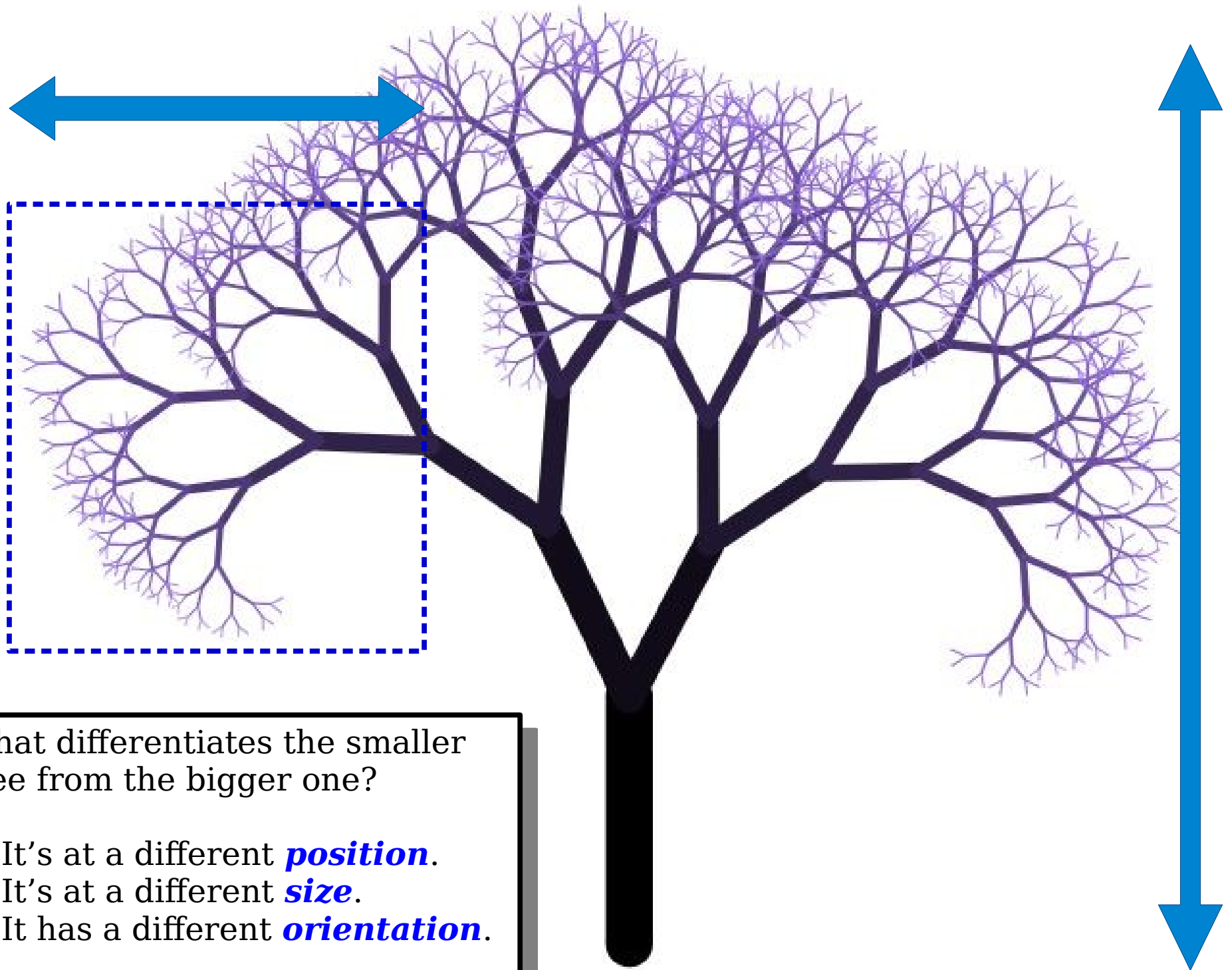
What differentiates the smaller tree from the bigger one?

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



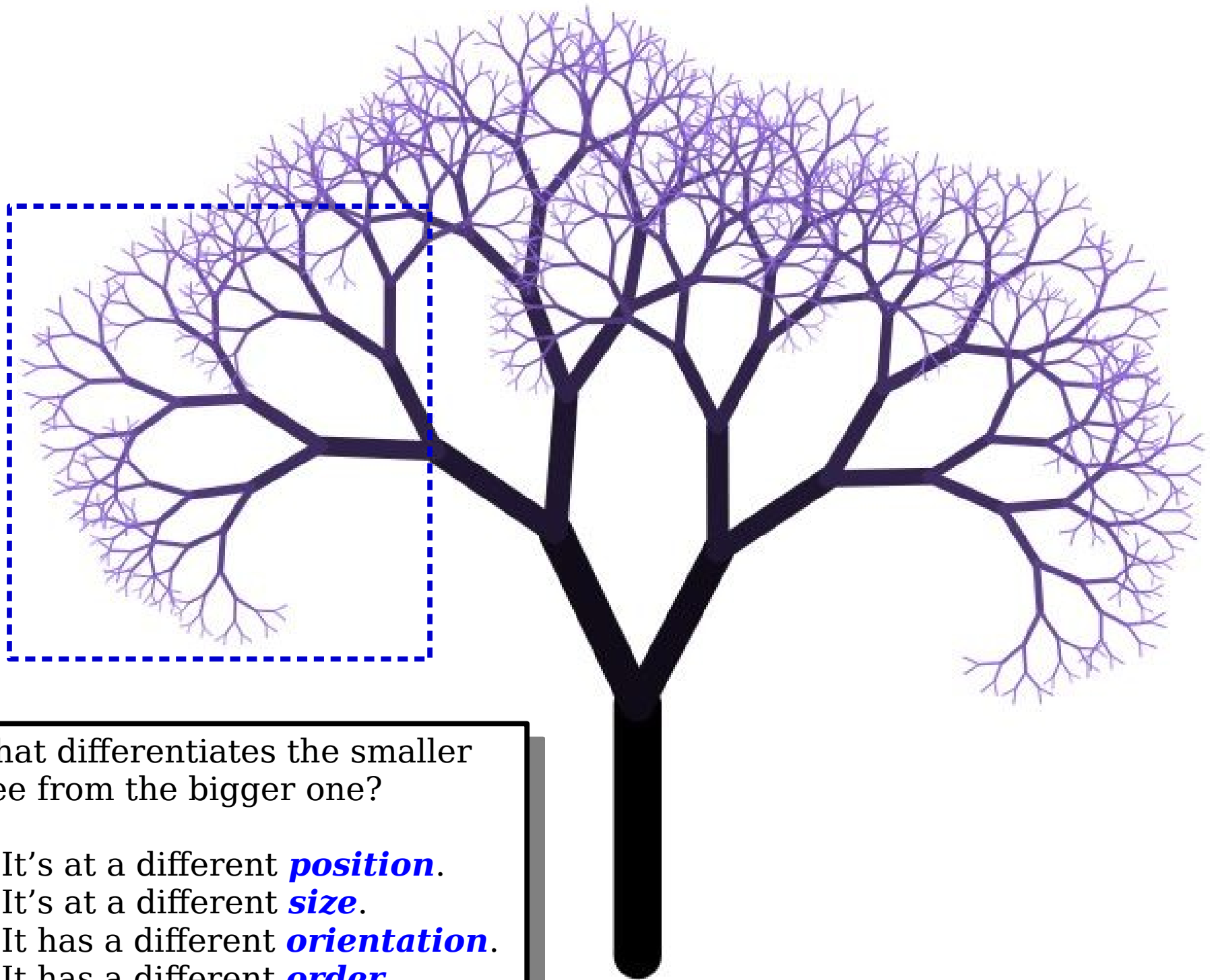
What differentiates the smaller tree from the bigger one?

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



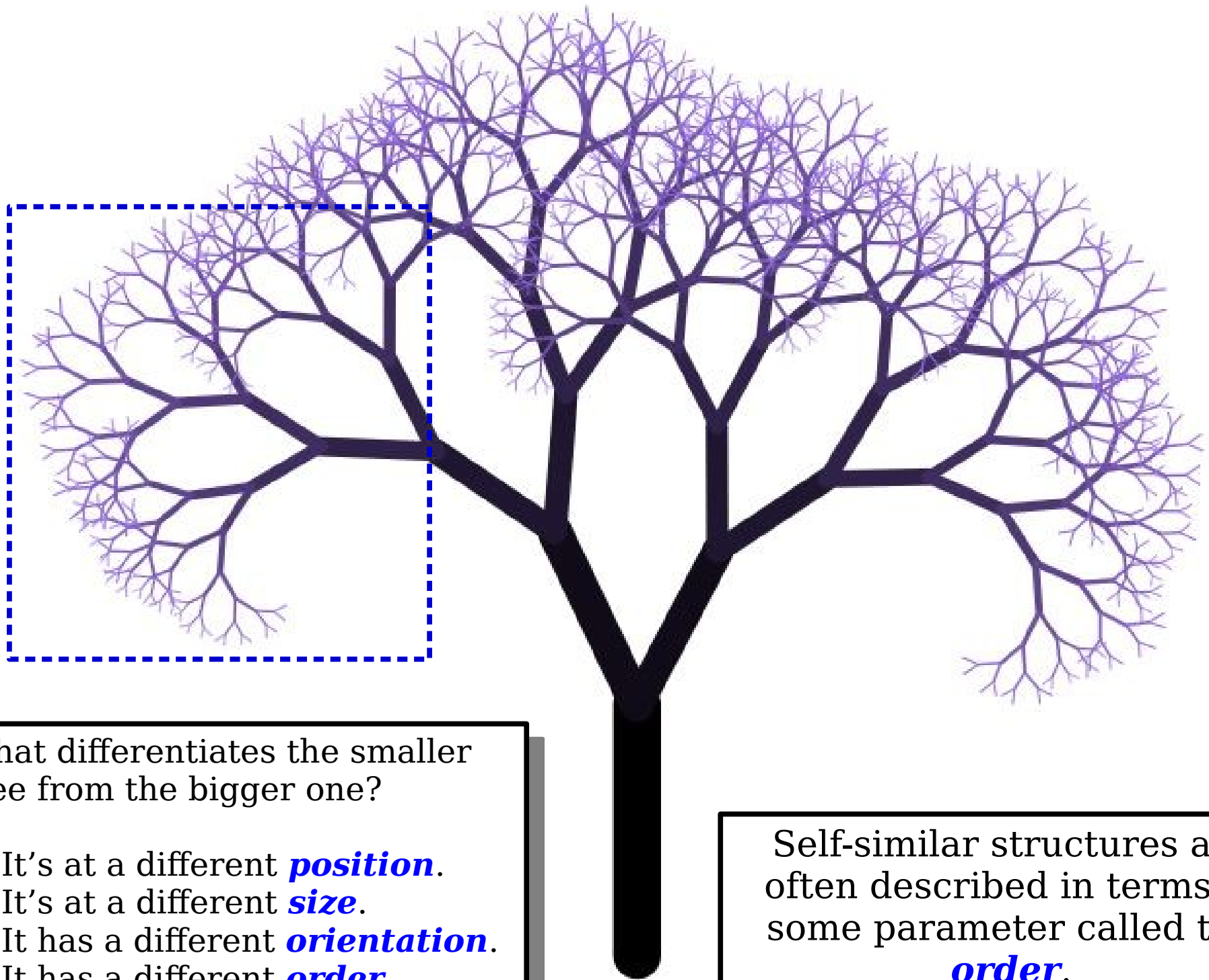
What differentiates the smaller tree from the bigger one?

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**.

Self-similar structures are often described in terms of some parameter called the **order**.

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***.

Self-similar structures are often described in terms of some parameter called the ***order***.

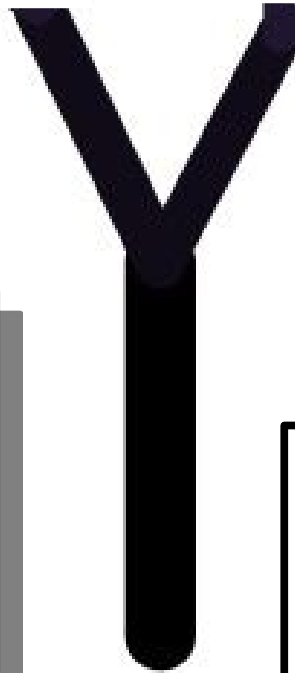
An order-1 tree.

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***.

Self-similar structures are often described in terms of some parameter called the ***order***.

An order-2 tree.

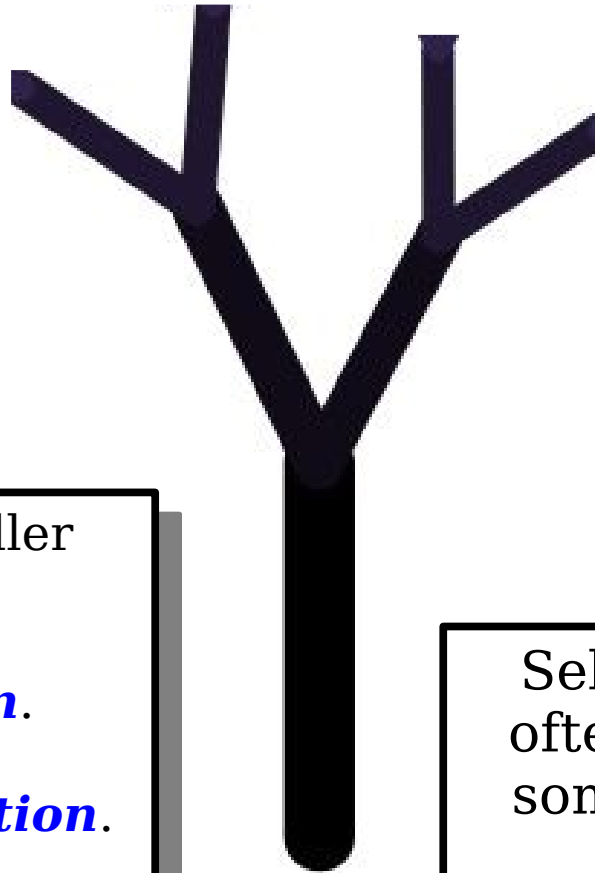


What differentiates the smaller tree from the bigger one?

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***.

Self-similar structures are often described in terms of some parameter called the ***order***.

An order-3 tree.

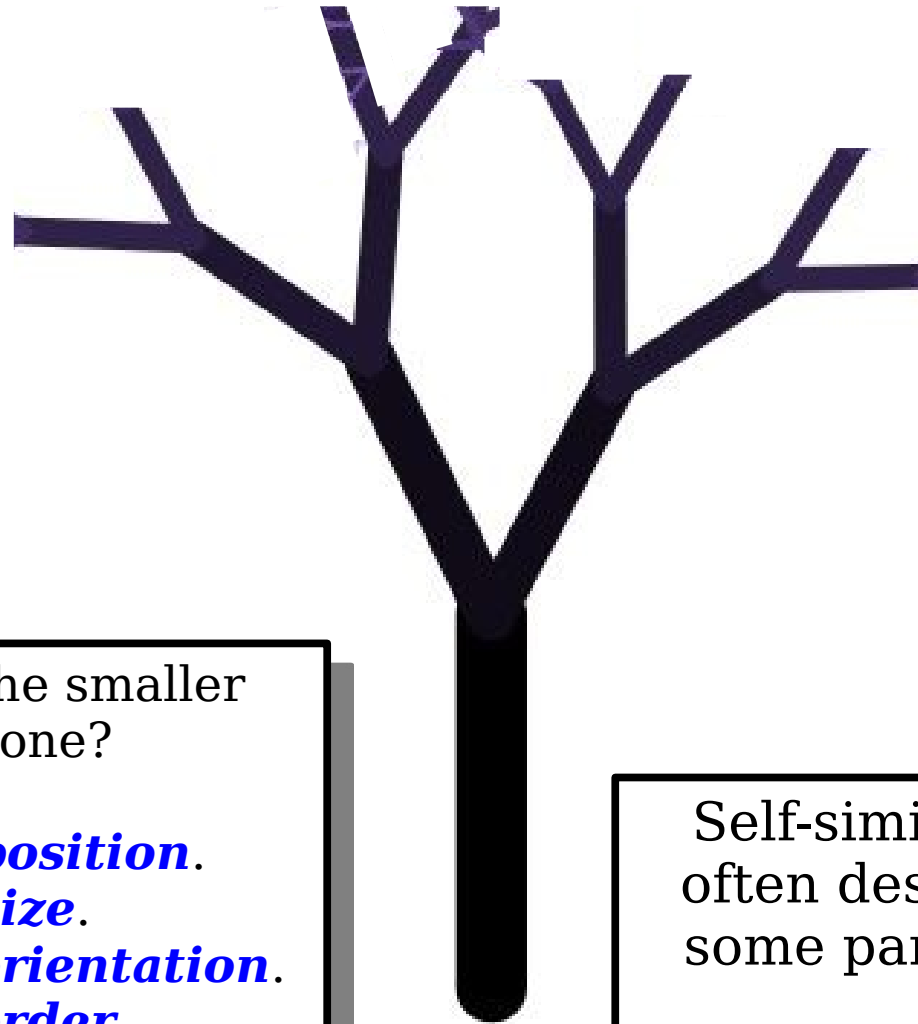


What differentiates the smaller tree from the bigger one?

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***.

Self-similar structures are often described in terms of some parameter called the ***order***.

An order-4 tree.

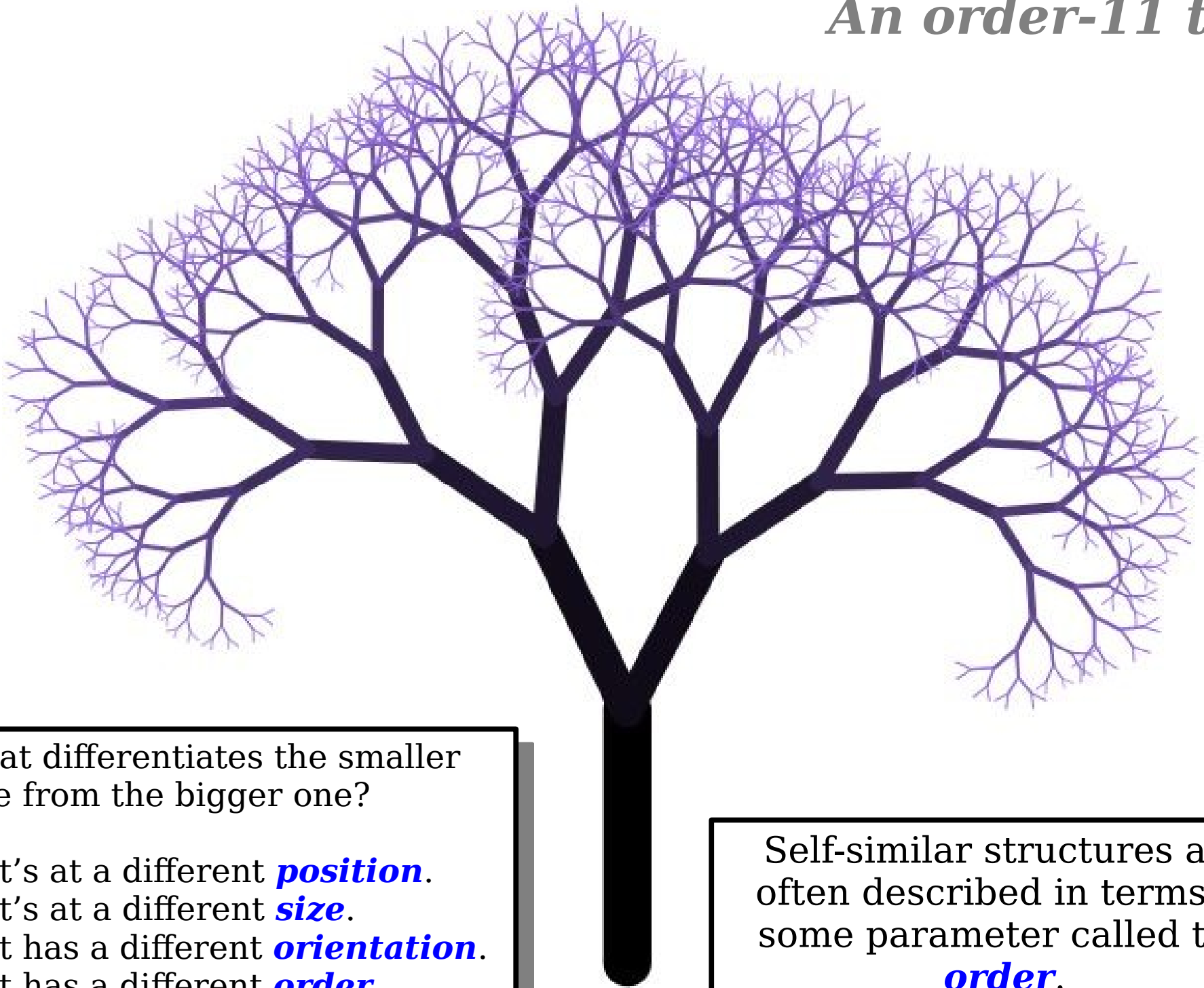


What differentiates the smaller tree from the bigger one?

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**.

Self-similar structures are often described in terms of some parameter called the **order**.

An order-11 tree.

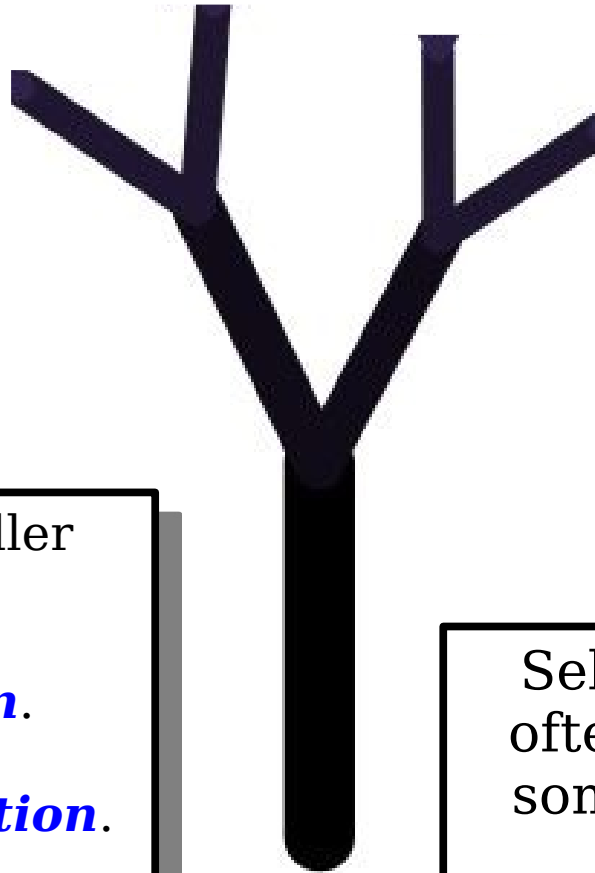


What differentiates the smaller tree from the bigger one?

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**.

Self-similar structures are often described in terms of some parameter called the **order**.

An order-3 tree.



What differentiates the smaller tree from the bigger one?

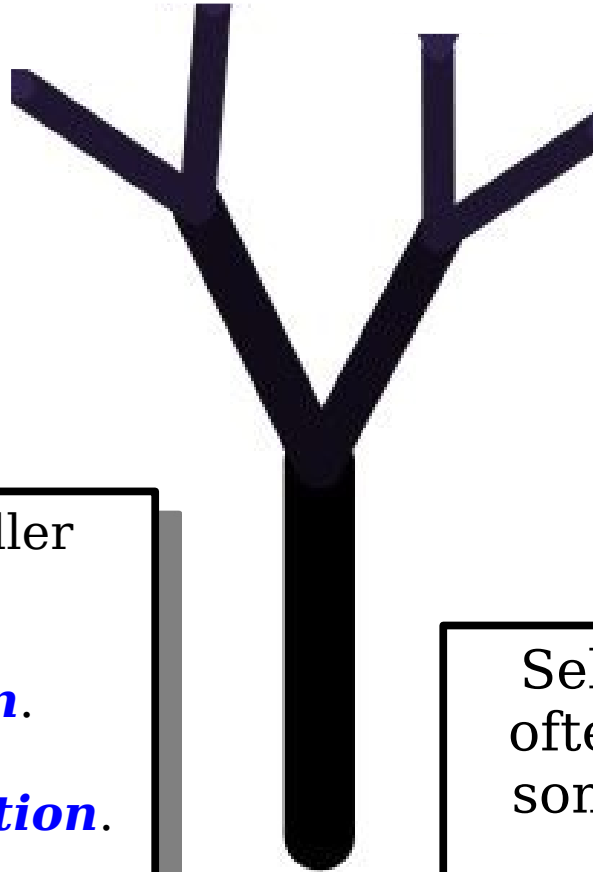
1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Self-similar structures are often described in terms of some parameter called the **order**.

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?

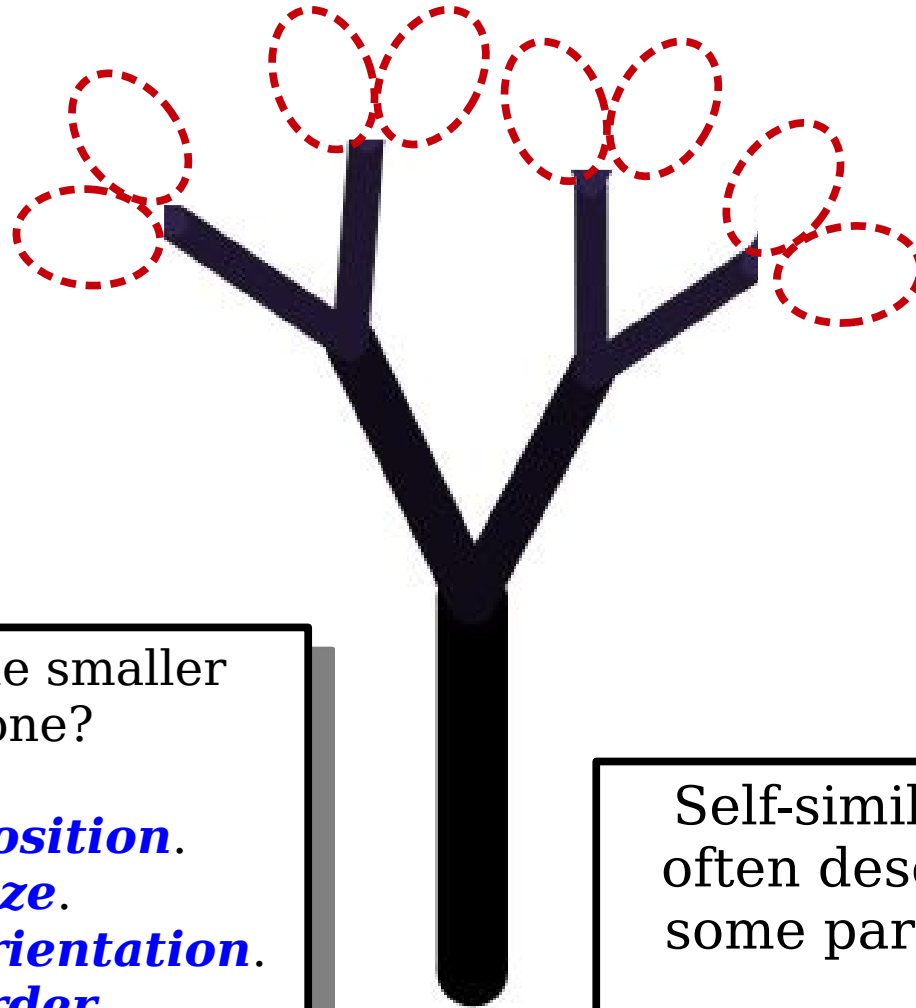
1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Self-similar structures are often described in terms of some parameter called the **order**.

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.

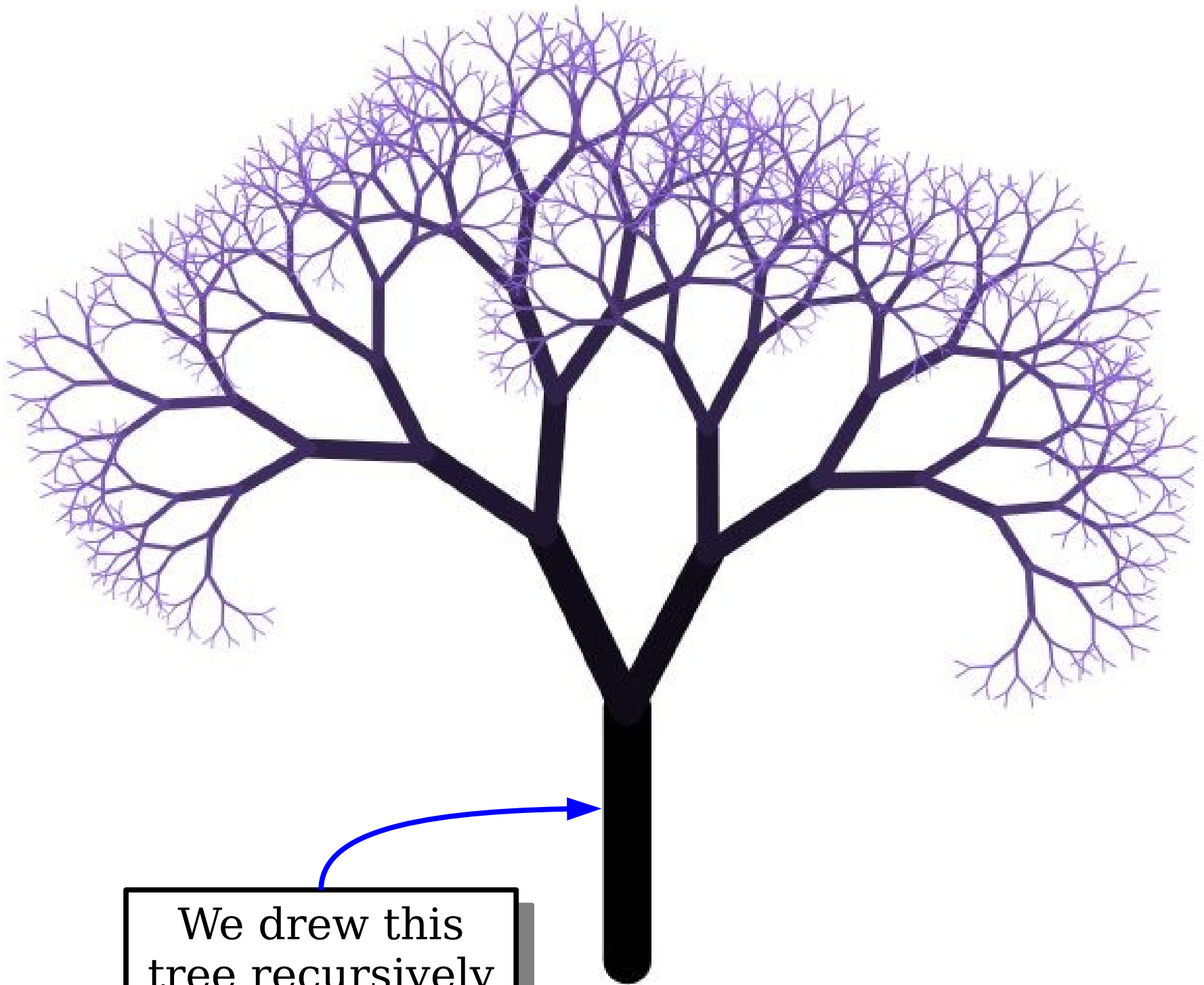


What differentiates the smaller tree from the bigger one?

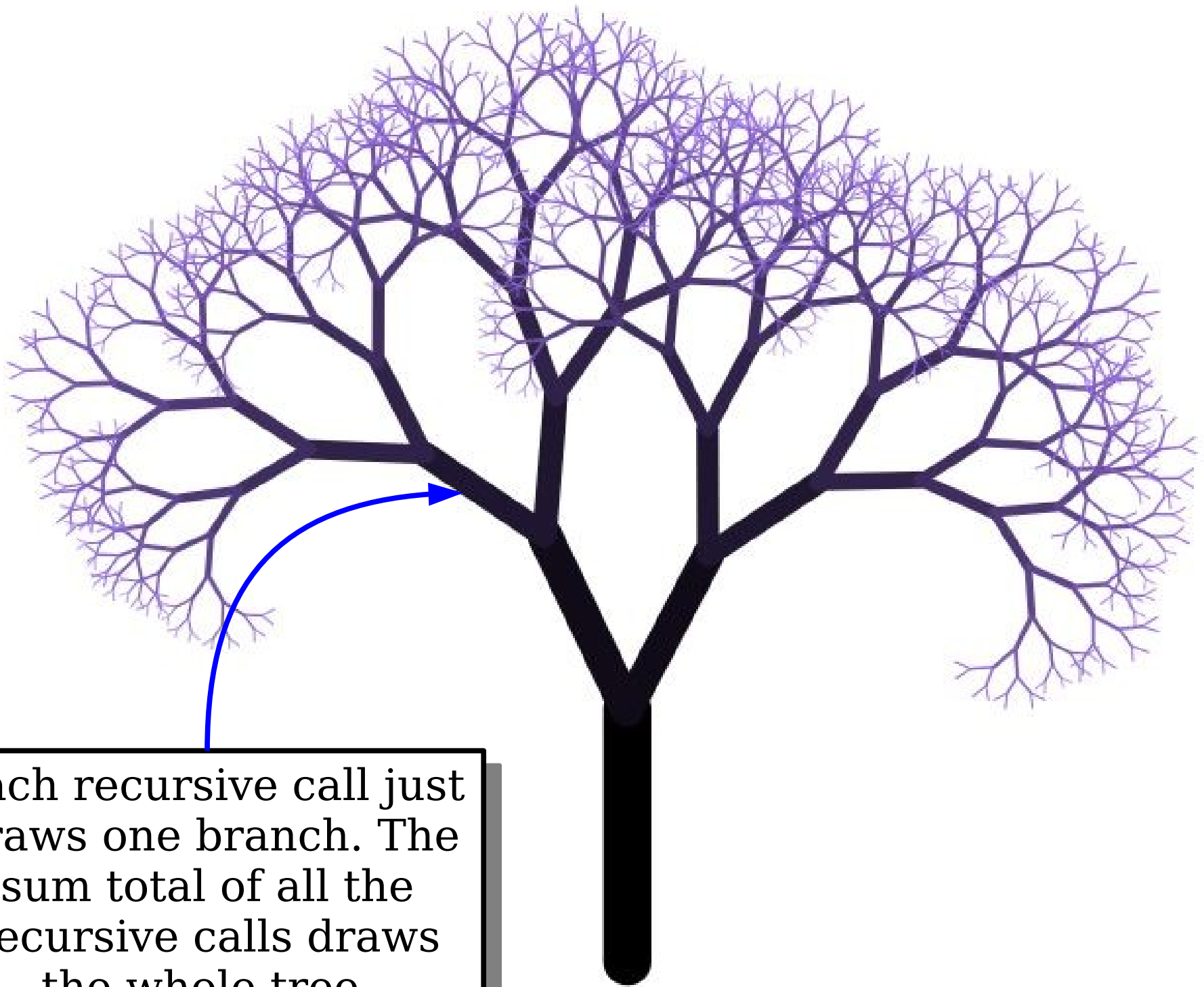
1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Self-similar structures are often described in terms of some parameter called the **order**.

To Summarize



We drew this
tree recursively



Each recursive call just draws one branch. The sum total of all the recursive calls draws the whole tree.

An Amazing Website

<http://recursivedrawing.com/>

A Quick, Relevant Tangent

Reasoning By Analogy

- What's wrong with this code?

```
const double PI = 3.14159265358979;
```

```
double areaOfCircle(double radius) {  
    return PI * radius * radius;  
}
```

```
int main() {  
    double radius = 1.61;  
    areaOfCircle(radius);  
  
    return 0;  
}
```

Formulate a hypothesis!

Reasoning By Analogy

- What's wrong with this code?

```
const double PI = 3.14159265358979;
```

```
double areaOfCircle(double radius) {  
    return PI * radius * radius;  
}
```

```
int main() {  
    double radius = 1.61;  
    areaOfCircle(radius);  
  
    return 0;  
}
```

Reasoning By Analogy

What's wrong with this code?

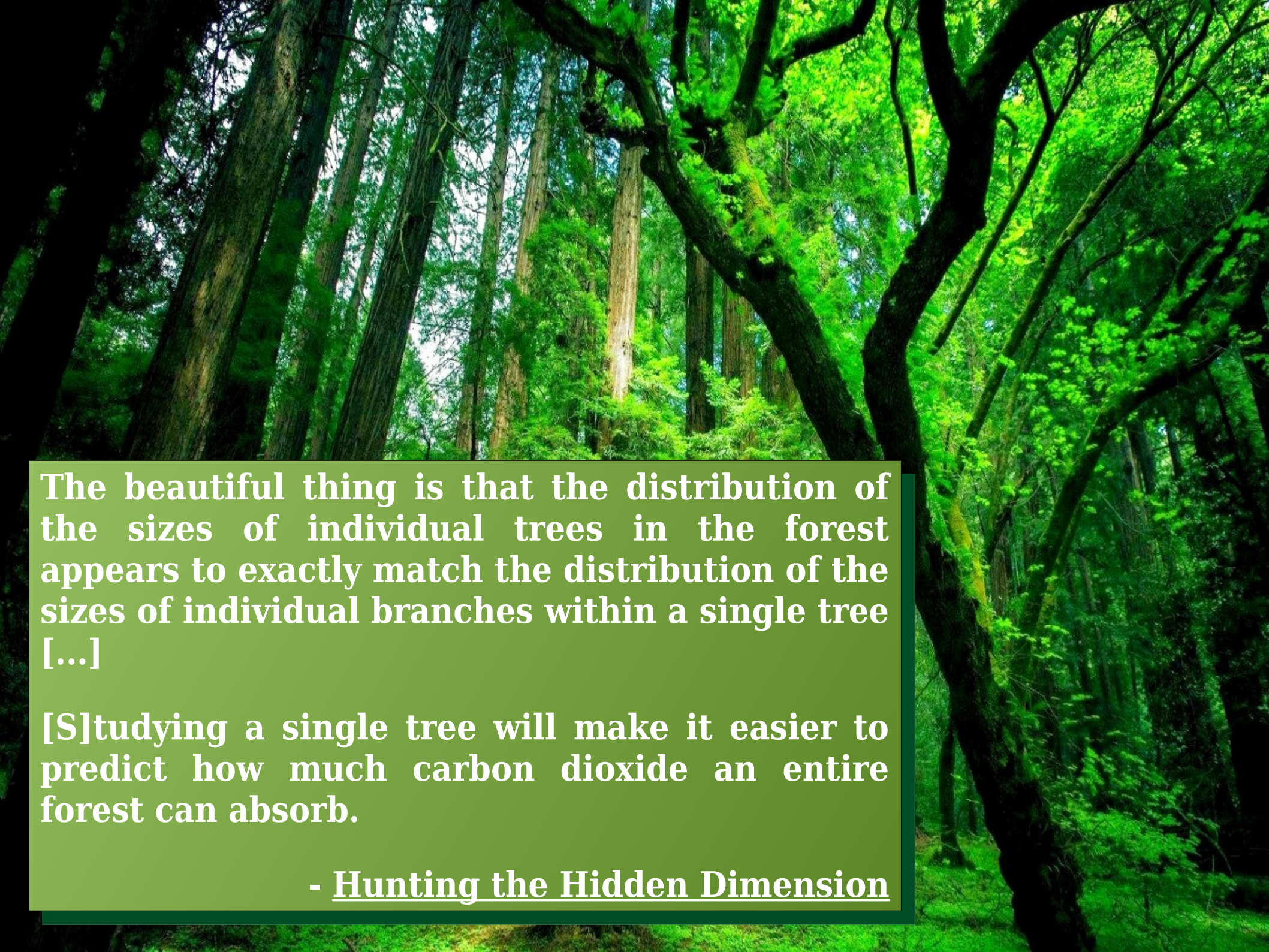
```
const double PI = 3.14159265358979;  
  
double areaOfCircle(double radius) {  
    return PI * radius * radius;  
}  
  
int main() {  
    double radius = 1.61;  
    areaOfCircle(radius);  
  
    return 0;  
}
```

It's odd to call a function that returns a value and then not use that value.

If you don't use a function's return value, it's likely a bug!

Back to Recursion...

A Practical Application

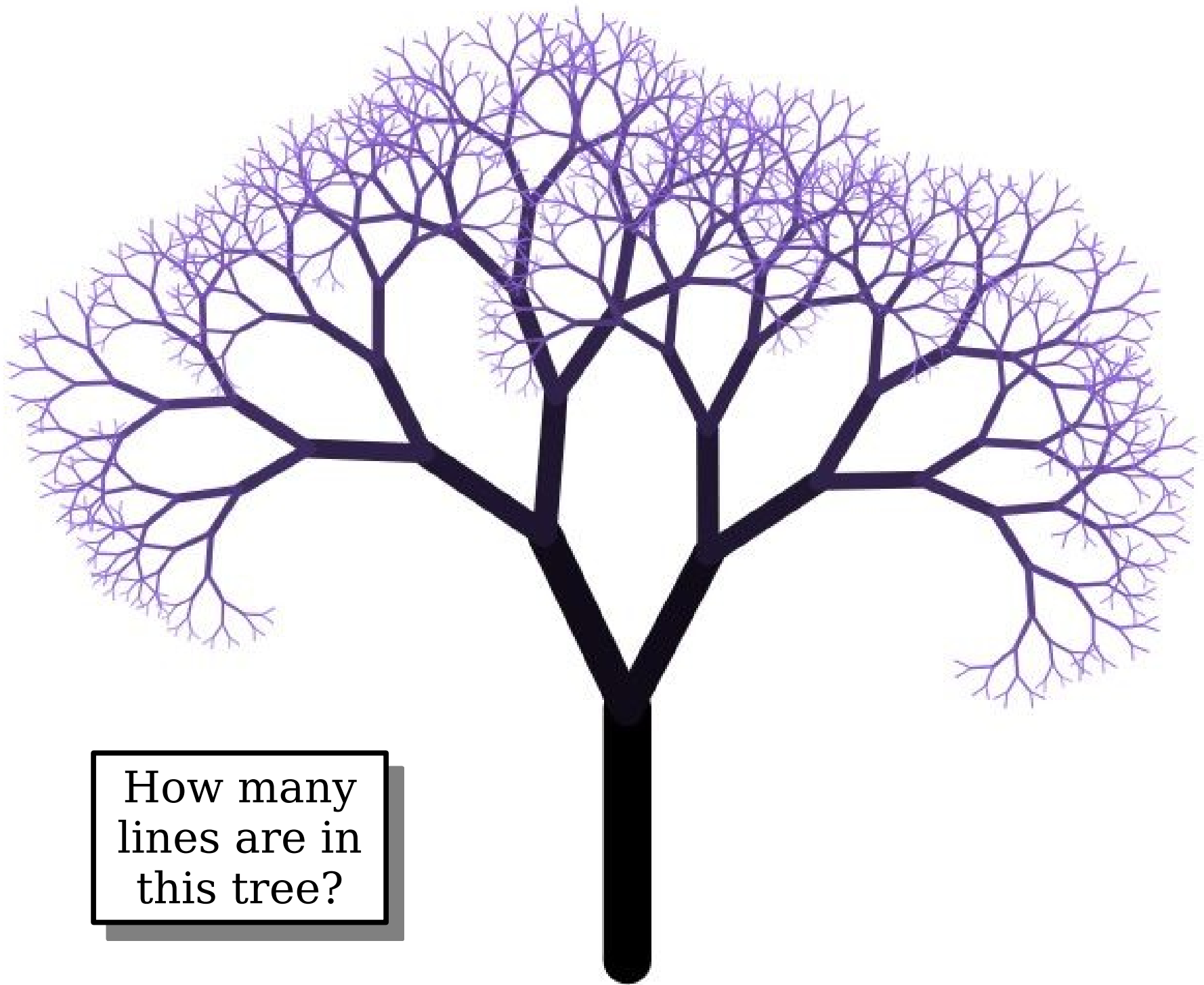
A photograph of a dense forest. In the foreground, a large, dark tree trunk with thick moss curves from the right side towards the center. The background is filled with many tall, straight, light-colored tree trunks reaching up towards a canopy of vibrant green leaves. Sunlight filters through the leaves, creating a dappled light effect.

The beautiful thing is that the distribution of the sizes of individual trees in the forest appears to exactly match the distribution of the sizes of individual branches within a single tree [...]

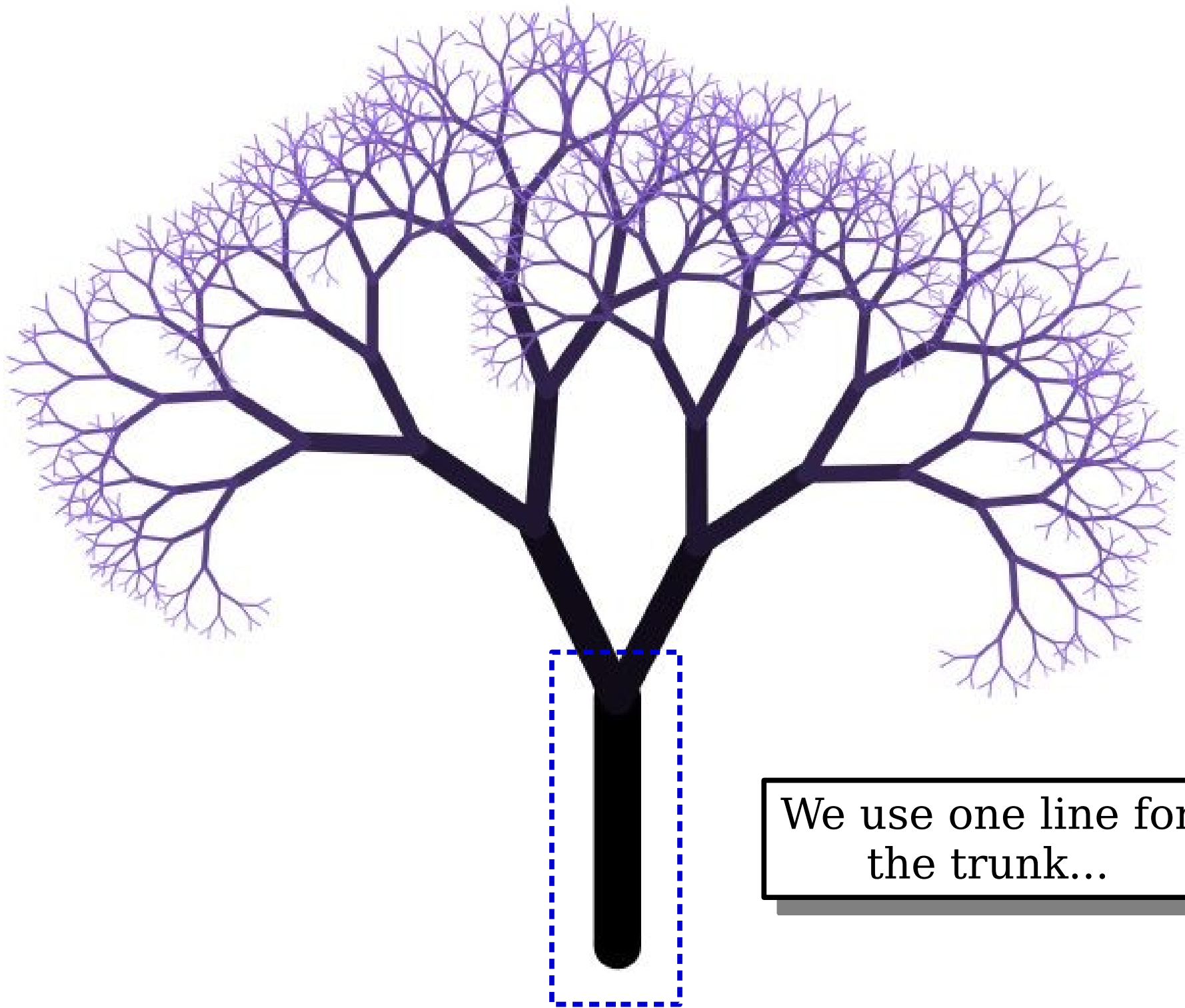
[S]tudying a single tree will make it easier to predict how much carbon dioxide an entire forest can absorb.

- Hunting the Hidden Dimension

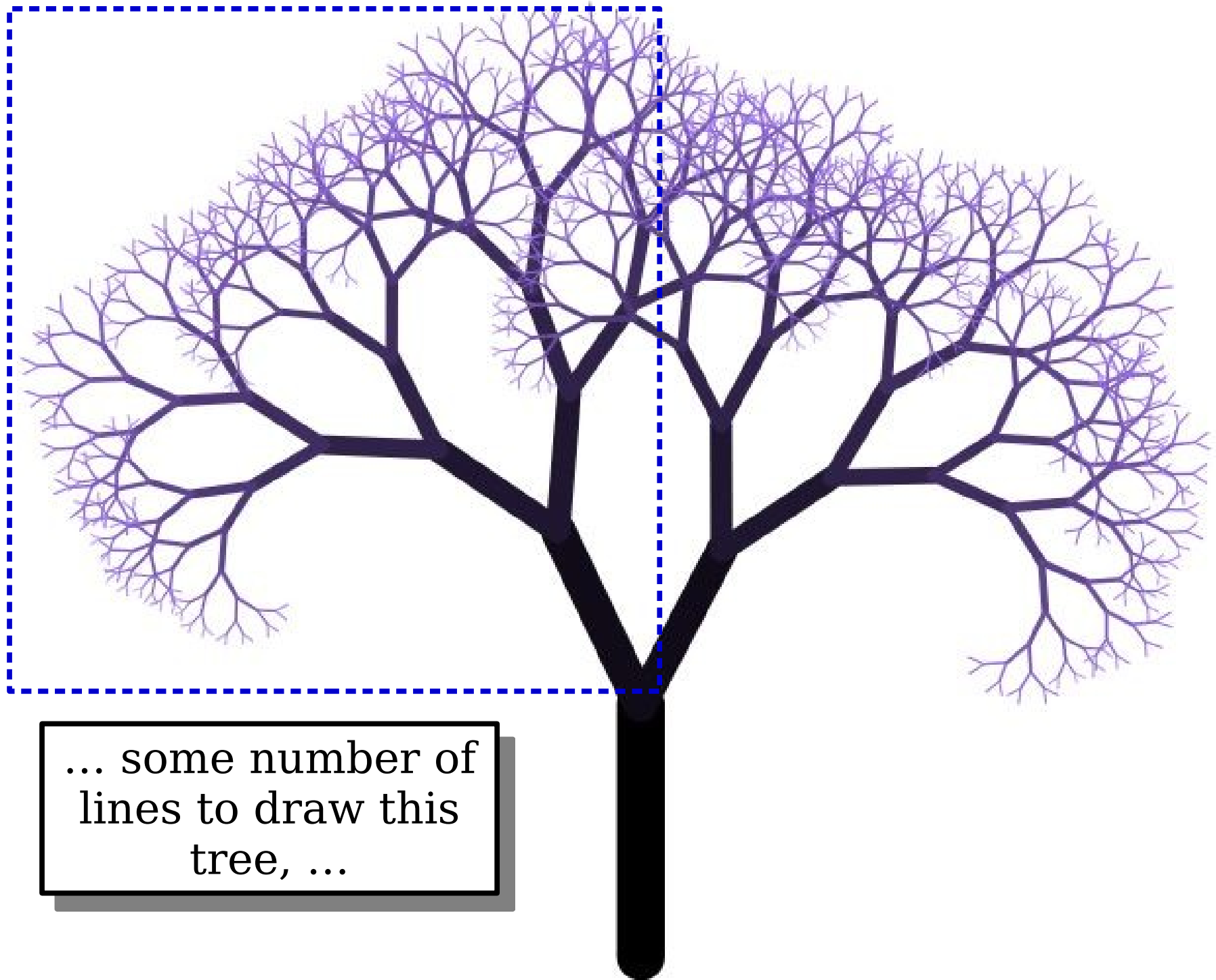
How many lines make up each tree?



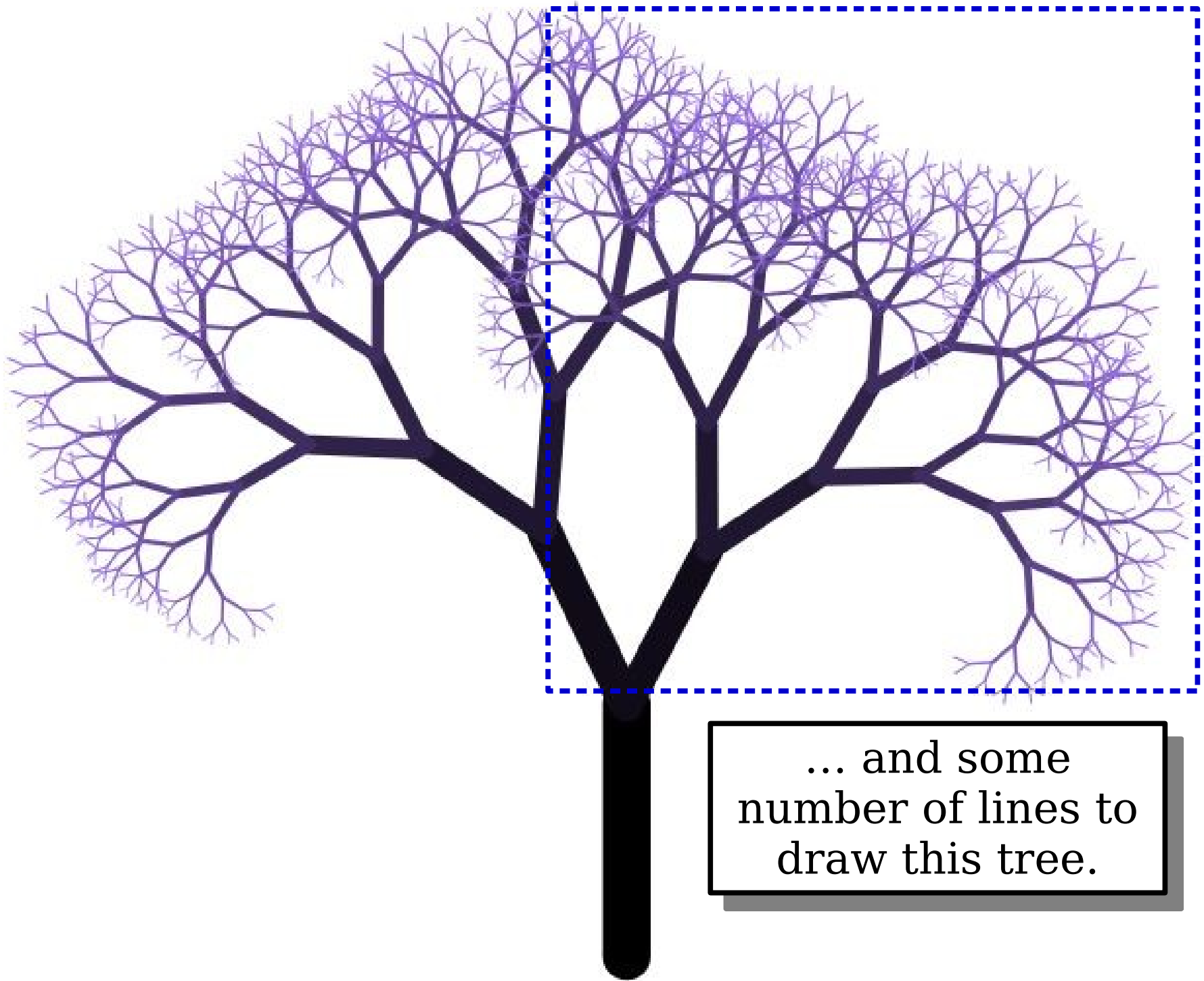
How many
lines are in
this tree?



We use one line for
the trunk...



... some number of
lines to draw this
tree, ...



... and some
number of lines to
draw this tree.

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```

0

numLines



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

0

numLines

```
int numLines = 0;  
GPoint endpoint = drawPolarLine(/* ... */);  
numLines++;
```

```
drawTree(/* ... */);  
drawTree(/* ... */);
```

```
return numLines;
```

```
}
```

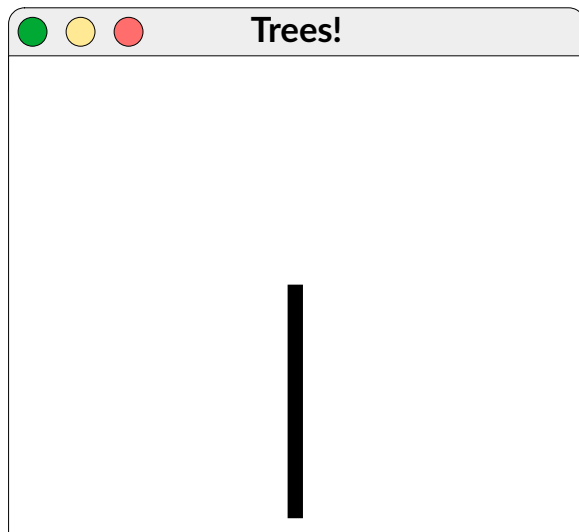


```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

0

numLines

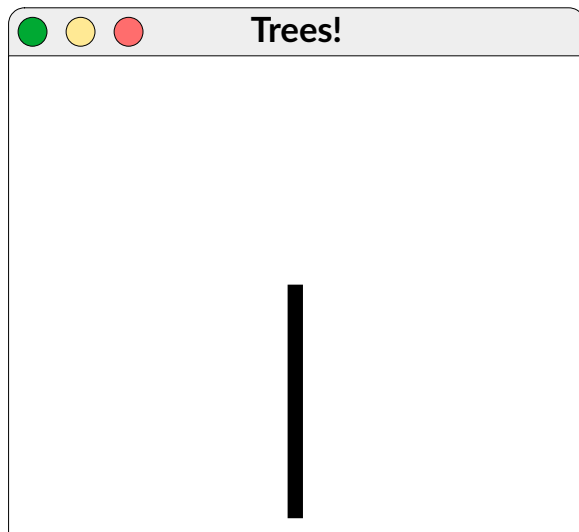
```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```

0

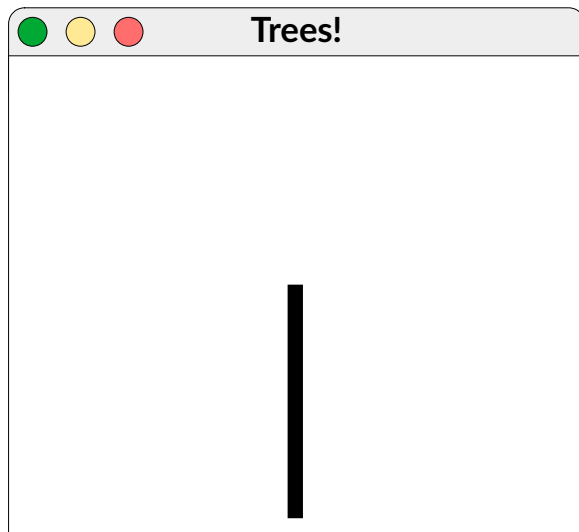
numLines



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```

1

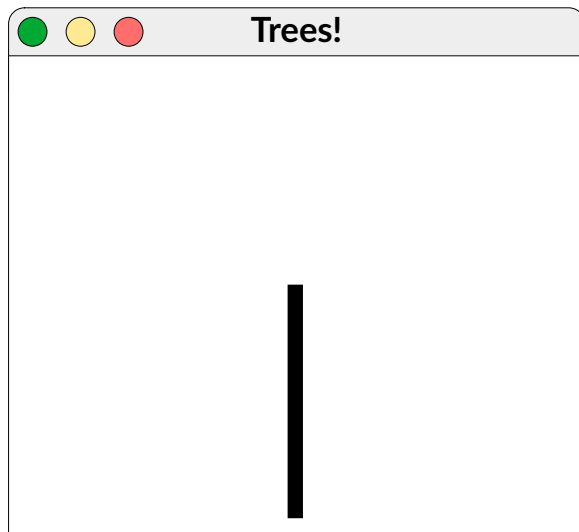
numLines



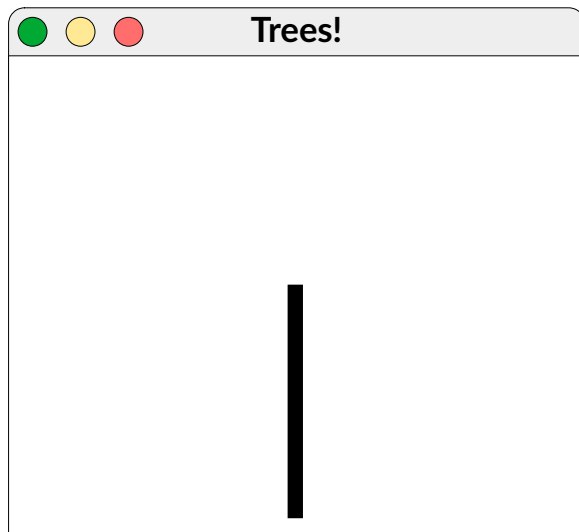
```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```

1

numLines

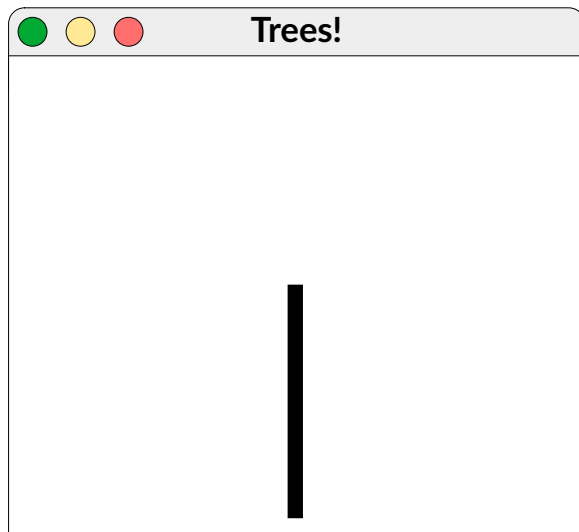


```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```

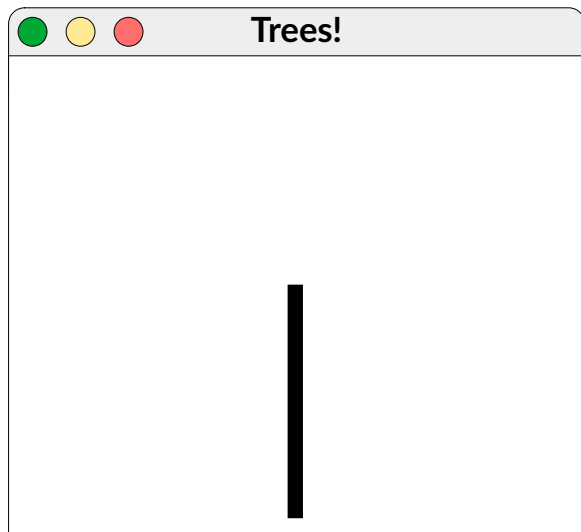


```
int drawTree(/* ... */) {  
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
  
        int numLines = 0;  
        GPoint endpoint = drawPolarLine(/* ... */);  
        numLines++;  
  
        drawTree(/* ... */);  
        drawTree(/* ... */);  
    }  
  
    return numLines;  
}
```

1



```
int drawTree(/* ... */) {  
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
  
        int numLines = 0;  
        GPoint endpoint = drawPolarLine(/* ... */);  
        numLines++;  
  
        drawTree(/* ... */);  
        drawTree(/* ... */);  
  
        return numLines;  
    }  
}
```



```
int drawTree(/* ... */) {
```

1

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }
```

0

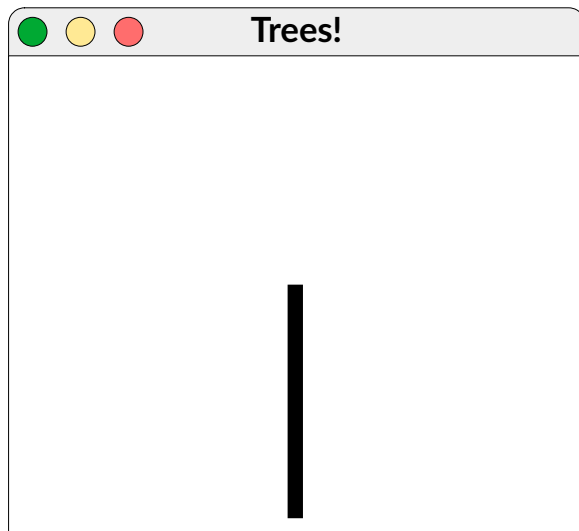
numLines

```
        int numLines = 0;  
        GPoint endpoint = drawPolarLine(/* ... */);  
        numLines++;
```

```
        drawTree(/* ... */);  
        drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

```
    if (order == 0) {  
        return 0;  
    }
```

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    drawTree(/* ... */);  
    drawTree(/* ... */);
```

```
    return numLines;  
}
```

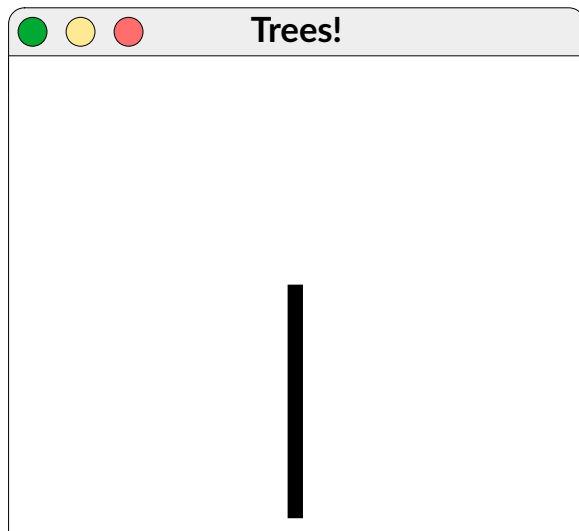
1

0

numLines

It's reasonable to guess that this line is the problem because it looks like it resets numLines to zero at each call.

But that's not actually the issue. Remember - every recursive call gets its own copies of all local variables.



```
int drawTree(/* ... */) {
```

1

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

0

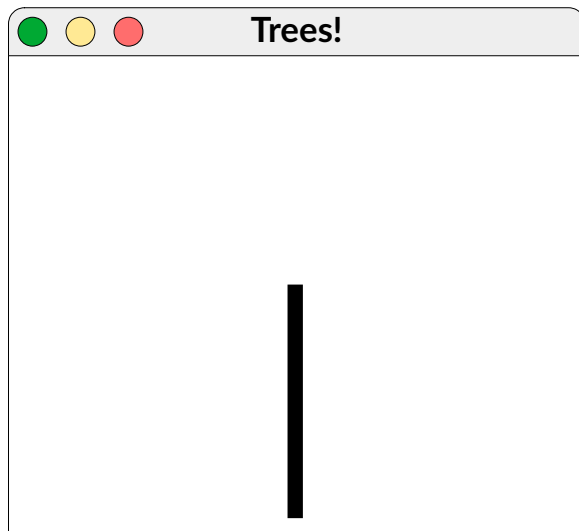
numLines

```
int numLines = 0;  
GPoint endpoint = drawPolarLine(/* ... */);  
numLines++;
```

```
drawTree(/* ... */);  
drawTree(/* ... */);
```

```
return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

0

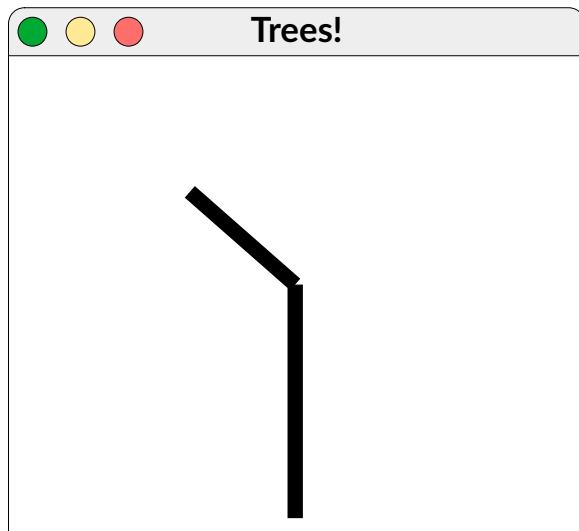
numLines

```
int numLines = 0;  
GPoint endpoint = drawPolarLine(/* ... */);  
numLines++;
```

```
drawTree(/* ... */);  
drawTree(/* ... */);
```

```
return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
    }
```

0

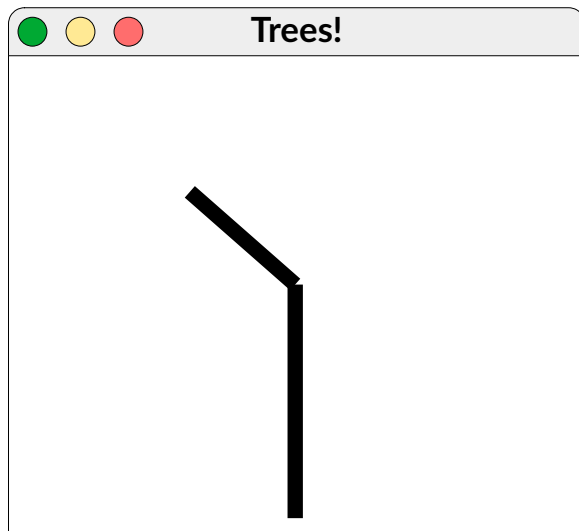
numLines

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    drawTree(/* ... */);  
    drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }
```

```
        int numLines = 0;
```

```
        GPoint endpoint = drawPolarLine(/* ... */);  
        numLines++;
```

```
        drawTree(/* ... */);  
        drawTree(/* ... */);
```

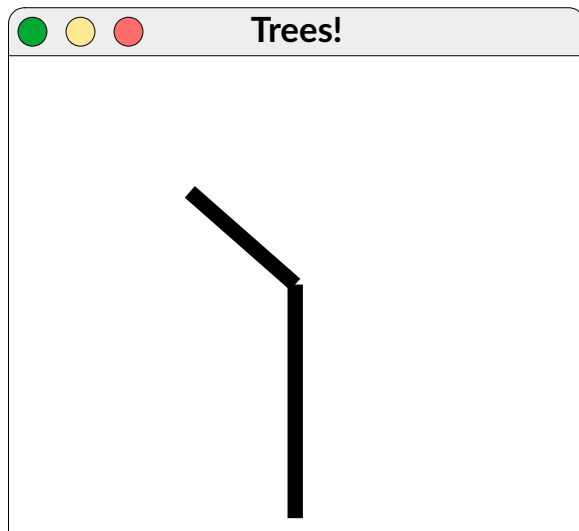
```
        return numLines;
```

```
    }
```

1

1

numLines



```
int drawTree(/* ... */) {
```

```
    if (order == 0) {  
        return 0;  
    }
```

```
    int numLines = 0;
```

```
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    drawTree(/* ... */);  
    drawTree(/* ... */);
```

```
    return numLines;
```

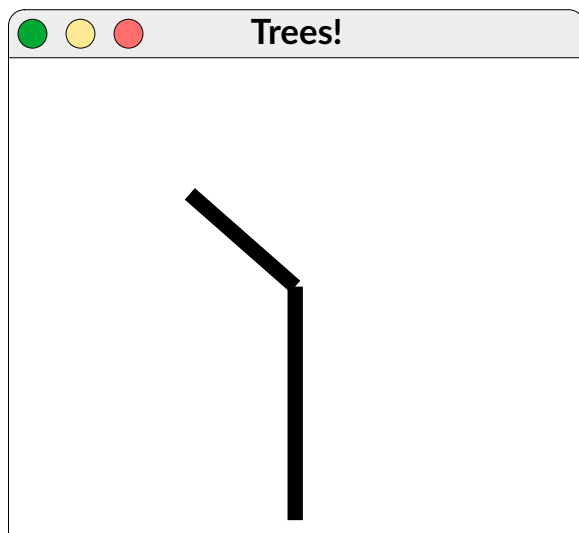
1

1

numLines

It's also reasonable to guess that the error is that this isn't incrementing the copy of numLines inside of the top-level call.

While it's true that this doesn't increment the top-level copy of numLines, that isn't an error *per se*. This function says it will return the number of lines drawn, not update a global total somewhere.



```
int drawTree(/* ... */) {
```

1

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

1

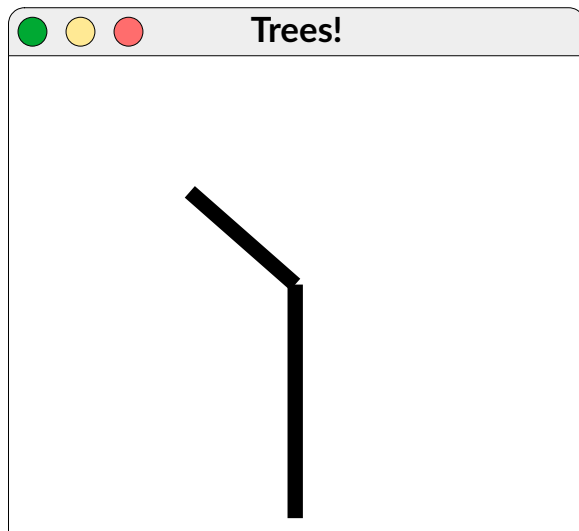
numLines

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    drawTree(/* ... */);  
    drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }
```

1

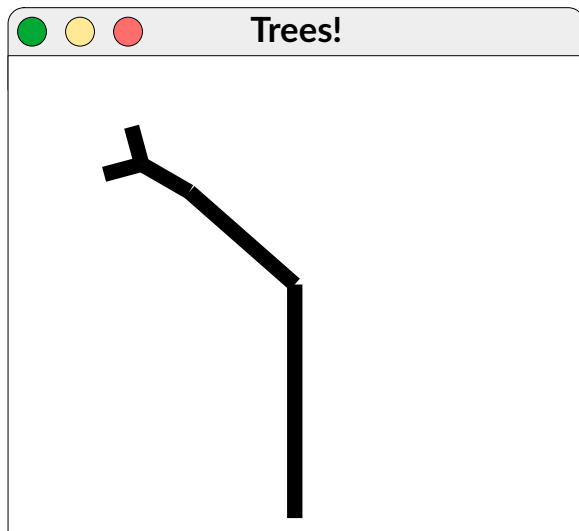
numLines

```
        int numLines = 0;  
        GPoint endpoint = drawPolarLine(/* ... */);  
        numLines++;
```

```
        drawTree(/* ... */);  
        drawTree(/* ... */);
```

```
        return numLines;
```

```
    }
```



```
int drawTree(/* ... */) {
```

```
    if (order == 0) {  
        return 0;  
    }
```

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

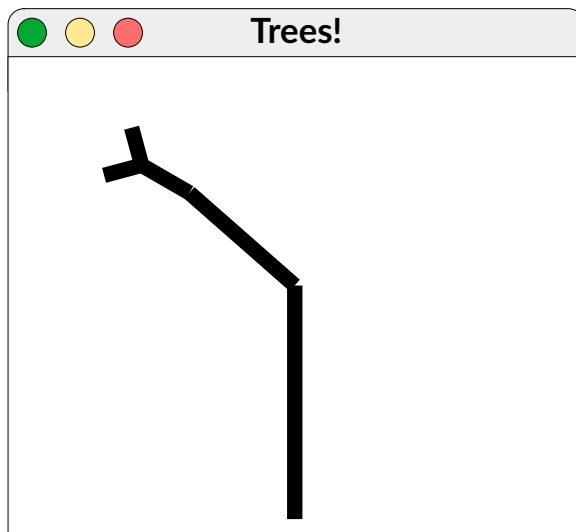
```
    drawTree(/* ... */);  
    drawTree(/* ... */);
```

```
    return numLines;  
}
```

1

1

numLines



This function returns an integer, but we didn't do anything with that integer! It would be like writing this line of code:

```
sqrt(137);
```

This computes a square root, but doesn't store it anywhere. Oops! Our total is now wrong.

```
int drawTree(/* ... */) {
```

1

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }
```

1

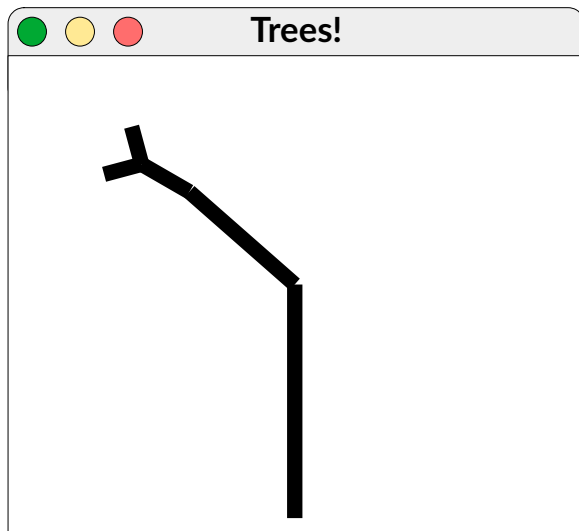
numLines

```
        int numLines = 0;  
        GPoint endpoint = drawPolarLine(/* ... */);  
        numLines++;
```

```
        drawTree(/* ... */);  
        drawTree(/* ... */);
```

```
        return numLines;
```

```
    }
```



```
int drawTree(/* ... */) {
```

1

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
    }
```

1

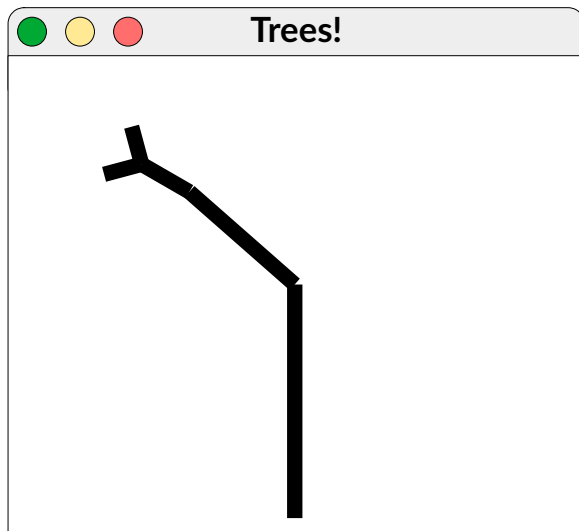
numLines

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    drawTree(/* ... */);  
    drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }
```

1

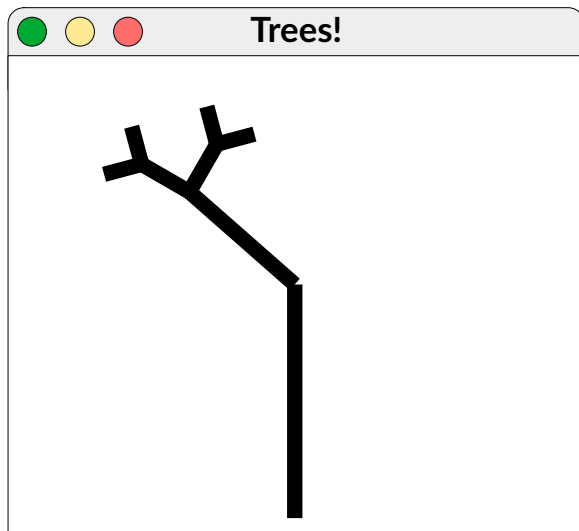
numLines

```
        int numLines = 0;  
        GPoint endpoint = drawPolarLine(/* ... */);  
        numLines++;
```

```
        drawTree(/* ... */),  
        drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
    }
```

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

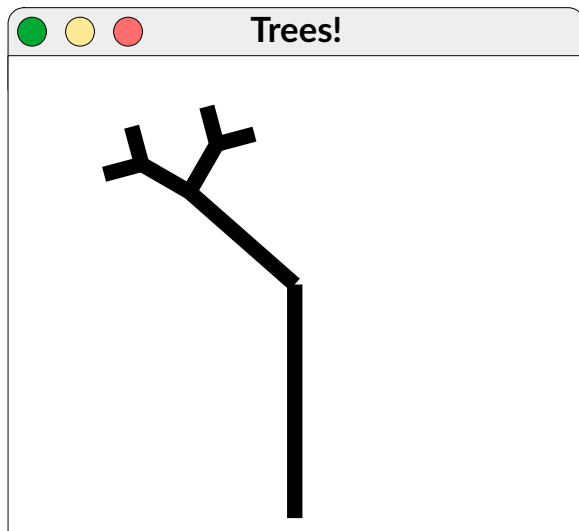
```
    drawTree(/* ... */);  
    drawTree(/* ... */);
```

```
    return numLines;  
}
```

1

1

numLines



Oops - we didn't do anything with the return value.

```
int drawTree(/* ... */) {
```

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
    }
```

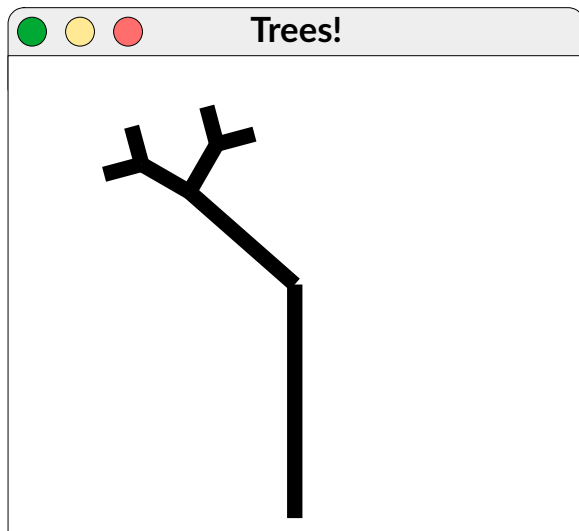
```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```

```
    return numLines;
```

1

1

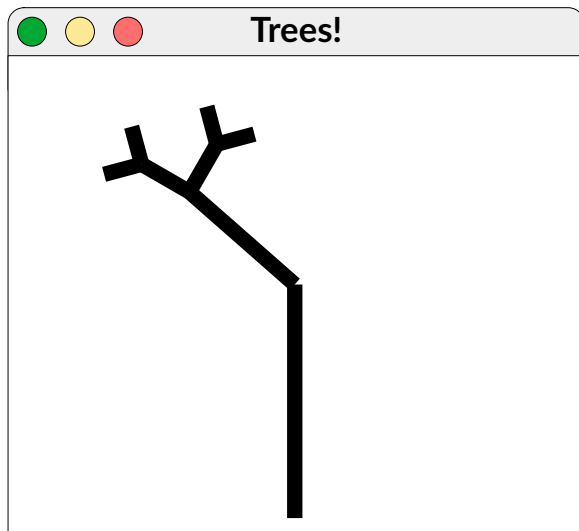
numLines



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```

1

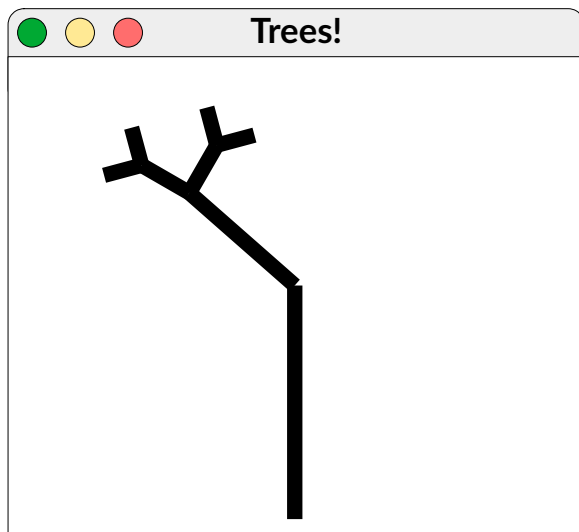
numLines



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
    return numLines;  
}
```

1

numLines

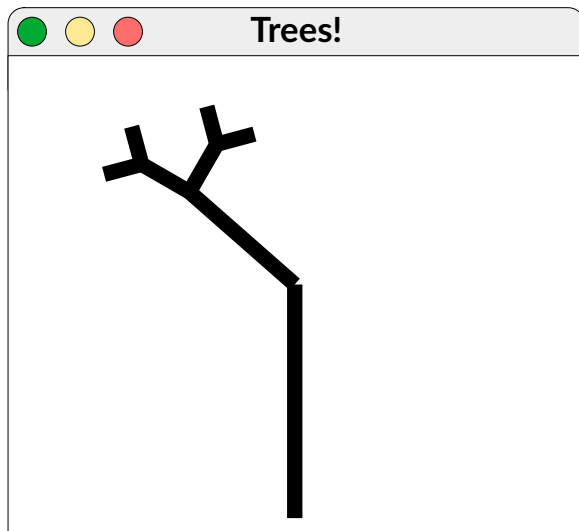


Oops - we didn't do anything with the return value.

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```

1

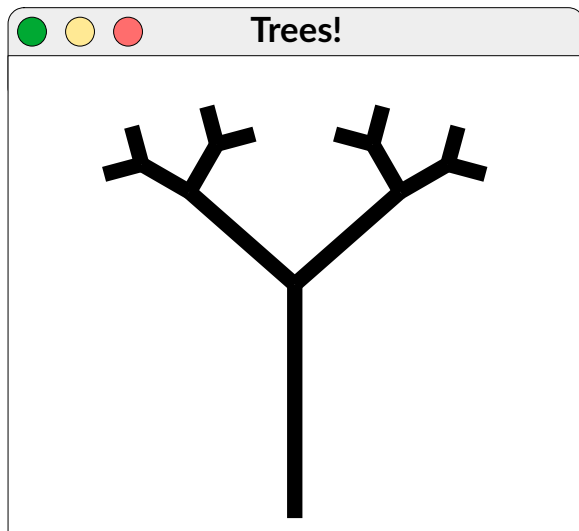
numLines



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```

1

numLines

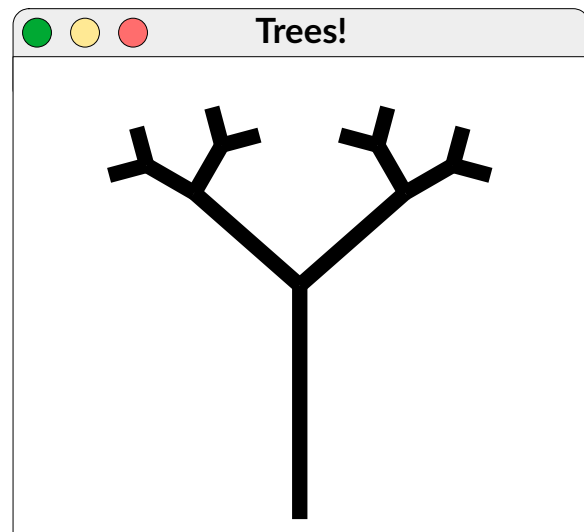


```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
  
    return numLines;  
}
```

1

numLines

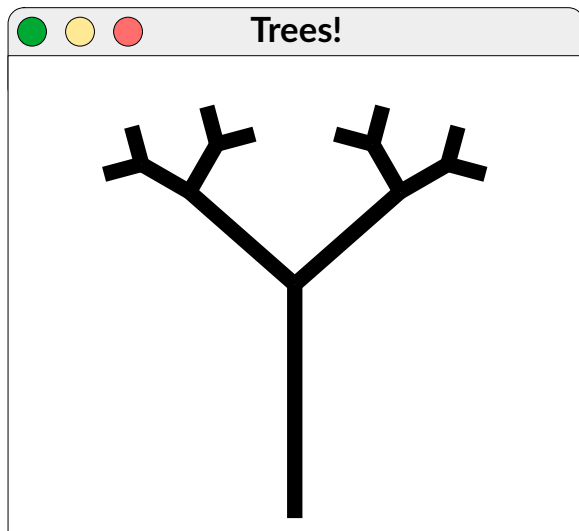
Oops - we didn't do
anything with the
return value.



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    drawTree(/* ... */);  
    drawTree(/* ... */);  
    return numLines;  
}
```

1

numLines



General Advice

- If a function returns a value, you should, in general, do something with that value.
 - Otherwise, the function did all this hard work for you, and you just dropped it on the floor!
- If you're writing a recursive function that returns a value, you should explicitly do something with the value returned by each recursive call.
 - Otherwise, your recursive call is trying to tell you something, and you're ignoring it!

The Correction

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);  
  
    return numLines;  
}
```



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);  
  
    return numLines;  
}
```



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);  
  
    return numLines;  
}
```



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);  
  
    return numLines;  
}
```

0

numLines



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

0

numLines

```
int numLines = 0;  
GPoint endpoint = drawPolarLine(/* ... */);  
numLines++;
```

```
numLines += drawTree(/* ... */);  
numLines += drawTree(/* ... */);
```

```
return numLines;
```

```
}
```



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

0

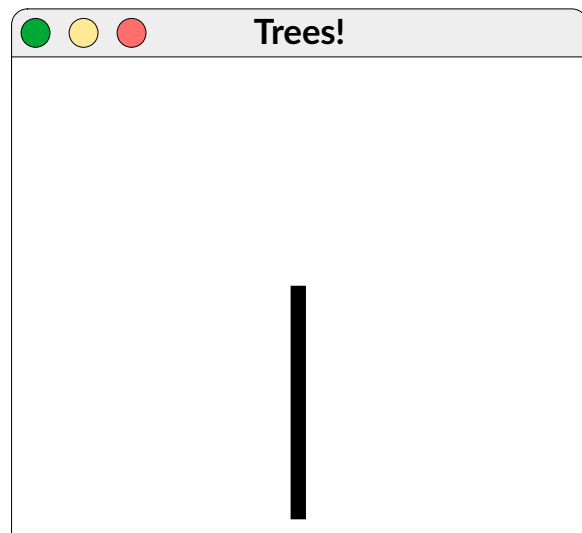
numLines

```
int numLines = 0;  
GPoint endpoint = drawPolarLine(/* ... */);  
numLines++;
```

```
numLines += drawTree(/* ... */);  
numLines += drawTree(/* ... */);
```

```
return numLines;
```

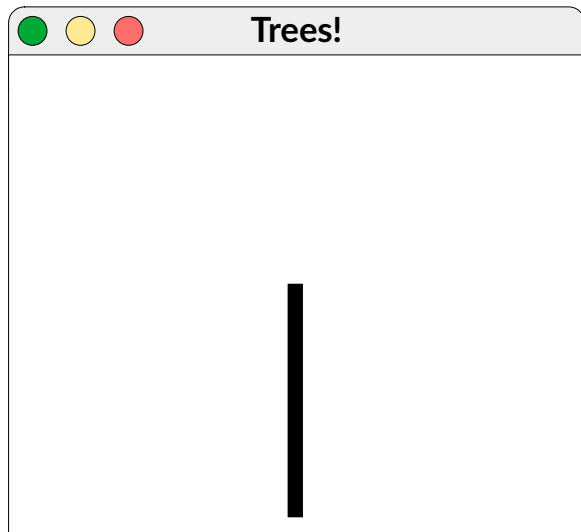
```
}
```



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);  
  
    return numLines;  
}
```

0

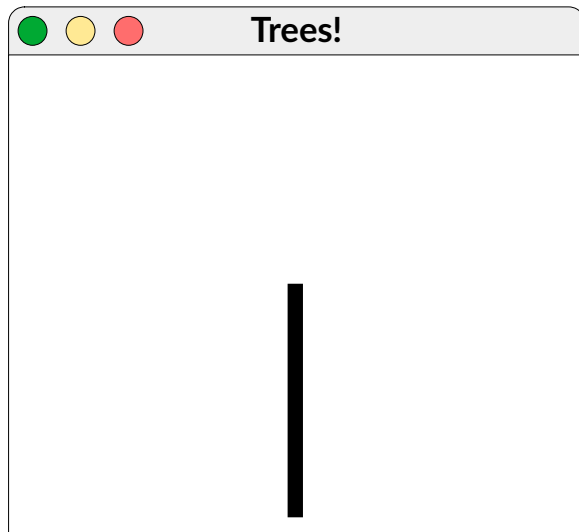
numLines



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);  
  
    return numLines;  
}
```

1

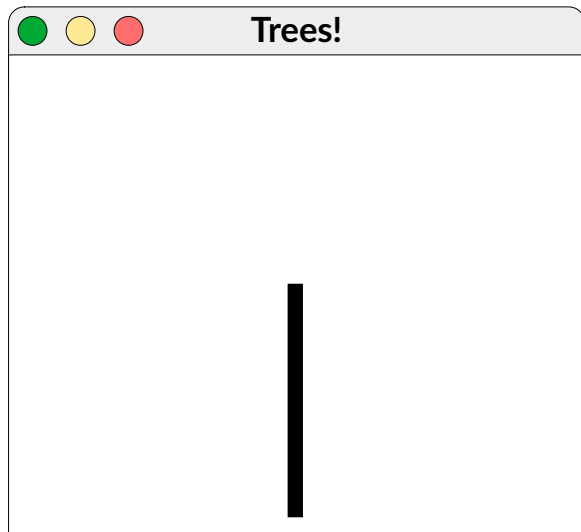
numLines



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);  
  
    return numLines;  
}
```

1

numLines



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

1

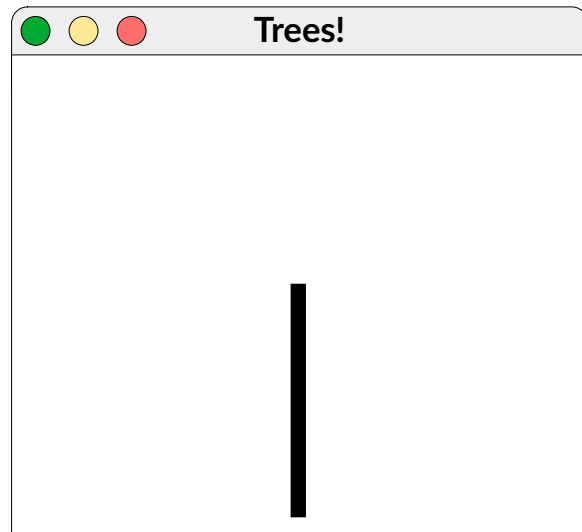
numLines

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

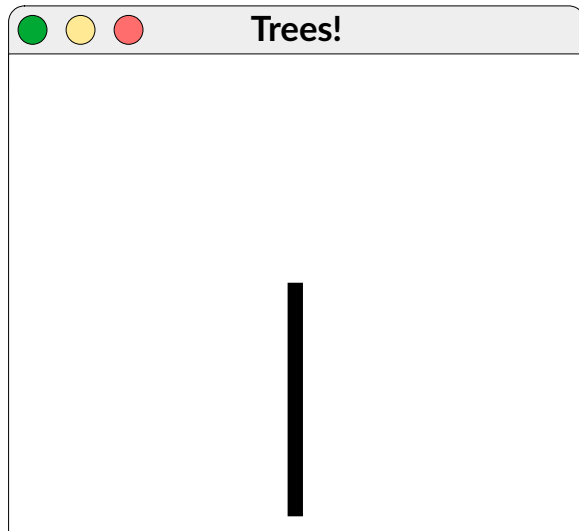
```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
    }
```

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

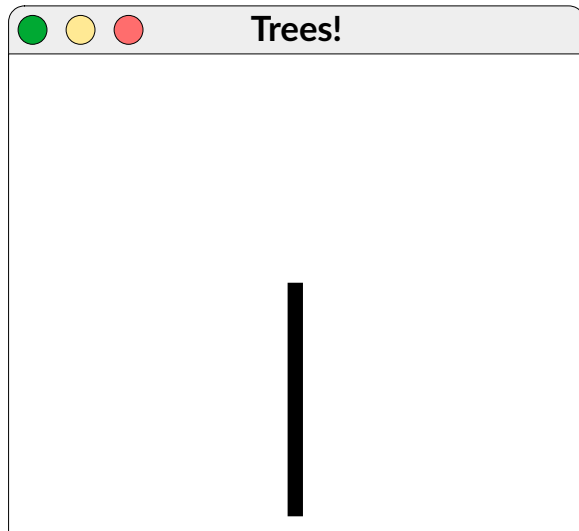
1

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;  
}
```



```
int drawTree(/* ... */) {
```

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
    }
```

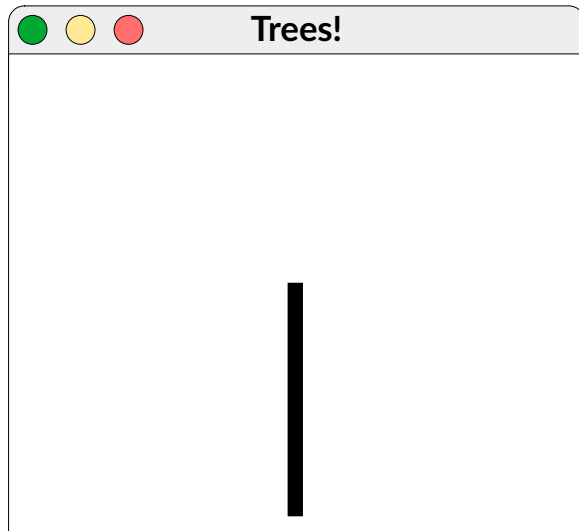
```
    int numLines = 0;
```

```
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }
```

0

numLines

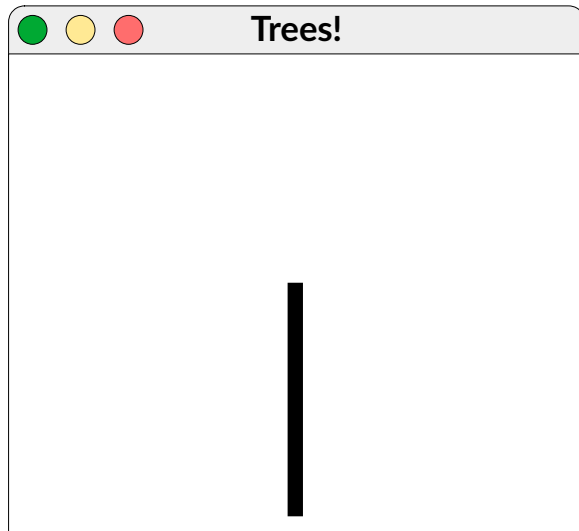
```
        int numLines = 0;
```

```
        GPoint endpoint = drawPolarLine(/* ... */);  
        numLines++;
```

```
        numLines += drawTree(/* ... */);  
        numLines += drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

0

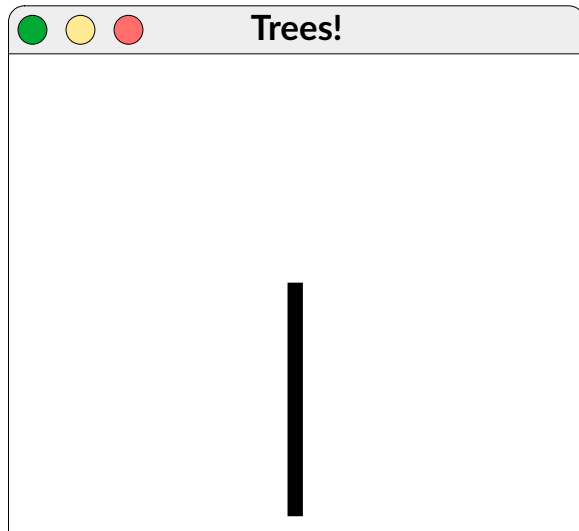
numLines

```
int numLines = 0;  
GPoint endpoint = drawPolarLine(/* ... */);  
numLines++;
```

```
numLines += drawTree(/* ... */);  
numLines += drawTree(/* ... */);
```

```
return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

0

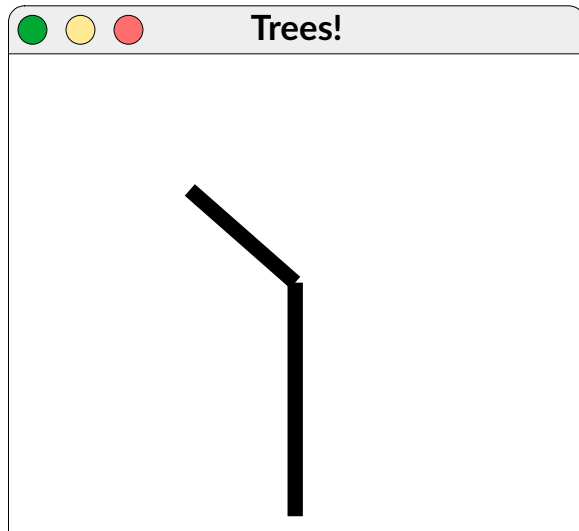
numLines

```
int numLines = 0;  
GPoint endpoint = drawPolarLine(/* ... */);  
numLines++;
```

```
numLines += drawTree(/* ... */);  
numLines += drawTree(/* ... */);
```

```
return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

0

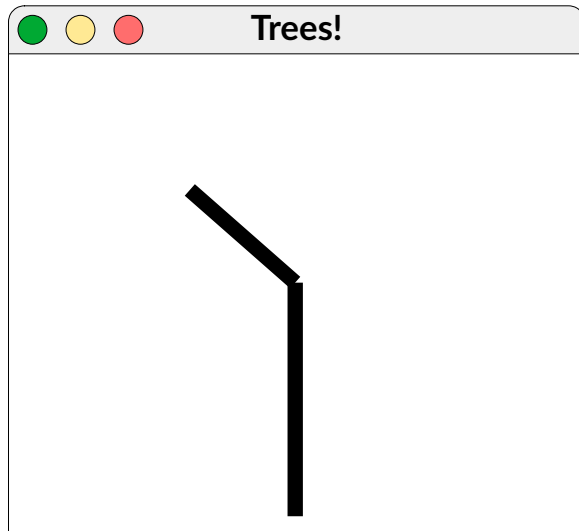
numLines

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

```
    if (order == 0) {  
        return 0;  
    }
```

```
    int numLines = 0;
```

```
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

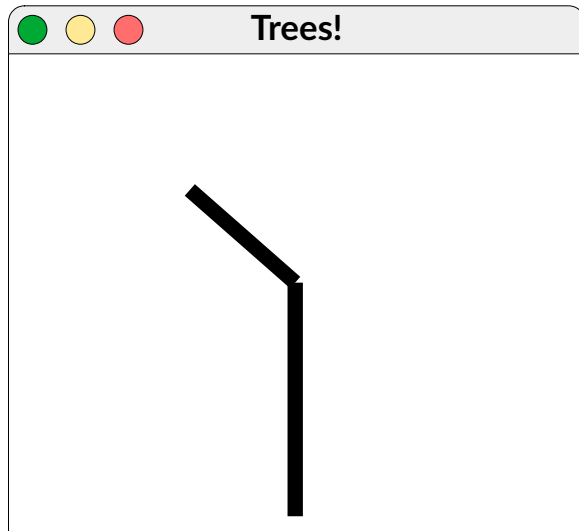
```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;  
}
```

1

1

numLines



```
int drawTree(/* ... */) {
```

```
    if (order == 0) {  
        return 0;  
    }
```

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

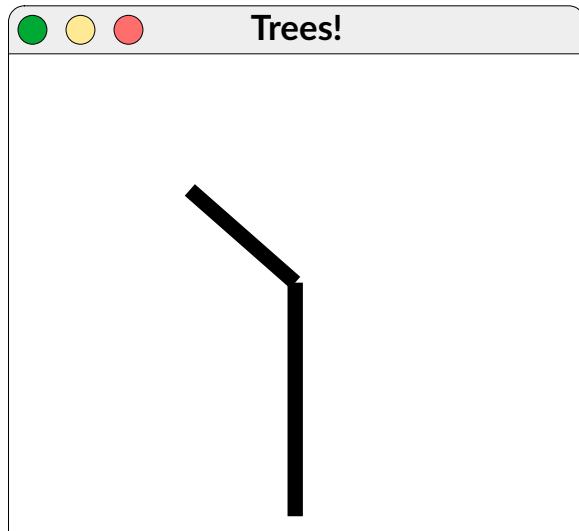
```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;  
}
```

1

1

numLines



```
int drawTree(/* ... */) {
```

```
    if (order == 0) {  
        return 0;  
    }
```

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

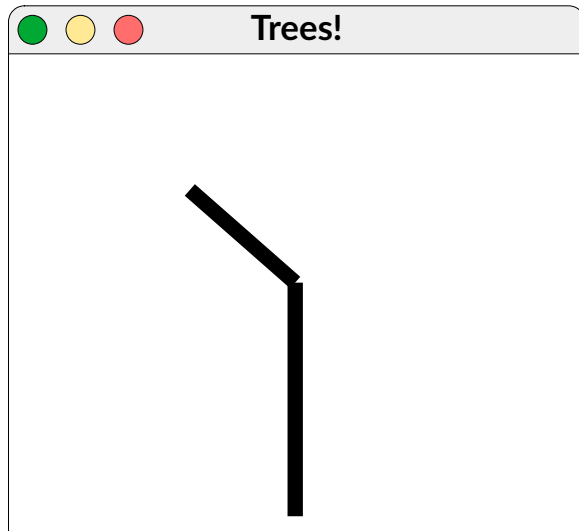
```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;  
}
```

1

1

numLines



```
int drawTree(/* ... */) {
```

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
    }
```

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

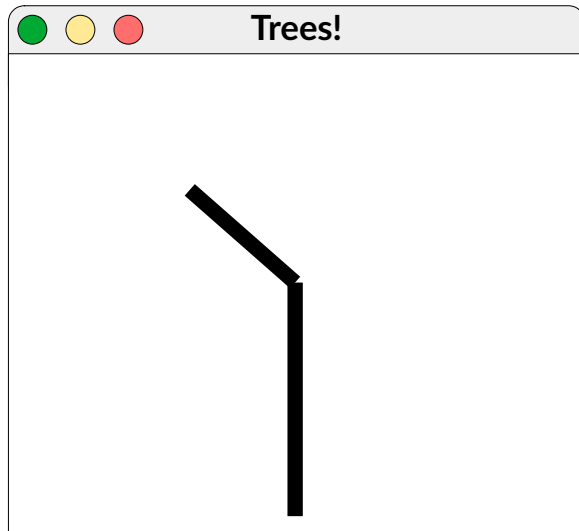
```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;  
}
```

1

1

numLines



This call draws a recursive tree.

```
int drawTree(/* ... */) {
```

```
    if (order == 0) {  
        return 0;  
    }
```

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

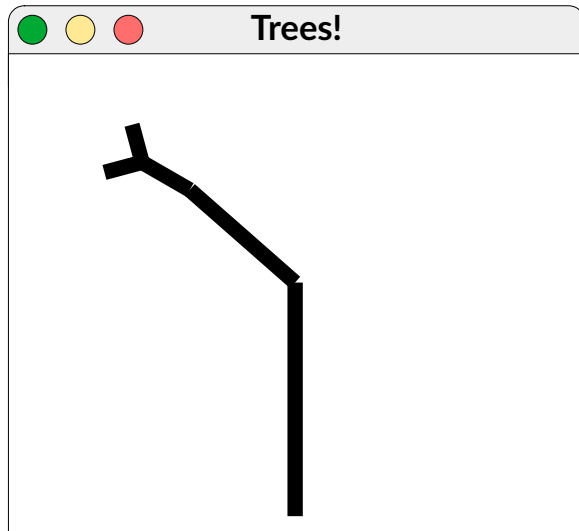
```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;  
}
```

1

1

numLines



This call draws a recursive tree.

```
int drawTree(/* ... */) {
```

```
    if (order == 0) {  
        return 0;  
    }
```

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

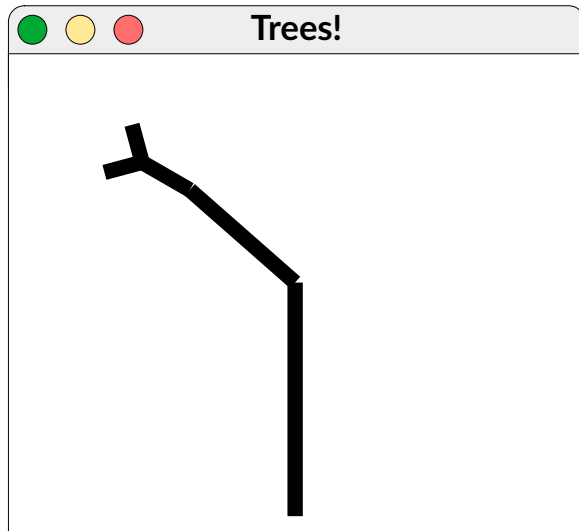
```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;  
}
```

1

1

numLines



This call draws a recursive tree.

It then returns the number of lines drawn.

```
int drawTree(/* ... */) {
```

1

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

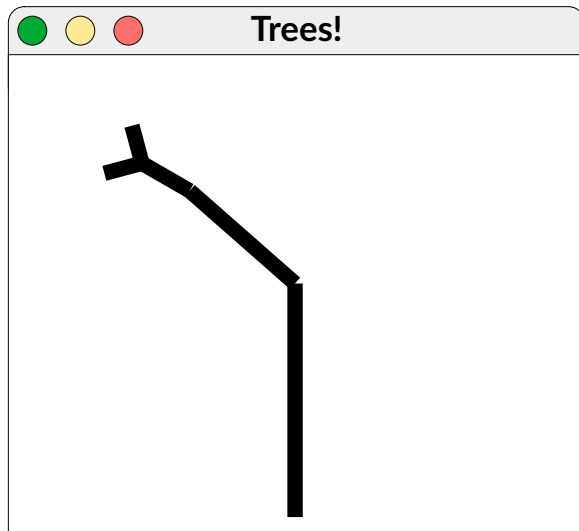
1

numLines

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;  
}
```



```
int drawTree(/* ... */) {
```

1

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
    }
```

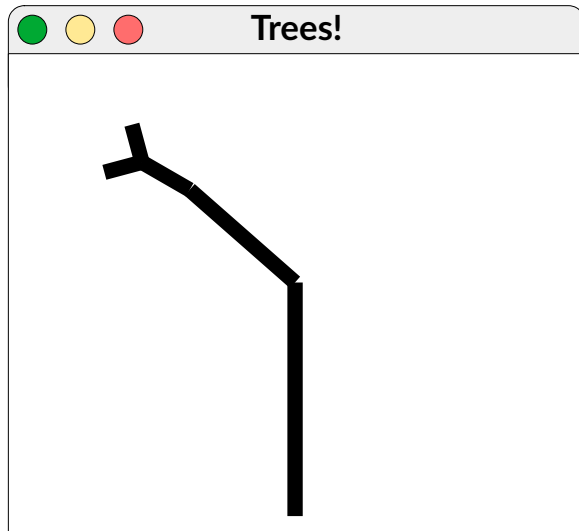
4

numLines

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;  
}
```



```
int drawTree(/* ... */) {
```

1

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }
```

4

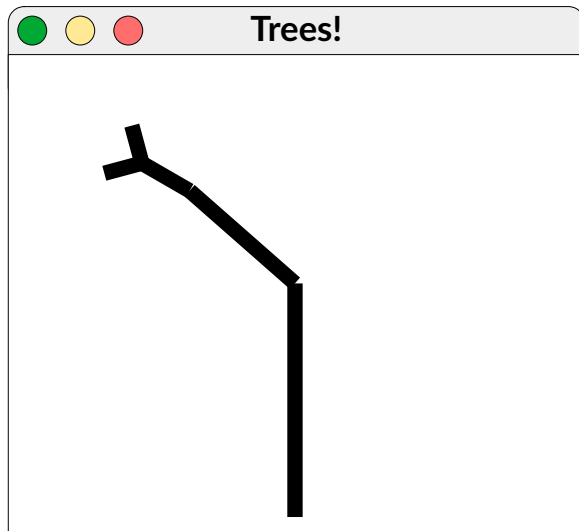
numLines

```
        int numLines = 0;  
        GPoint endpoint = drawPolarLine(/* ... */);  
        numLines++;
```

```
        numLines += drawTree(/* ... */);  
        numLines += drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
    int drawTree(/* ... */) {  
        if (order == 0) {  
            return 0;  
        }  
    }
```

4

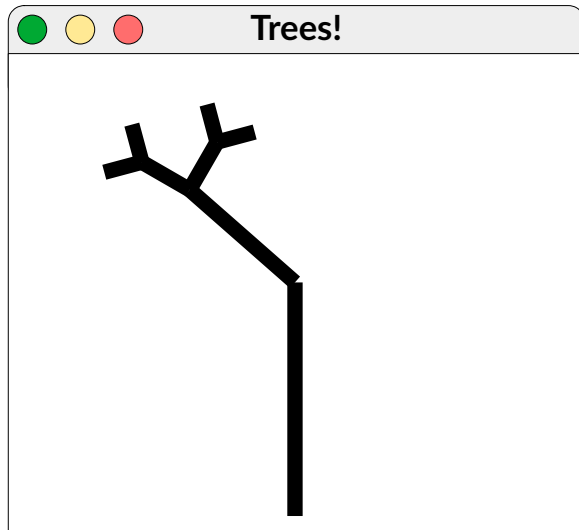
numLines

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

7

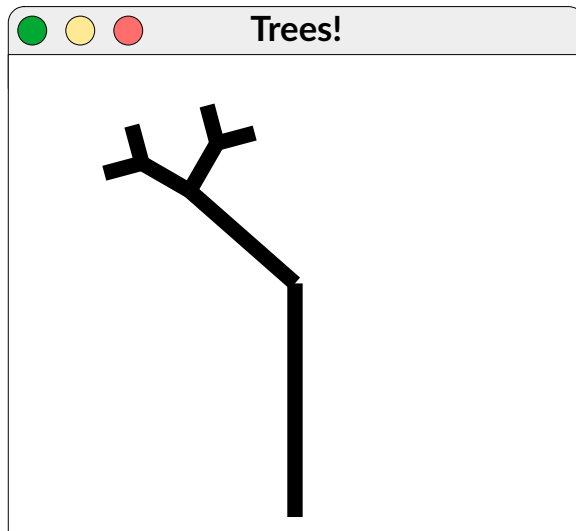
numLines

```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

```
    return numLines;
```

```
}
```



```
int drawTree(/* ... */) {
```

1

```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

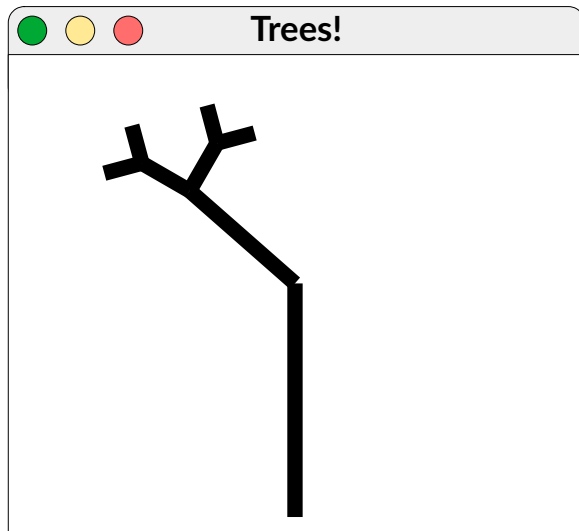
7

numLines

```
int numLines = 0;  
GPoint endpoint = drawPolarLine(/* ... */);  
numLines++;
```

```
numLines += drawTree(/* ... */);  
numLines += drawTree(/* ... */);
```

```
return numLines;
```



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }
```

1

numLines

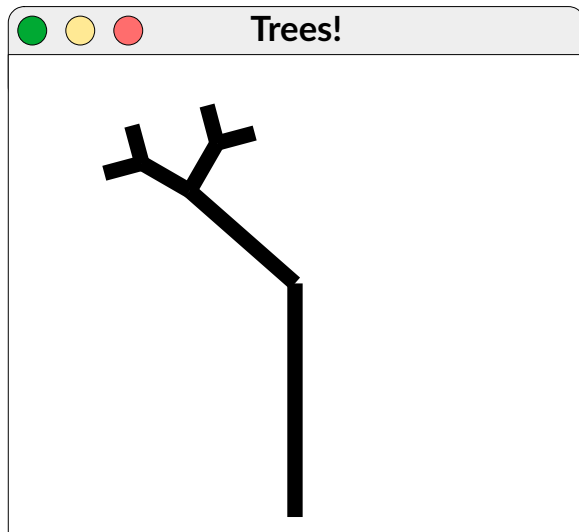
```
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;
```

```
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);
```

7

```
    return numLines;
```

```
}
```



```

int drawTree(/* ... */) {
    if (order == 0) {
        return 0;
    }

    int numLines = 0;
    GPoint endpoint = drawPolarLine(/* ... */);
    numLines++;

    numLines += drawTree(/* ... */);
    numLines += drawTree(/* ... */);

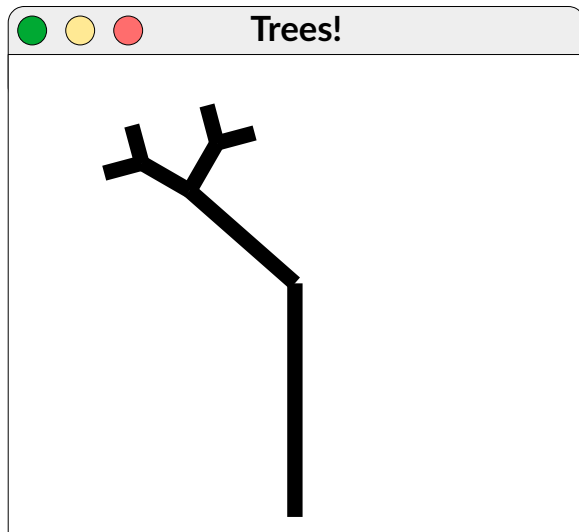
    return numLines;
}

```

1

numLines

7



```

int drawTree(/* ... */) {
    if (order == 0) {
        return 0;
    }

    int numLines = 0;
    GPoint endpoint = drawPolarLine(/* ... */);
    numLines++;

    numLines += drawTree(/* ... */);
    numLines += drawTree(/* ... */);

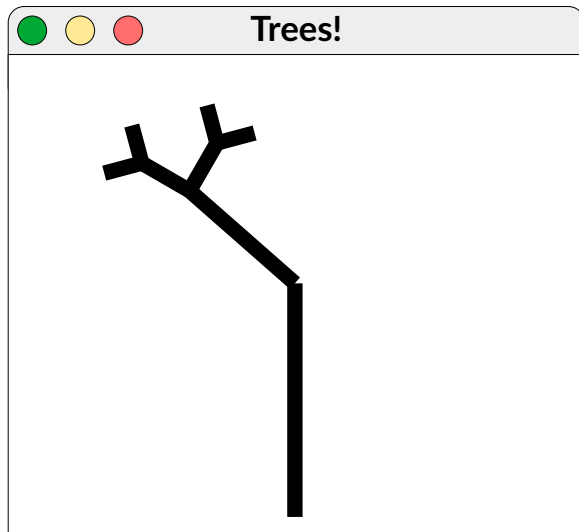
    return numLines;
}

```

8

numLines

7



```

int drawTree(/* ... */) {
    if (order == 0) {
        return 0;
    }

    int numLines = 0;
    GPoint endpoint = drawPolarLine(/* ... */);
    numLines++;

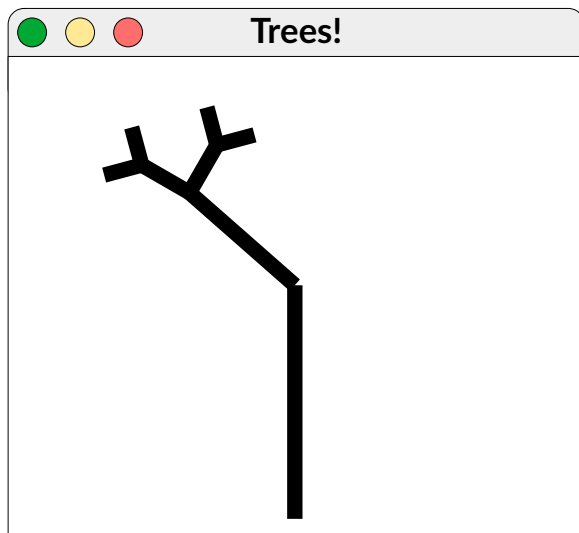
    numLines += drawTree(/* ... */);
    numLines += drawTree(/* ... */);

    return numLines;
}

```

8

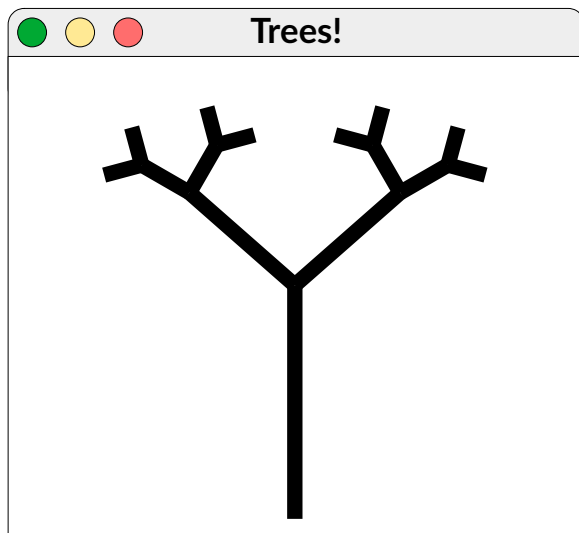
numLines



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);  
  
    return numLines;  
}
```

8

numLines



```

int drawTree(/* ... */) {
    if (order == 0) {
        return 0;
    }

    int numLines = 0;
    GPoint endpoint = drawPolarLine(/* ... */);
    numLines++;

    numLines += drawTree(/* ... */);
    numLines += drawTree(/* ... */);

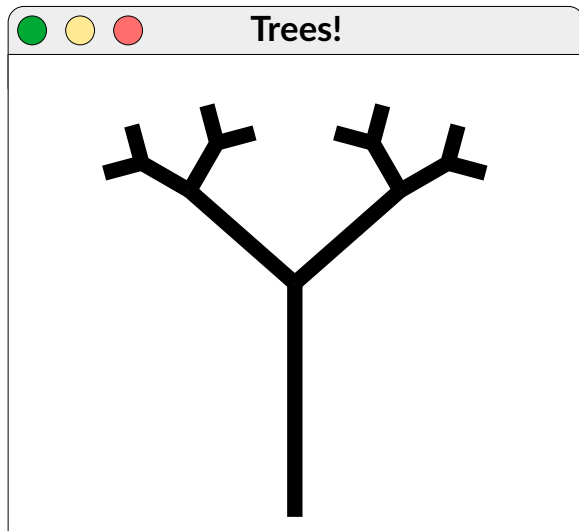
    return numLines;
}

```

8

numLines

7



```

int drawTree(/* ... */) {
    if (order == 0) {
        return 0;
    }

    int numLines = 0;
    GPoint endpoint = drawPolarLine(/* ... */);
    numLines++;

    numLines += drawTree(/* ... */);
    numLines += drawTree(/* ... */);

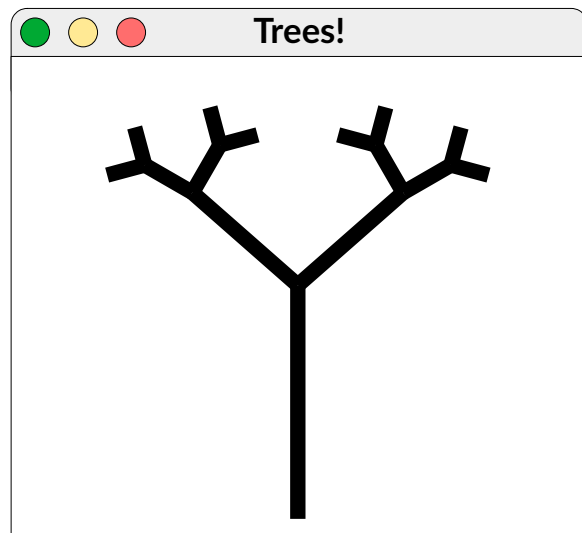
    return numLines;
}

```

15

numLines

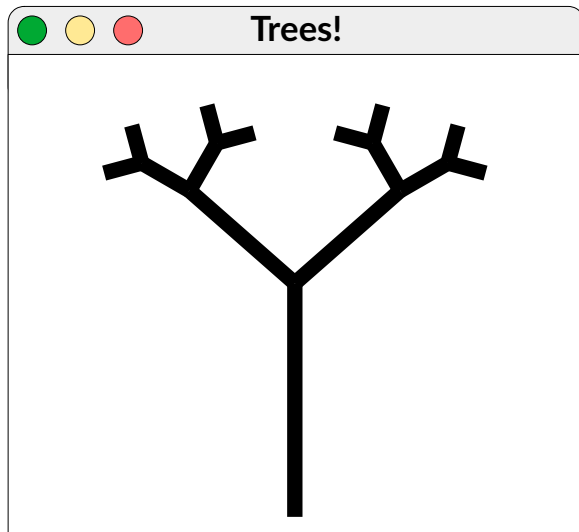
7



```
int drawTree(/* ... */) {  
    if (order == 0) {  
        return 0;  
    }  
  
    int numLines = 0;  
    GPoint endpoint = drawPolarLine(/* ... */);  
    numLines++;  
  
    numLines += drawTree(/* ... */);  
    numLines += drawTree(/* ... */);  
  
    return numLines;  
}
```

15

numLines



Summary From Today

- Self-similar figures exist in many places, and recursion is a great way to draw them.
- When drawing a self-similar figure, identify what aspects of the figure are different at different scales.
- Assigning an order to a self-similar figure is a great way to make a base case.
- When writing a recursive function that returns a value, make sure you use the result of each recursive call. Otherwise, important data can get lost.

Your Action Items

- ***Read Chapter 8.***
 - There's a ton of goodies in there! It'll help you solidify your understanding of recursion and recursive techniques.
- ***Keep Working On Assignment 2.***
 - Need help? Stop by the LaIR or post on EdStem! That's what we're here for.

Next Time

- ***Recursive Enumeration***
 - Finding all objects of a given type.
- ***Enumerating Subsets***
 - A classic combinatorial problem!