# Thinking Recursively

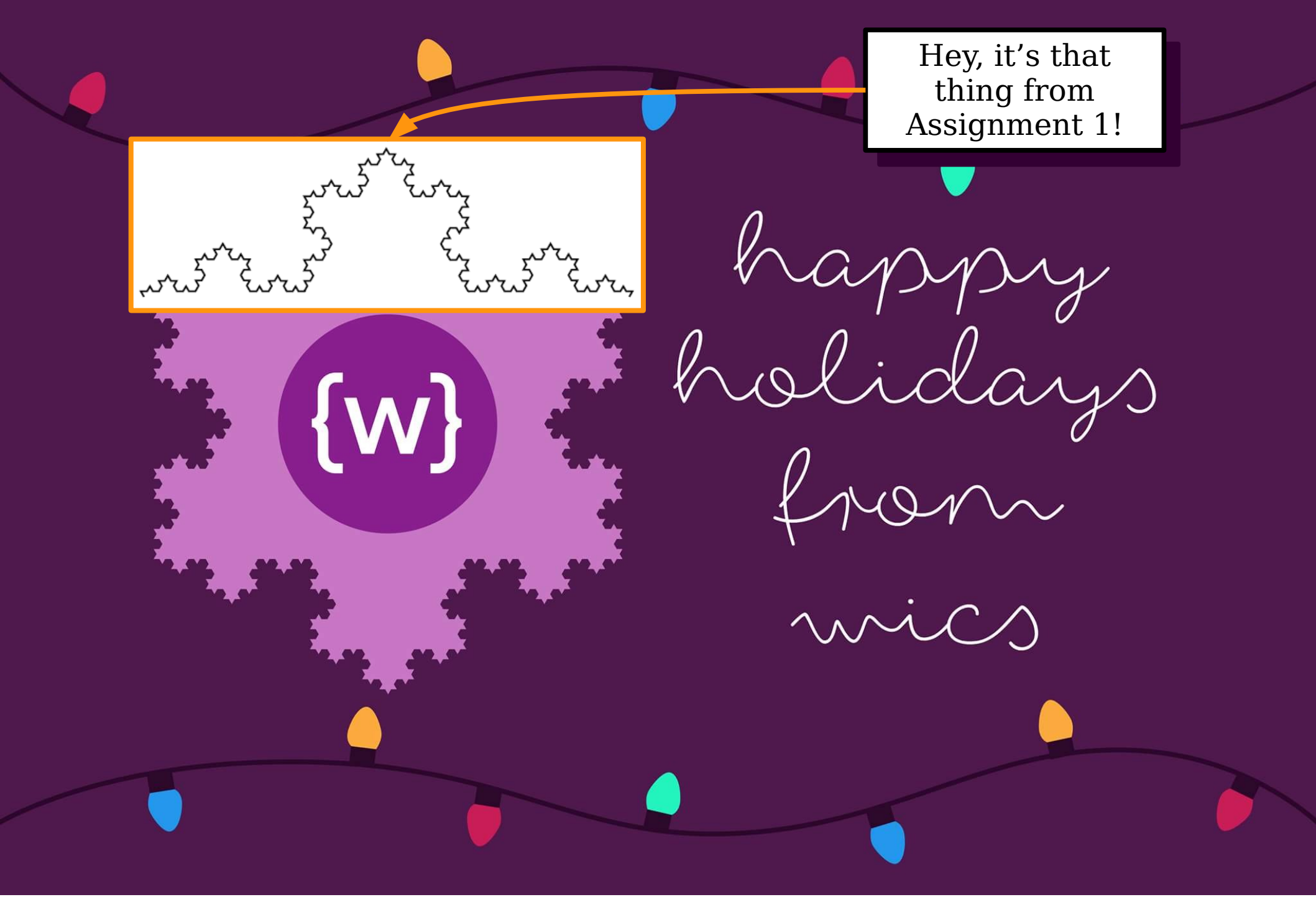
## Part I

# Outline for Today

- ***Self-Similarity***
  - Recursive patterns are everywhere!
- ***Wrapper Functions***
  - Hiding recursion from clients.
- ***Recursive Enumeration***
  - Listing all solutions to a problem.
- ***Decision Trees***
  - A powerful framework.

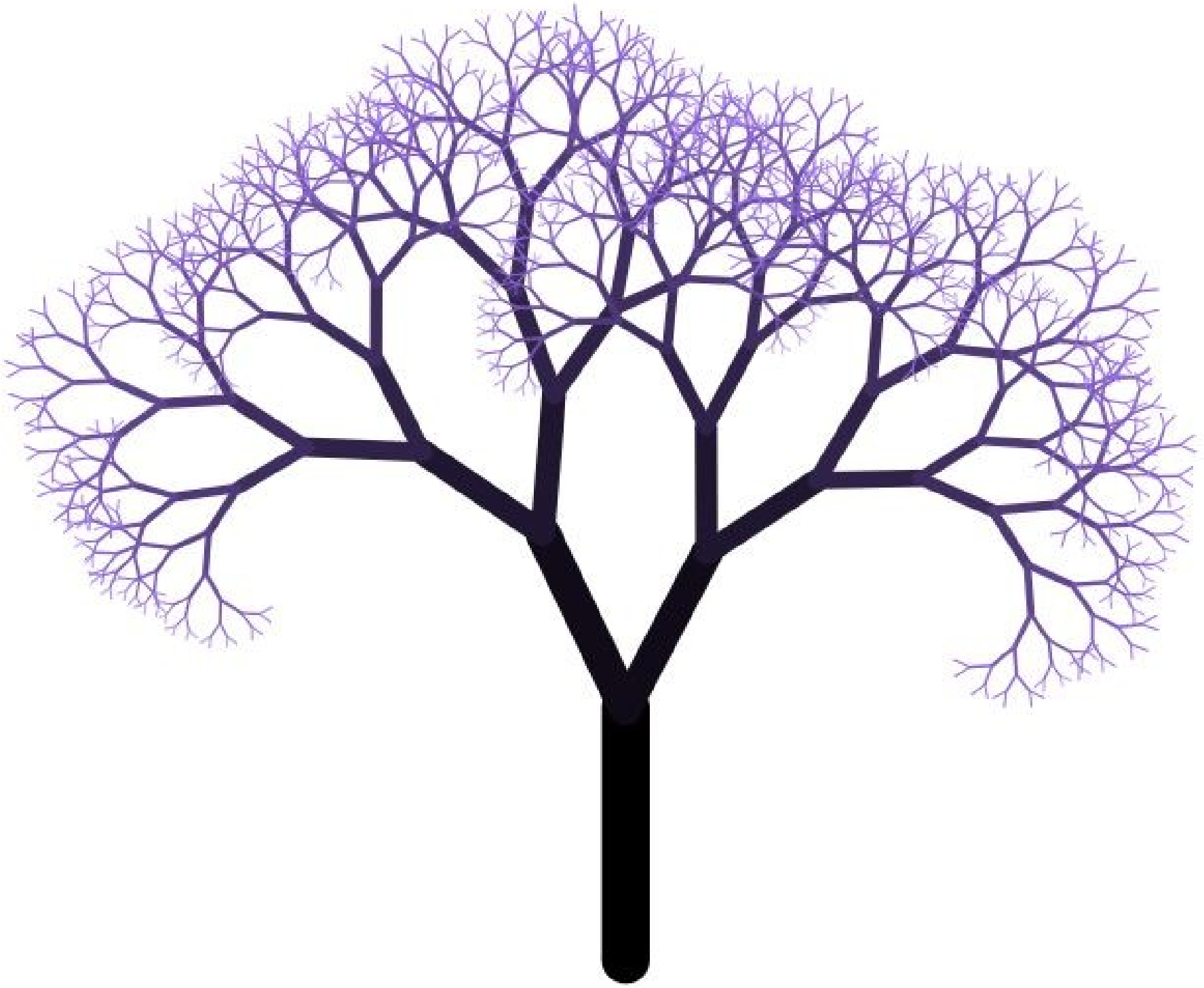
# Self-Similarity

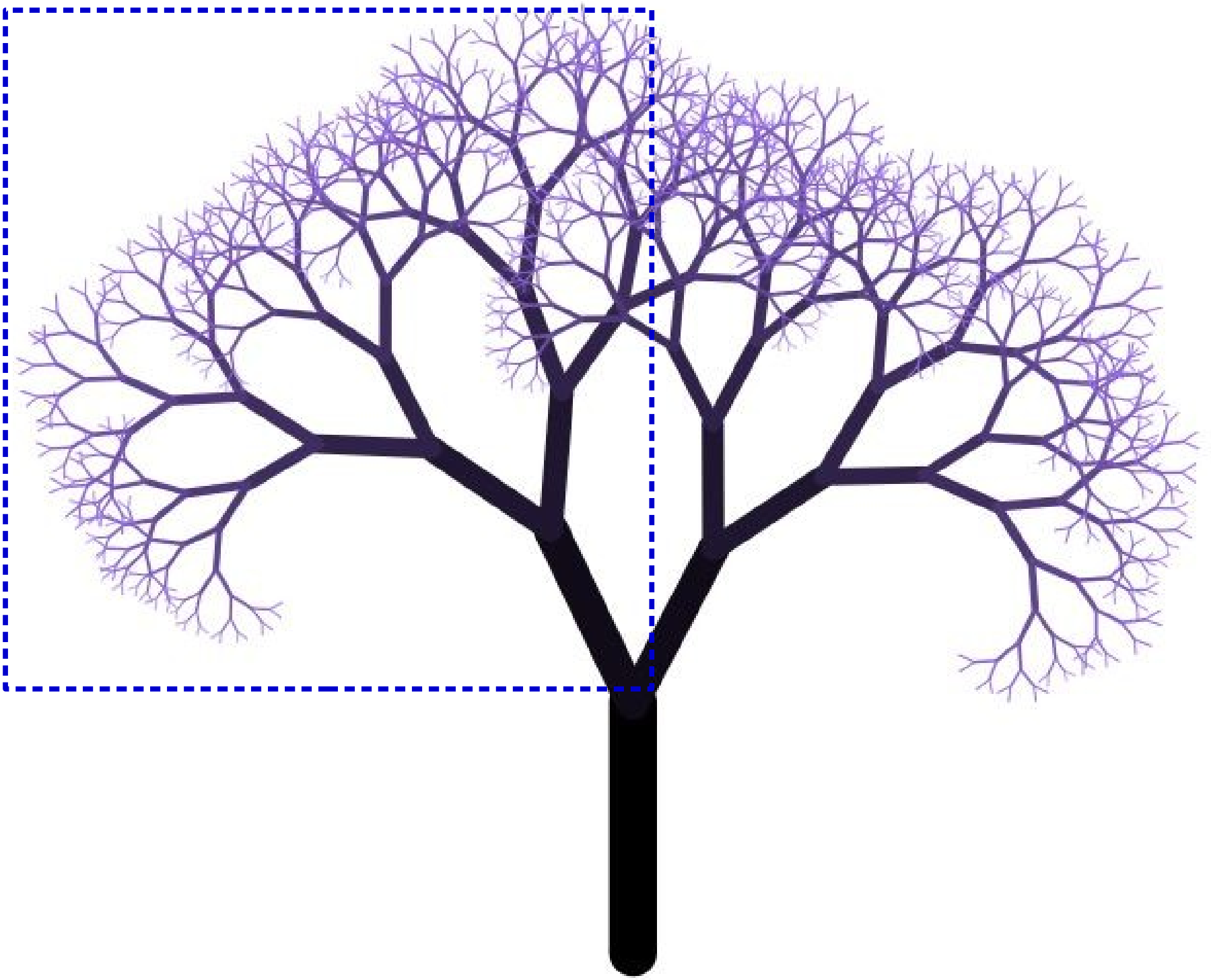
An object is ***self-similar*** if it contains a smaller copy of itself.

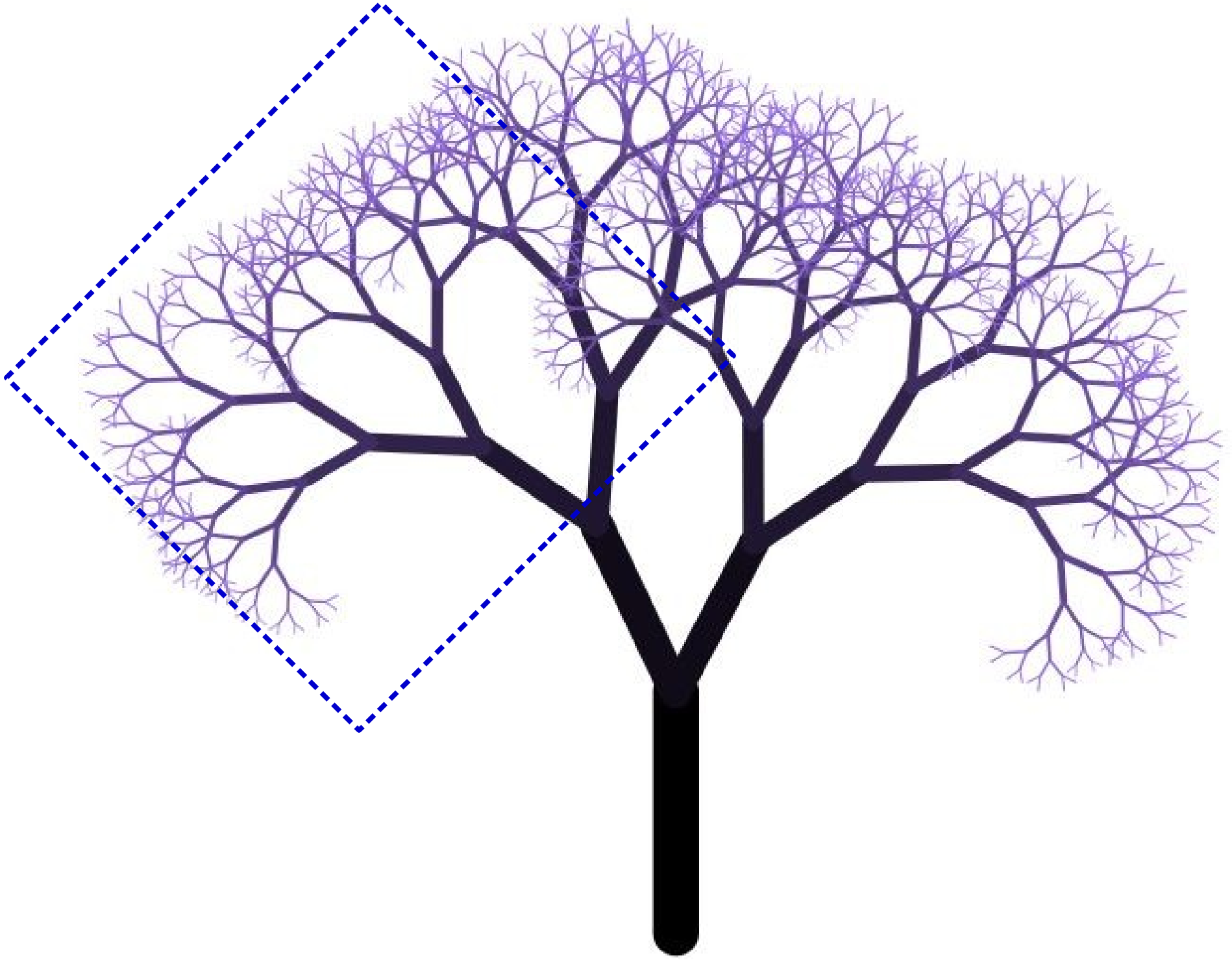


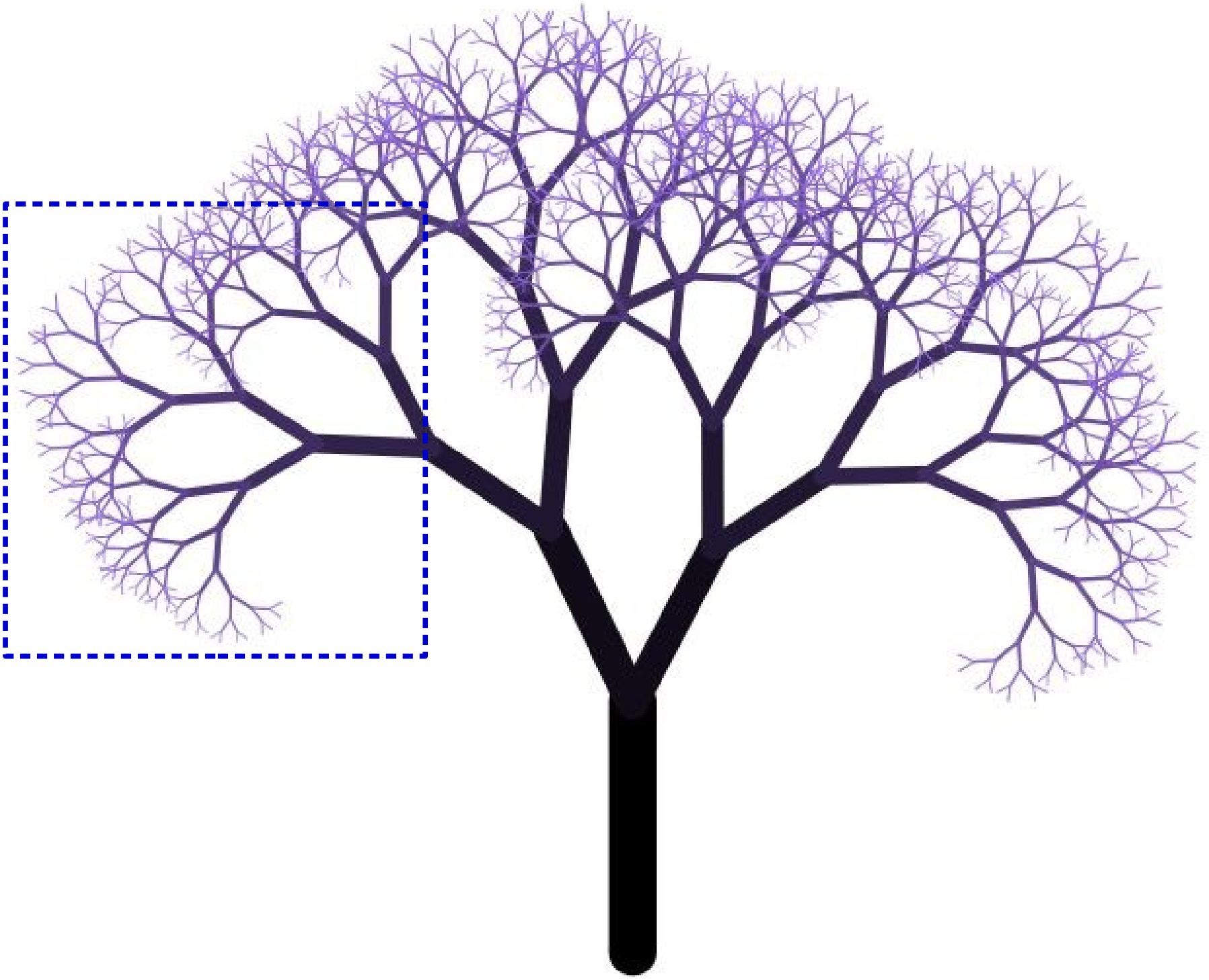

An object is ***self-similar*** if it contains a smaller copy of itself.

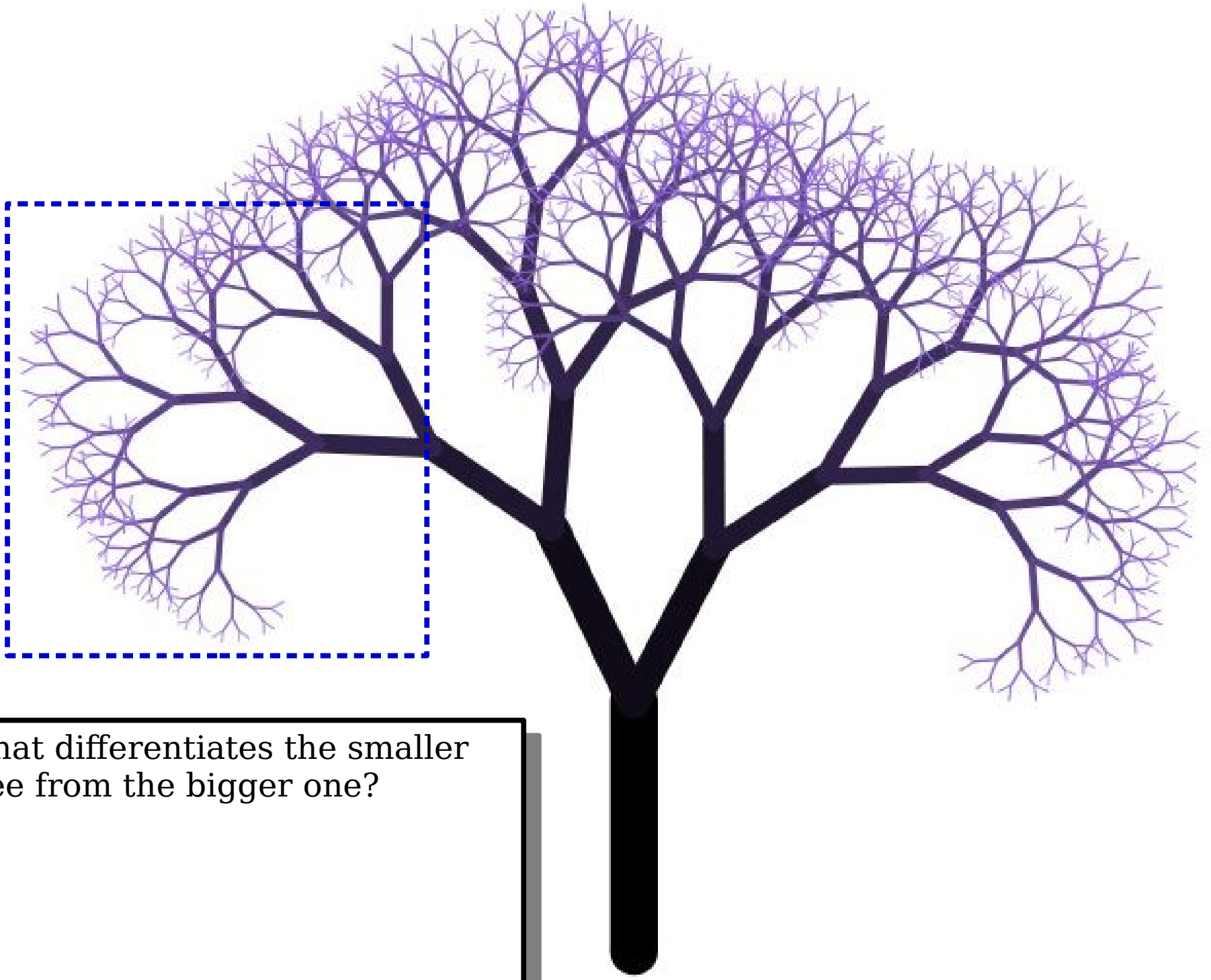
# Drawing Self-Similar Shapes



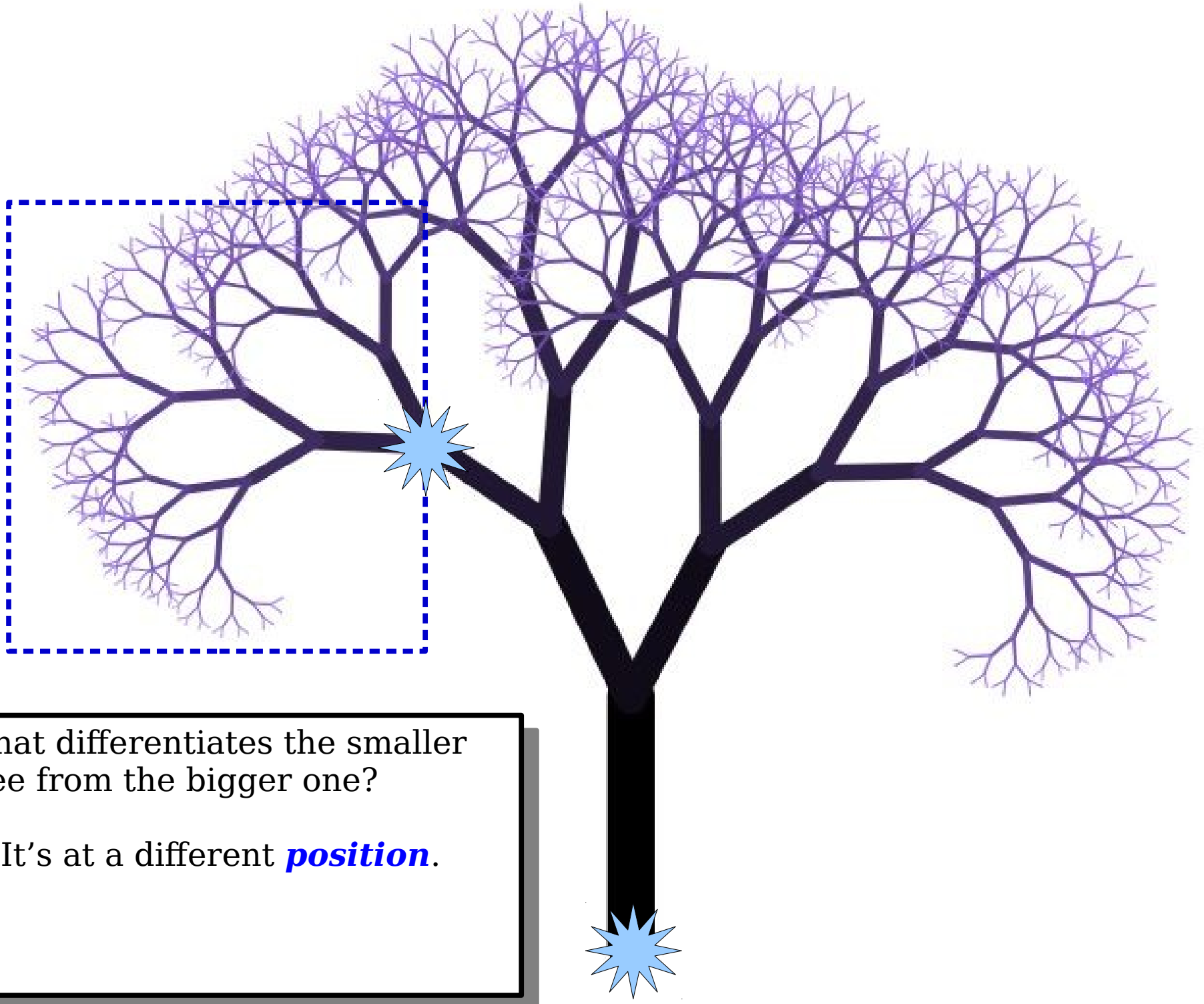






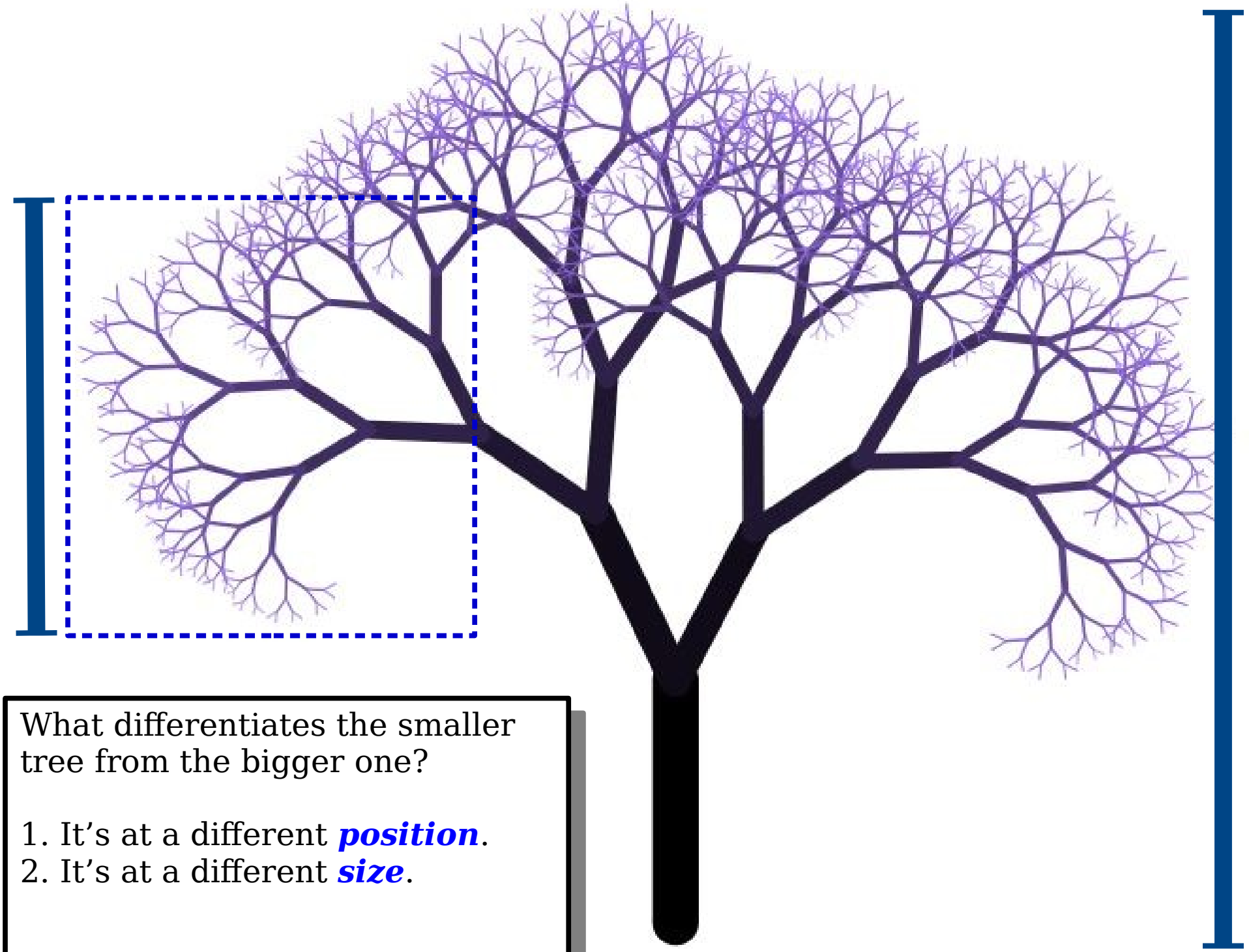


What differentiates the smaller tree from the bigger one?



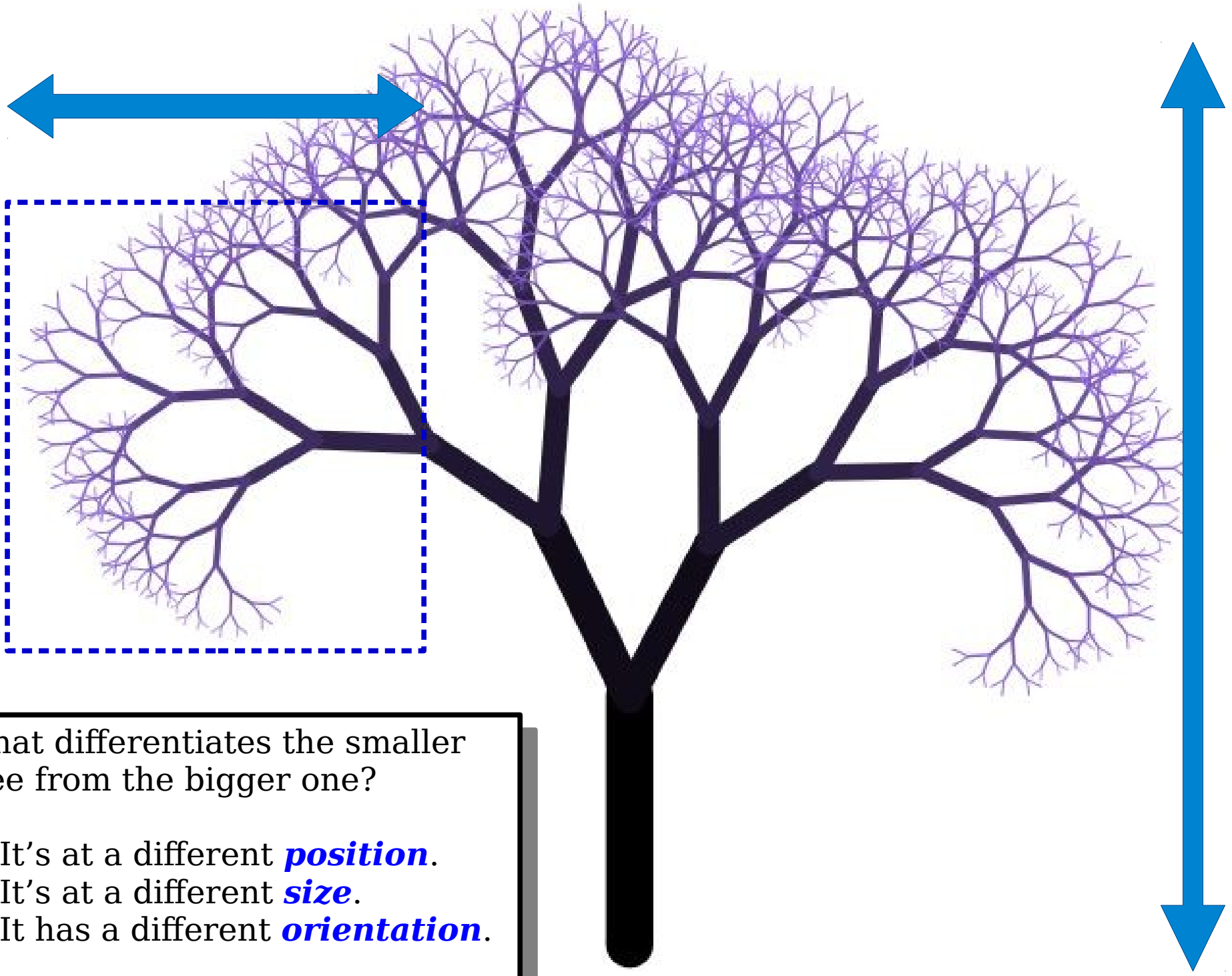
What differentiates the smaller tree from the bigger one?

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



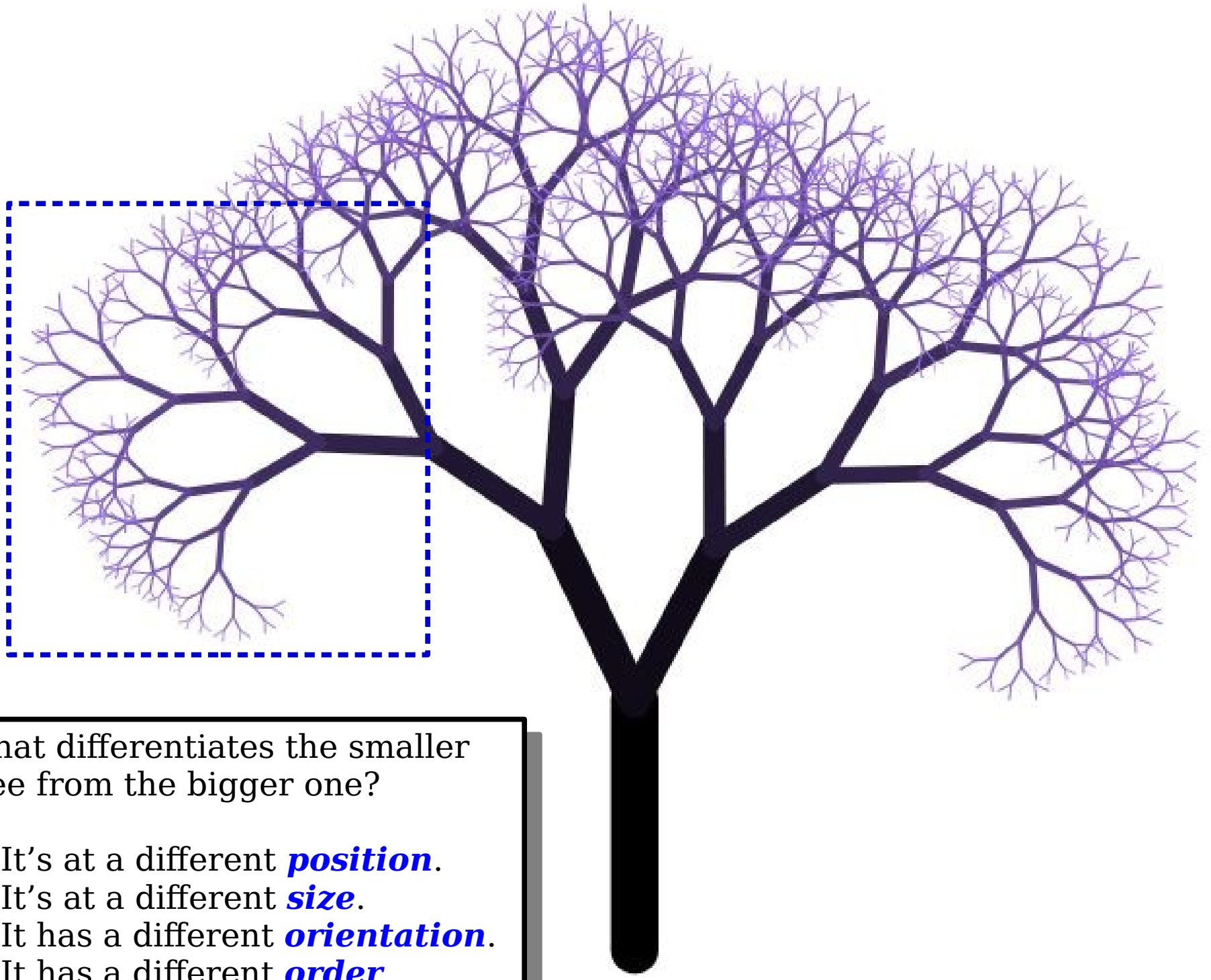
What differentiates the smaller tree from the bigger one?

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



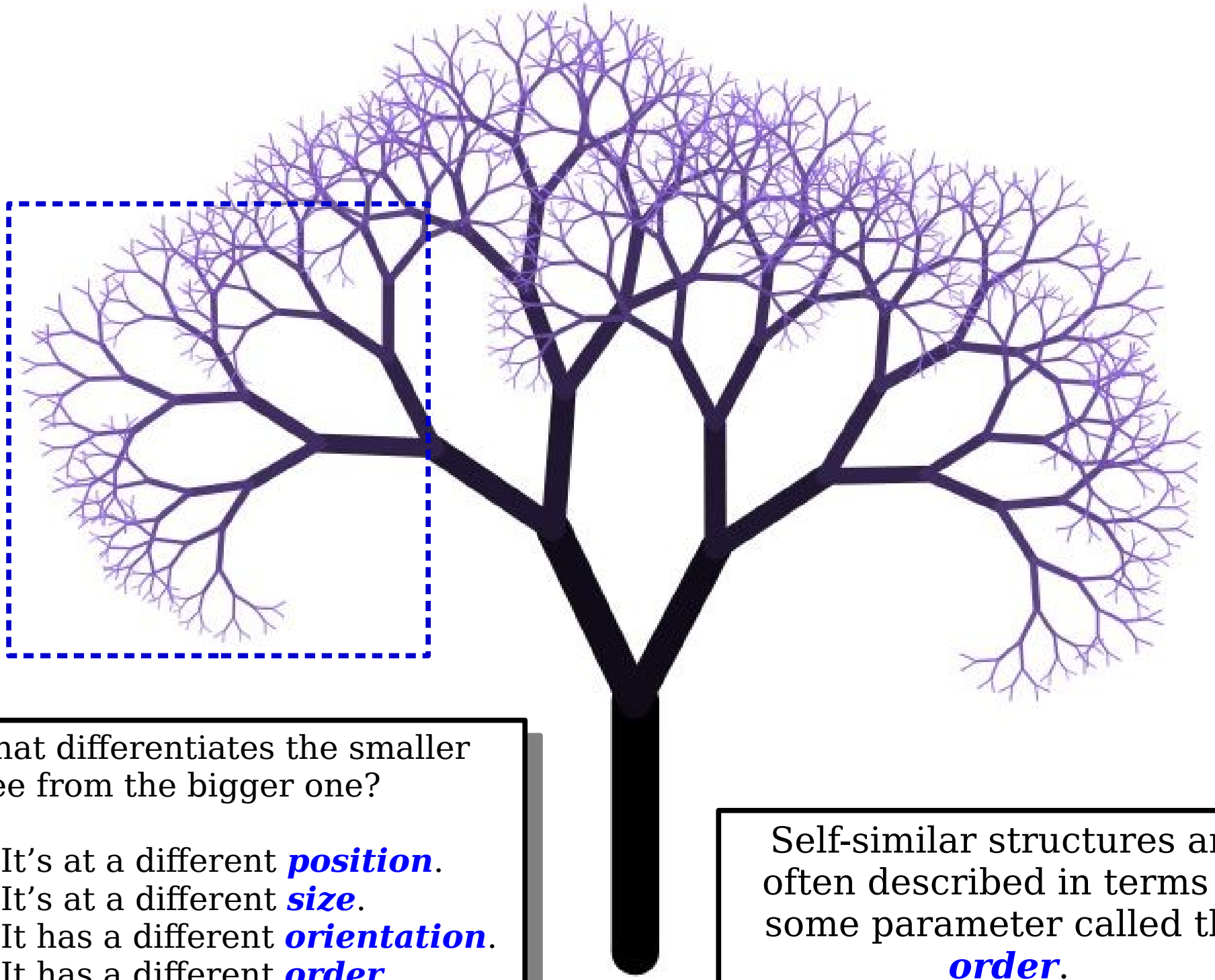
What differentiates the smaller tree from the bigger one?

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

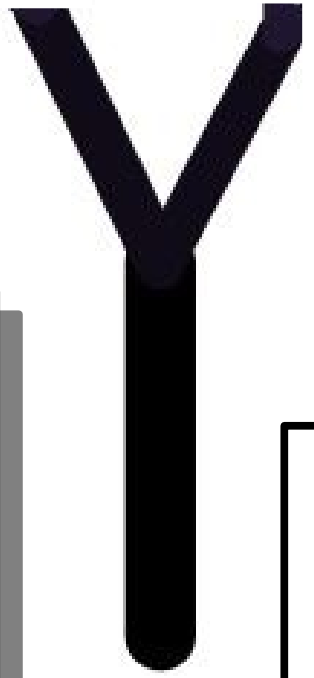
# *An order-1 tree.*

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.*

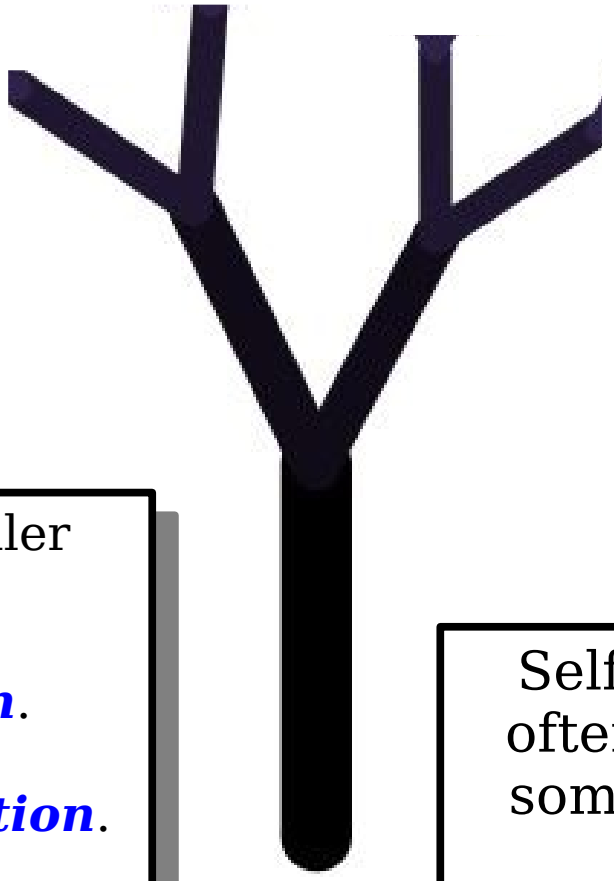


What differentiates the smaller tree from the bigger one?

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.*

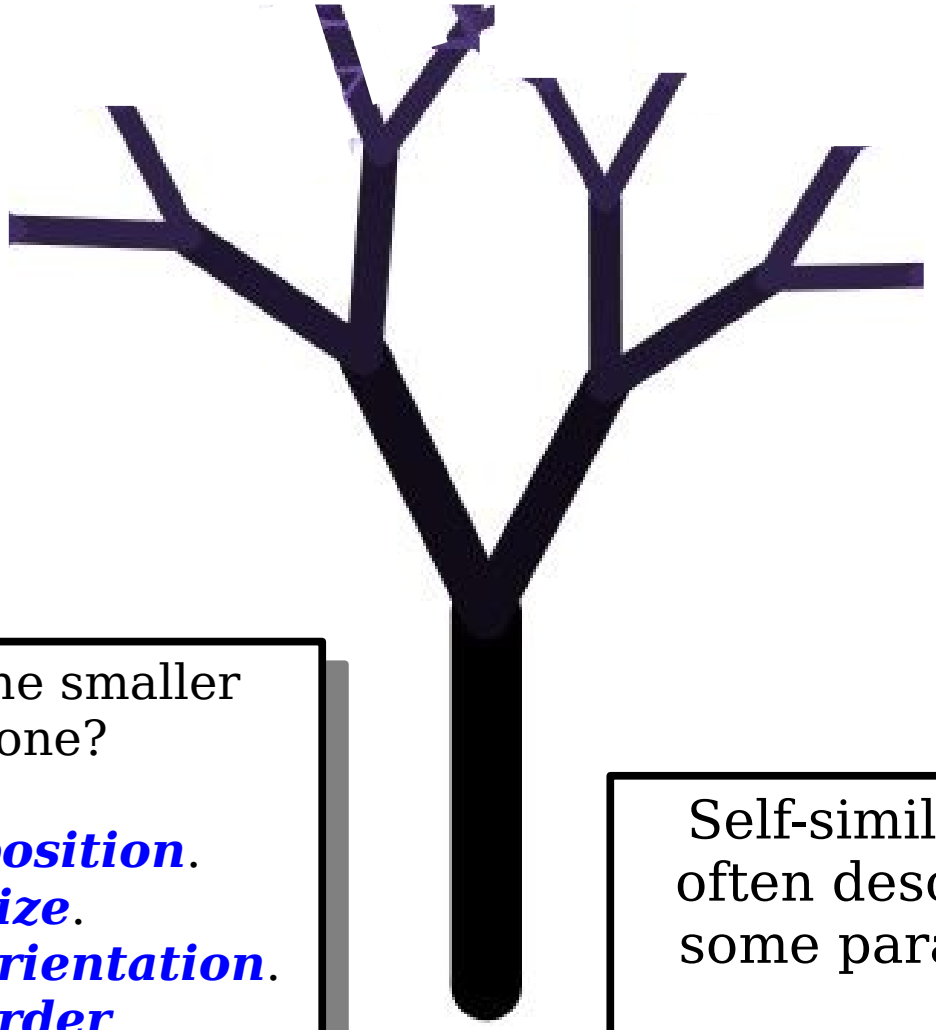


What differentiates the smaller tree from the bigger one?

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.*

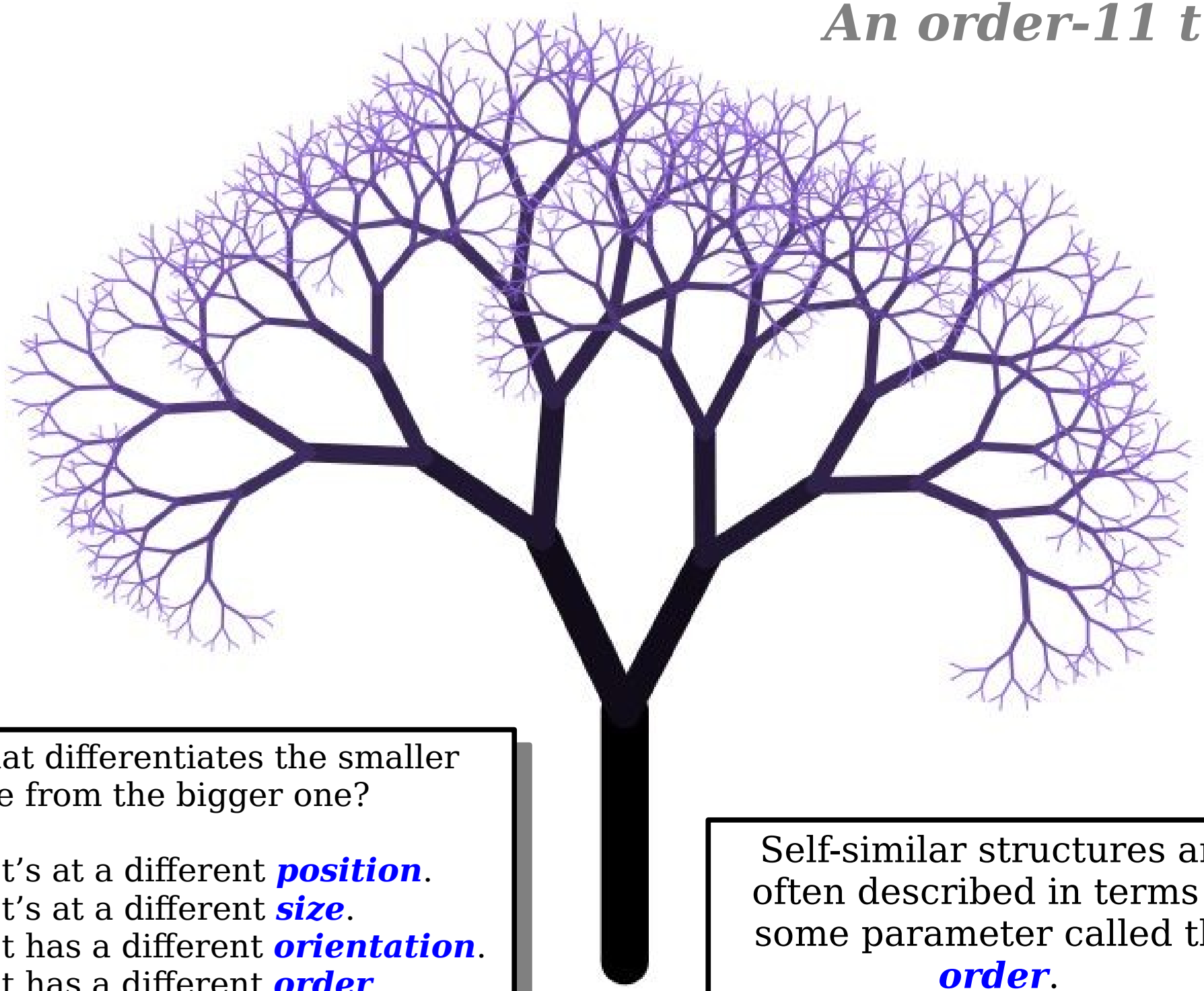


What differentiates the smaller tree from the bigger one?

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.*

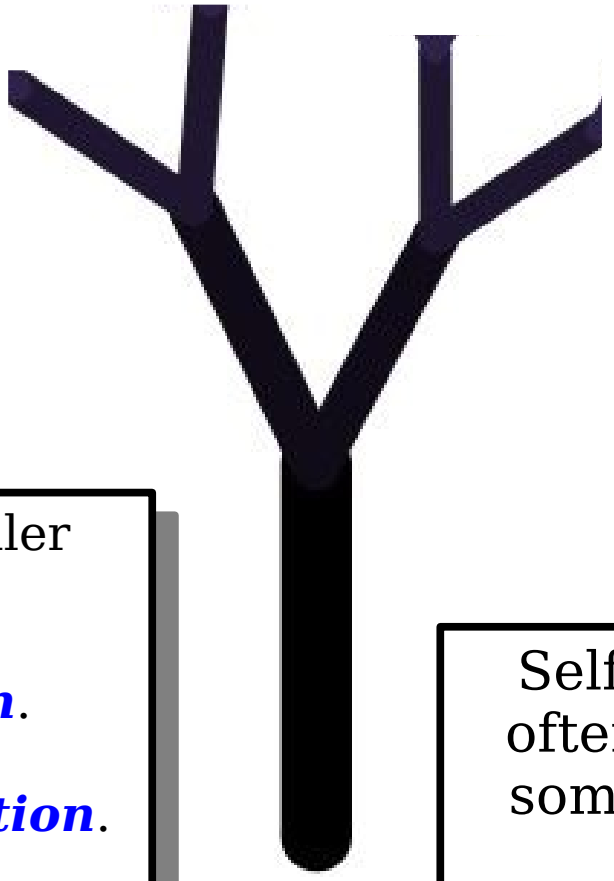


What differentiates the smaller tree from the bigger one?

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?

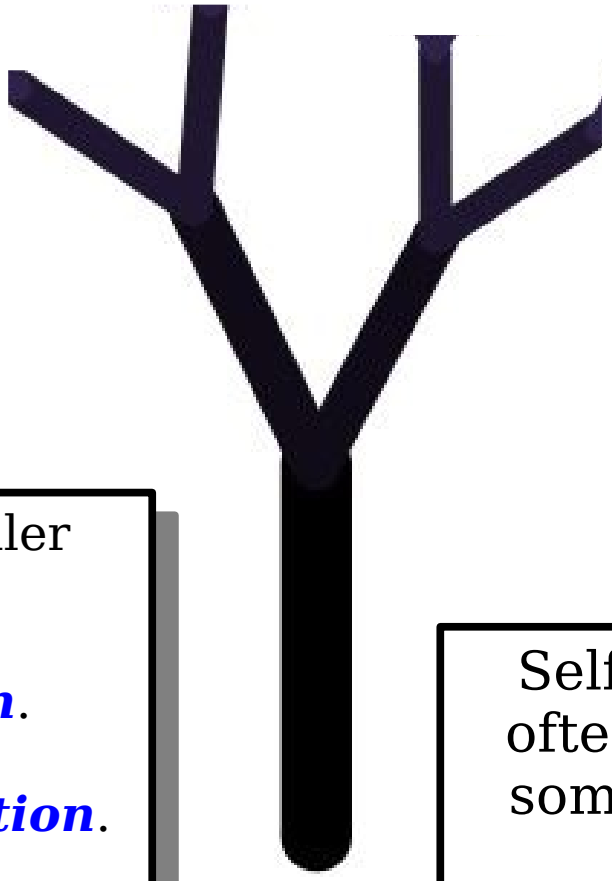
1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

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?

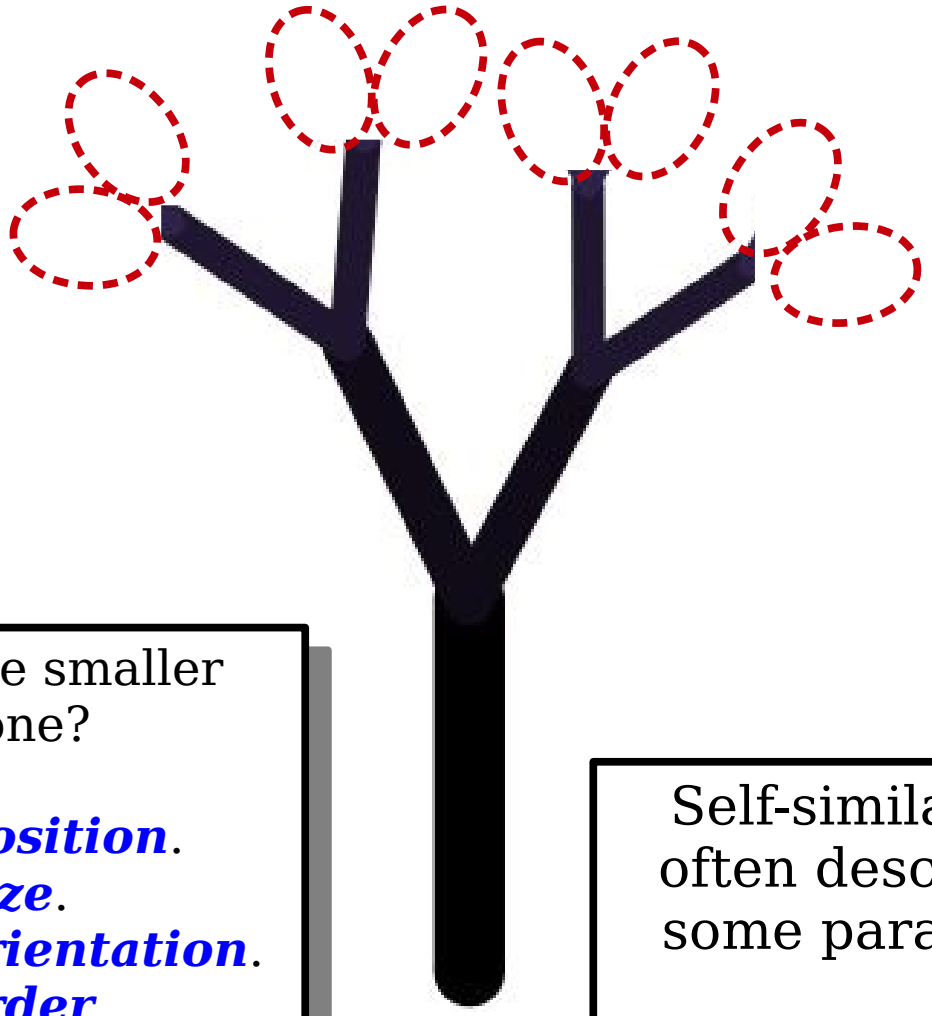
1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

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***.

# An order-3 tree.

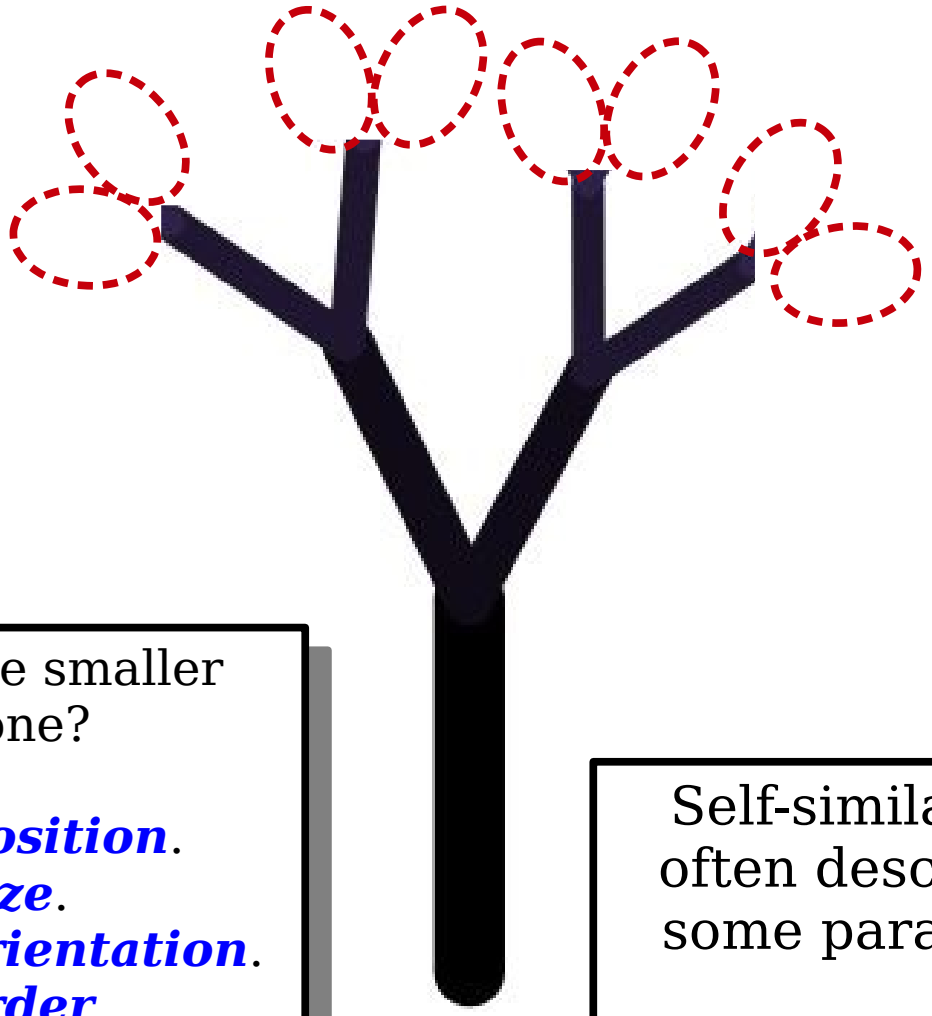
An order-0 tree is nothing at all.

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

We can draw lines in the window by calling

```
window.drawPolarLine(x, y, r, θ);
```

with θ specified in degrees.



What differentiates the smaller tree from the bigger one?

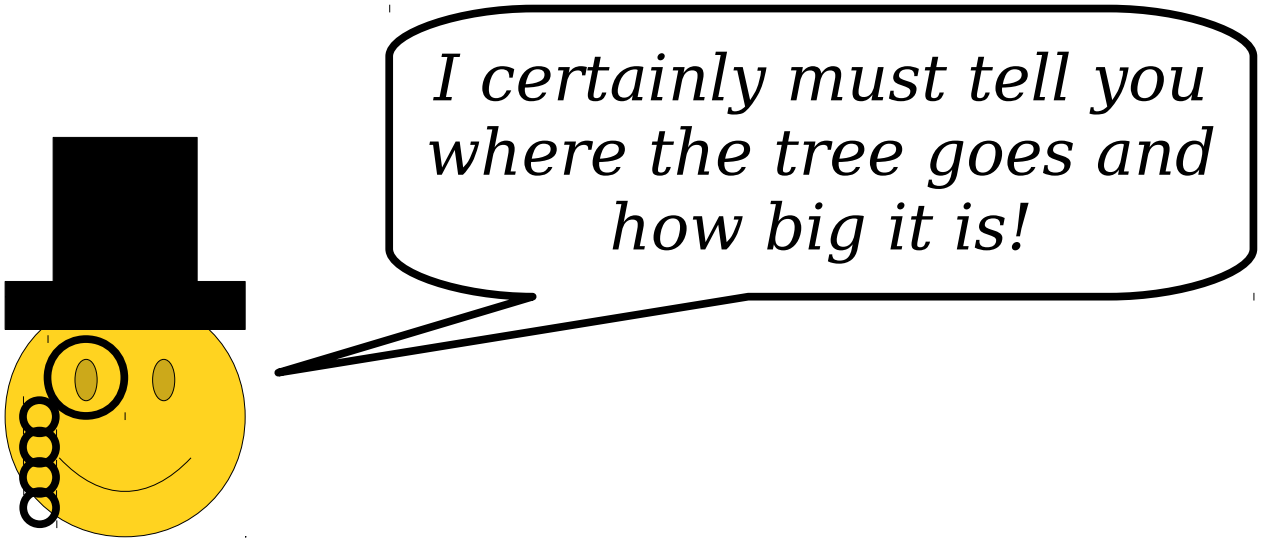
1. It's at a different ***position***.
2. It's at a different ***size***.
3. It has a different ***orientation***.
4. It has a different ***order***.

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

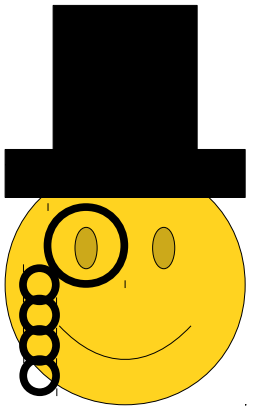
```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

drawTree(window,
         treeRootX, treeRootY,
         treeHeight,
         90, 8);
```

```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

drawTree(window,
         treeRootX, treeRootY,
         treeHeight,
         90, 8);
```

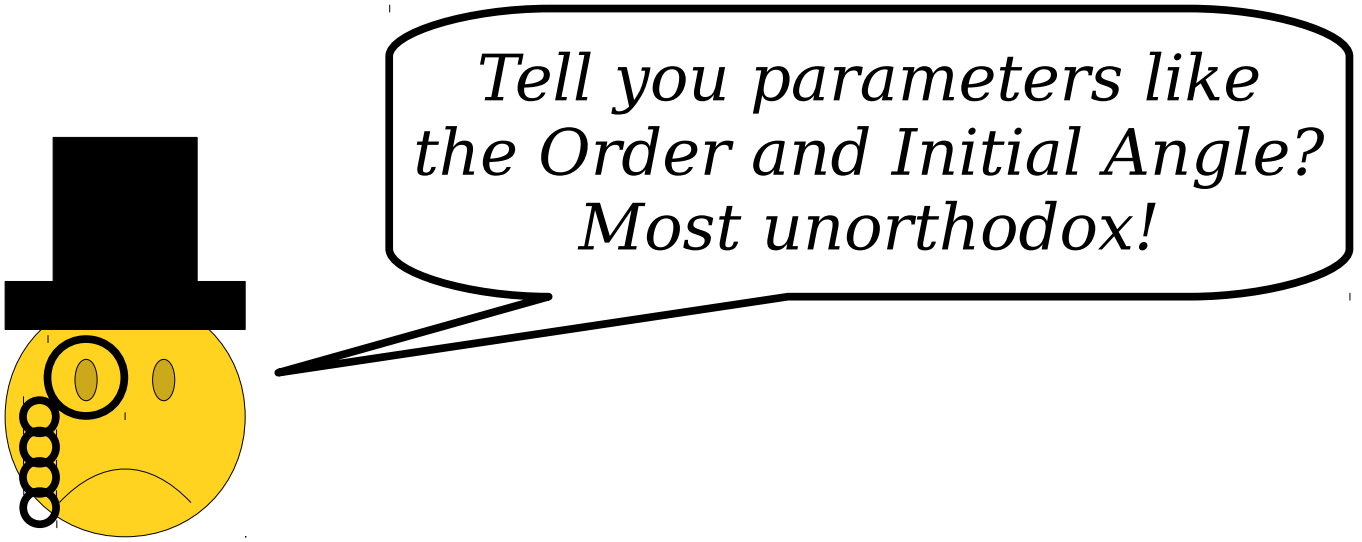
```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

drawTree(window,
         treeRootX, treeRootY,
         treeHeight,
         90, 8);
```



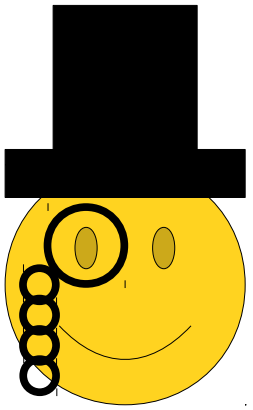
```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

drawTree(window,
         treeRootX, treeRootY,
         treeHeight,
         90, 8);
```



```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

drawTree(window,
         treeRootX, treeRootY,
         treeHeight,
         90, 8);
```



```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

drawTree(window,
         treeRootX, treeRootY,
         treeHeight);
```

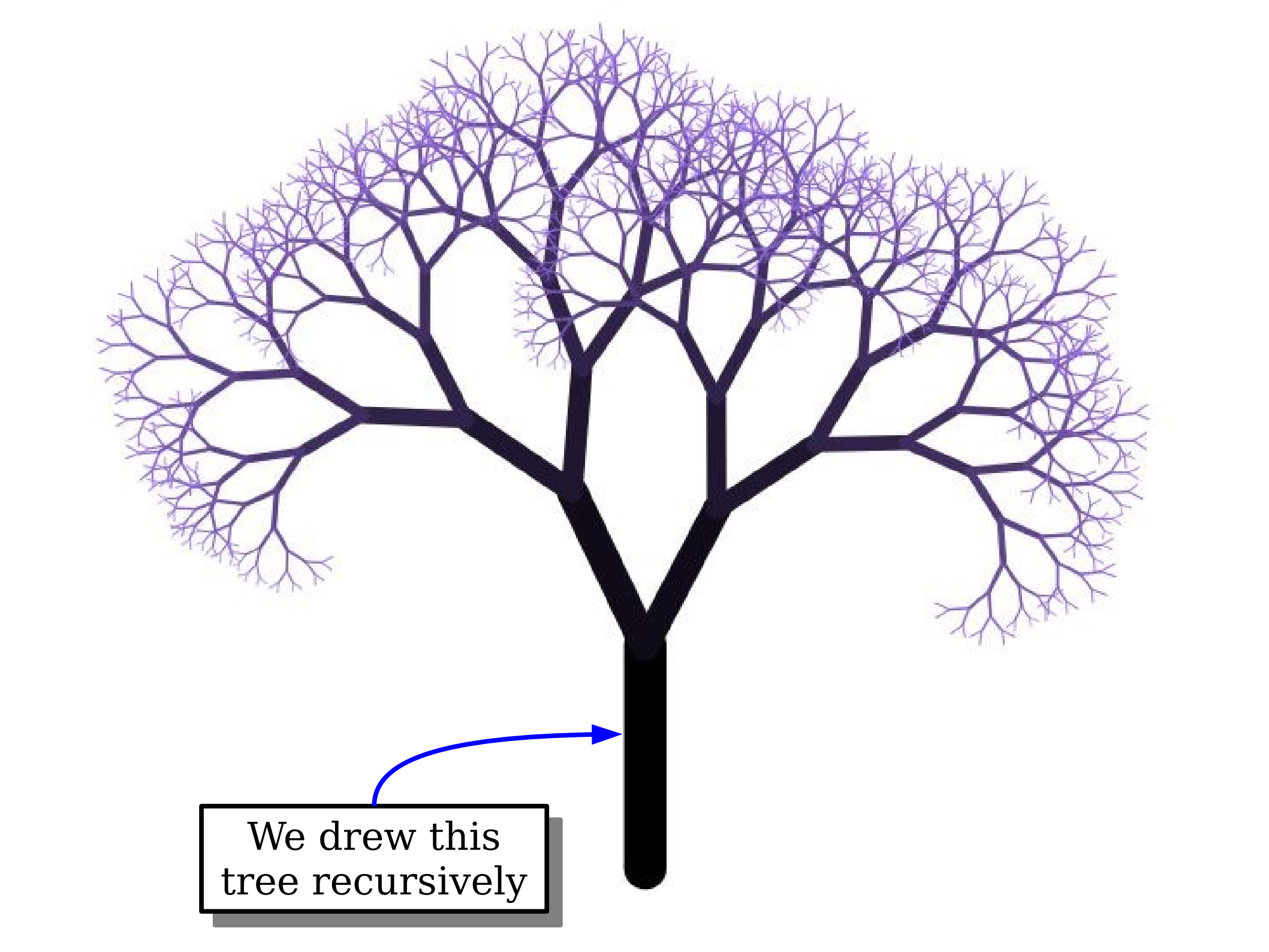


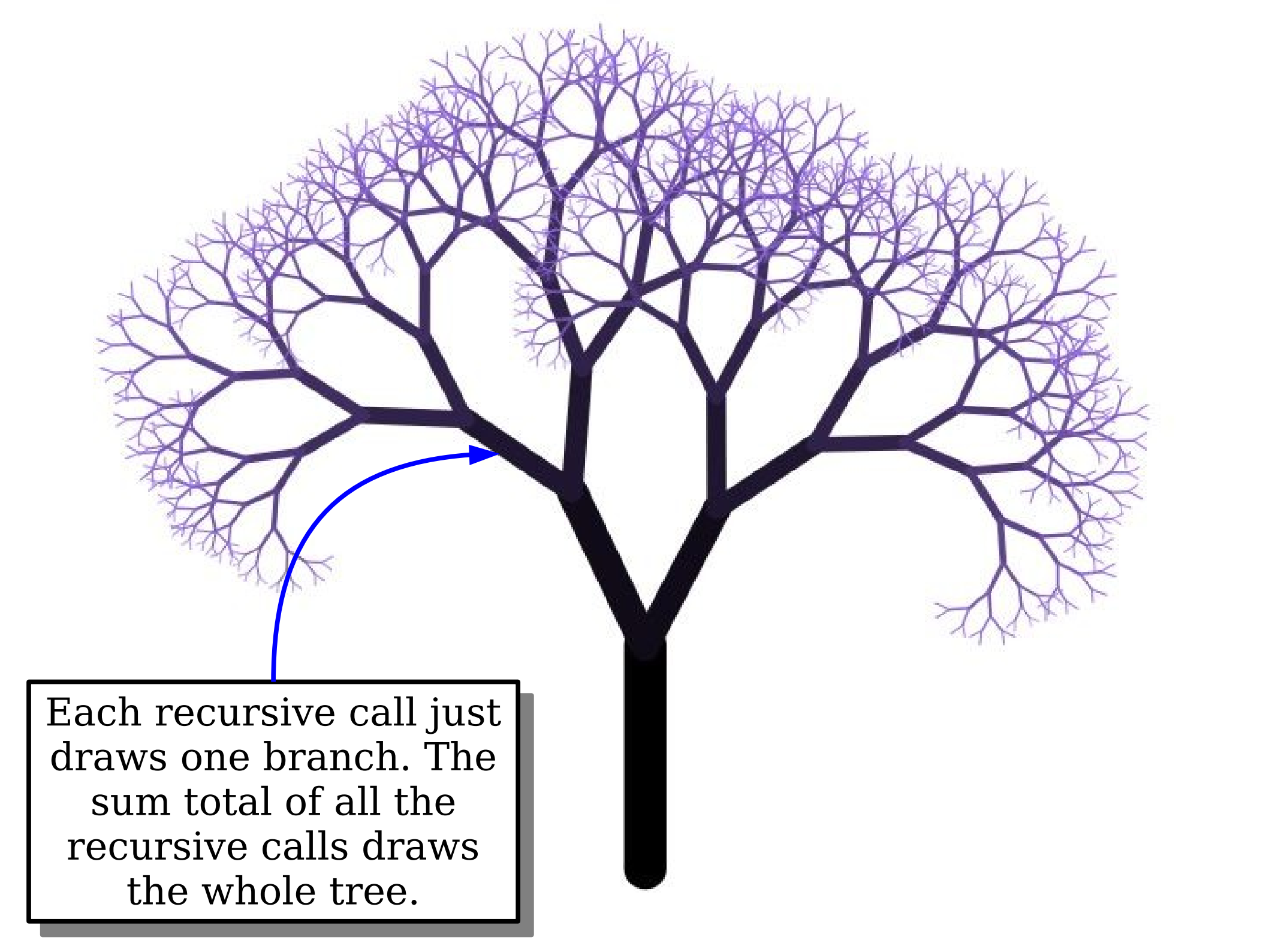
```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

drawTree(window,
         treeRootX, treeRootY,
         treeHeight);
```

# Wrapper Functions

- Some recursive functions need extra arguments as part of an implementation detail.
  - In our case, the order of the tree is not something we want to expose.
- A ***wrapper function*** is a function that does some initial prep work, then fires off a recursive call with the right arguments.

# 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

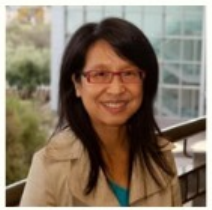
***http://recursivedrawing.com/***

Time-Out for Announcements!

STANFORD COMPUTER SCIENCE'S

# UNDERGRAD RESEARCH PANEL

How do you find research opportunities in the Computer Science department? What is CURIS? How can you do research for your senior project or thesis? How can you join a lab? RSVP to this panel to find out!


Professor Monica Lam


Professor Mary Wootters


Professor Michael Bernstein


Professor Anshul Kundaje

**JANUARY 31, 2019**
**5:30PM**
**GATES 415**

**There will be pizza!!!!!!**

RSVP using **this link**!

# Assignment 2

- Assignment 2 is due this upcoming Monday.
  - If you’re following our suggested timetable, you should be done with Crystals at this point and should be working on Evil Hangman.
- Have questions?
  - Stop by the LaIR!
  - Email your section leader!

# Submitting Your Work

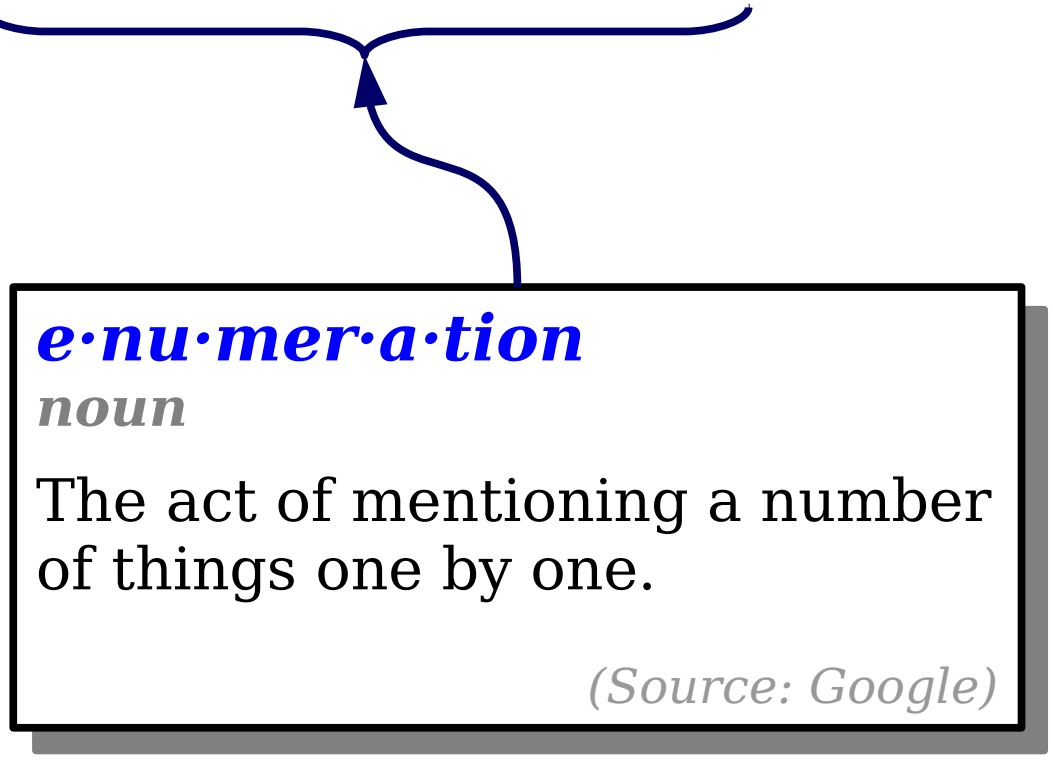
- Each assignment handout has a “Submission Instructions” section at the end with information about what files to submit.
- ***Please submit all the files listed there***. Otherwise, we can’t grade your work.
- Thanks!

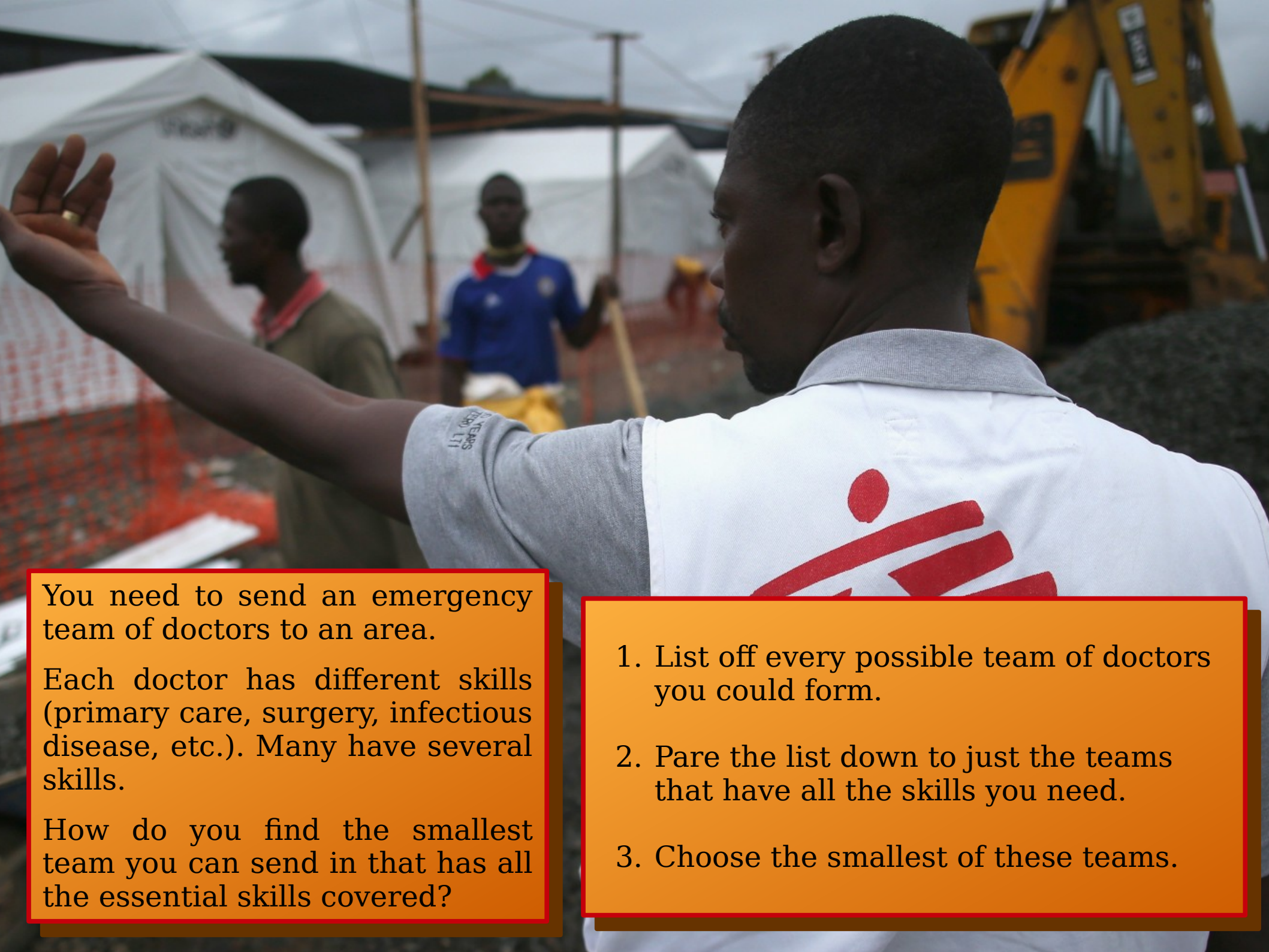
# Looking Ahead: The Midterm

- Our midterm exam is Tuesday, February 19th.
- If you have any exam conflicts or will need OAE accommodations, please contact Kate as soon as possible.
- This is a big class – we need some lead time to make everything work out!

Onward and Forward!

# Recursive Enumeration



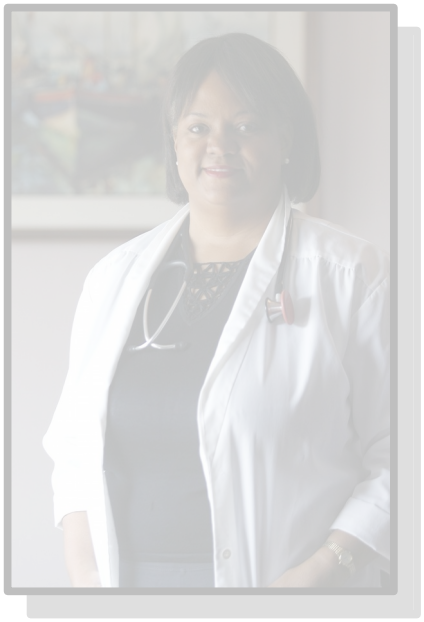


You need to send an emergency team of doctors to an area.

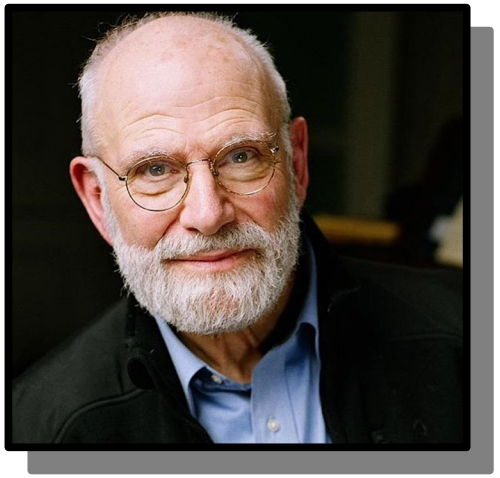
Each doctor has different skills (primary care, surgery, infectious disease, etc.). Many have several skills.

How do you find the smallest team you can send in that has all the essential skills covered?

1. List off every possible team of doctors you could form.
2. Pare the list down to just the teams that have all the skills you need.
3. Choose the smallest of these teams.



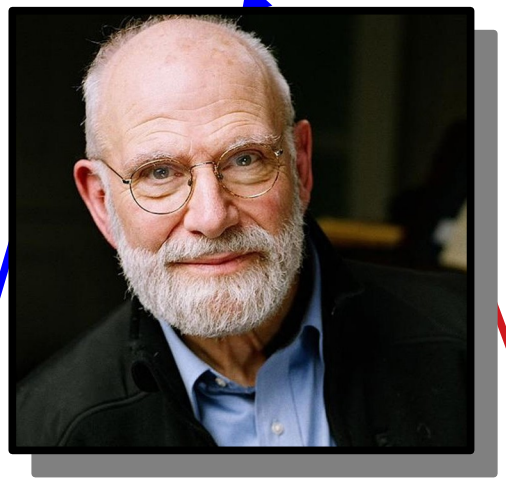

NYU
Medical
Center

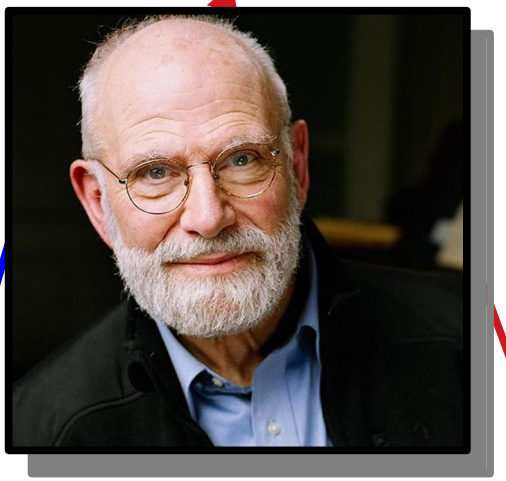



NYU
Medical
Center

✓

×





✓

×

✓

×

…

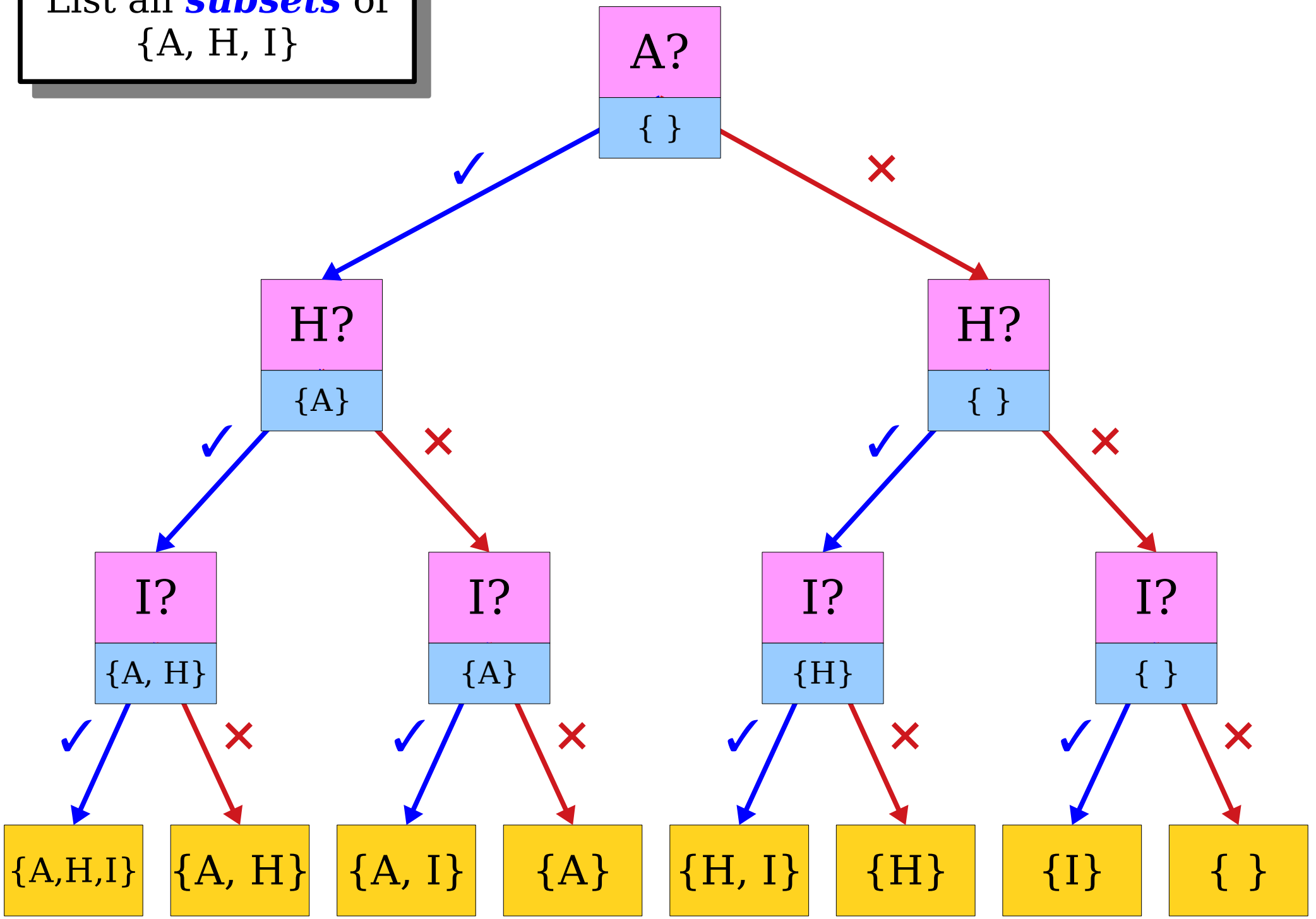
…

…

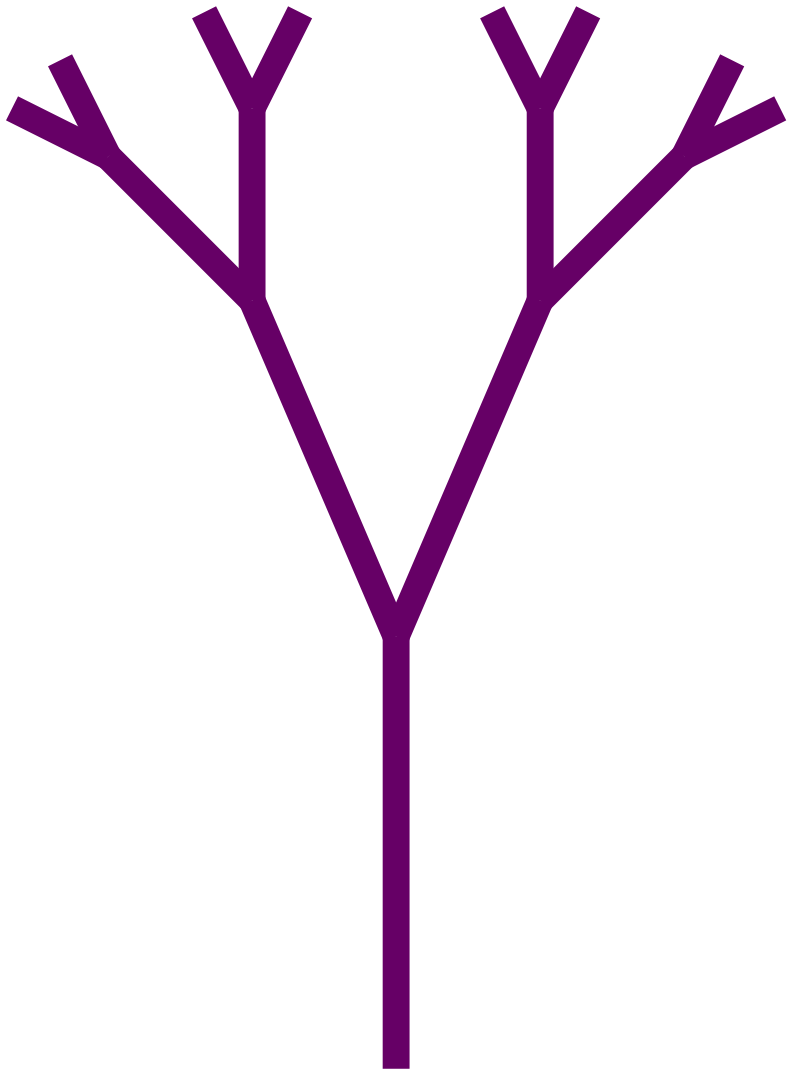
…

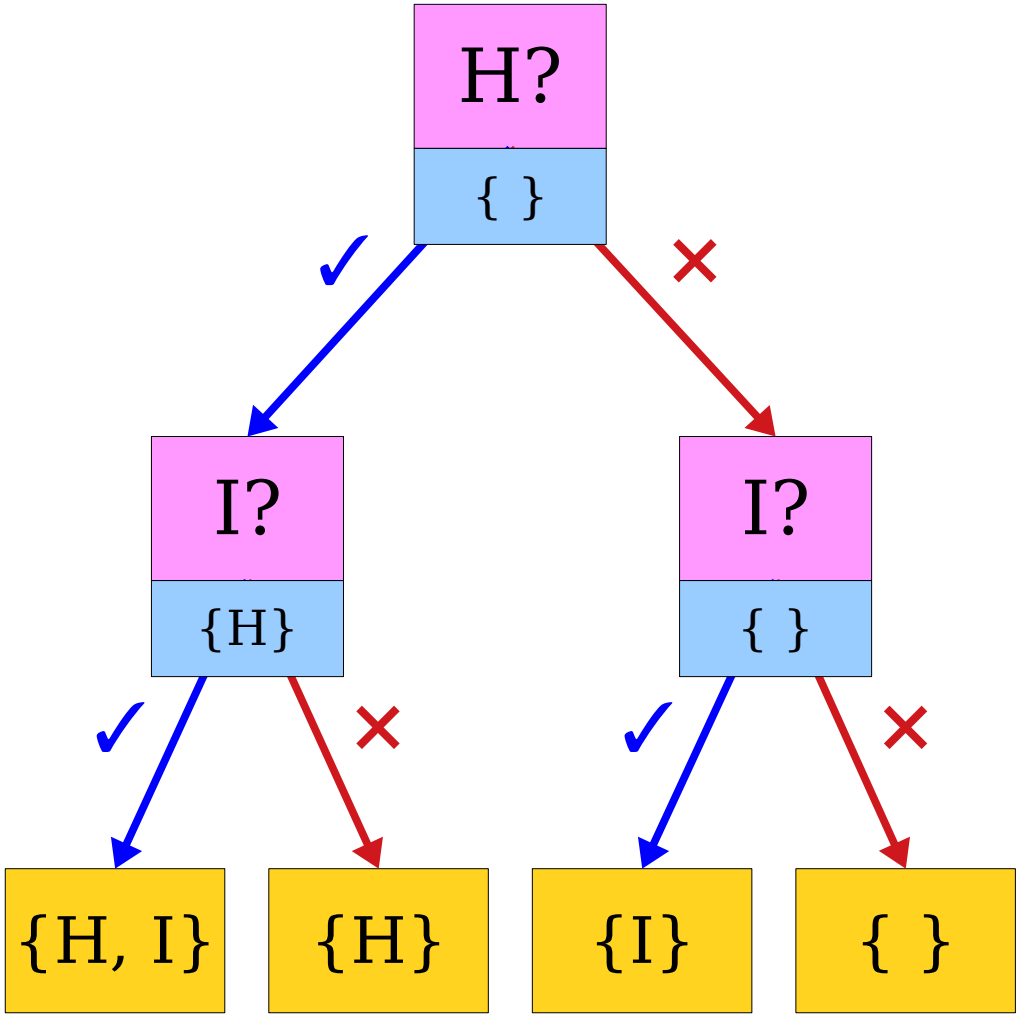
This structure is called a ***decision tree***.

List all ***subsets*** of {A, H, I}

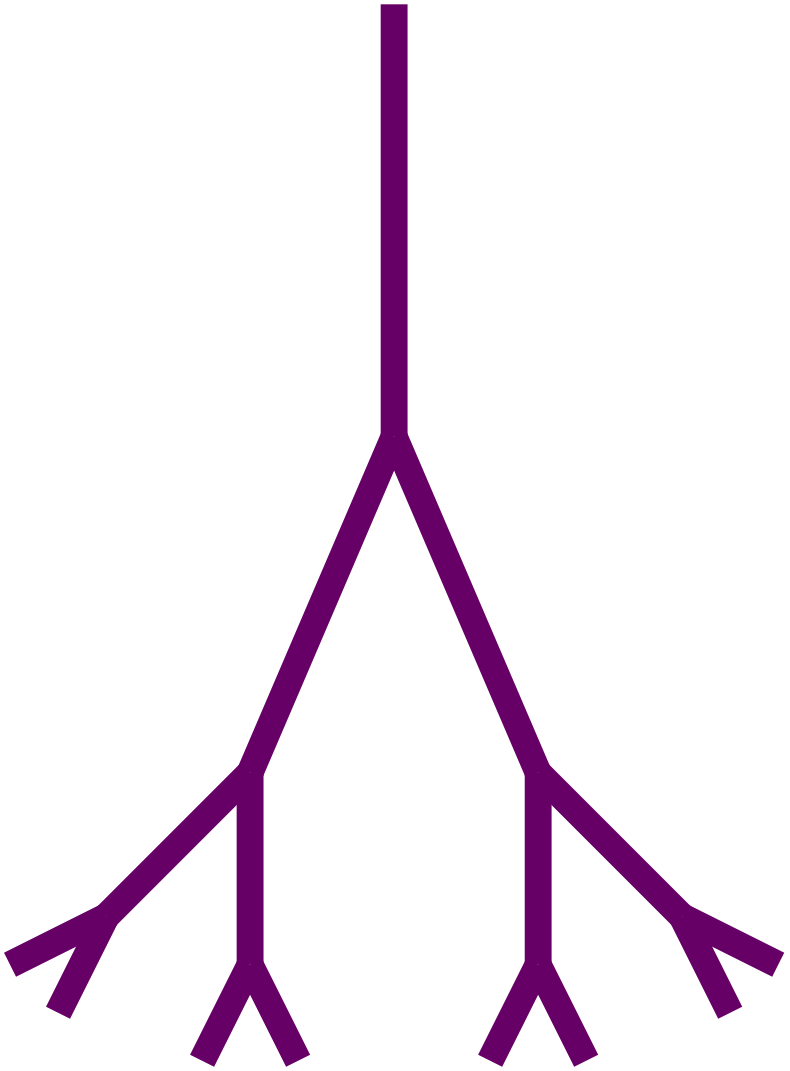


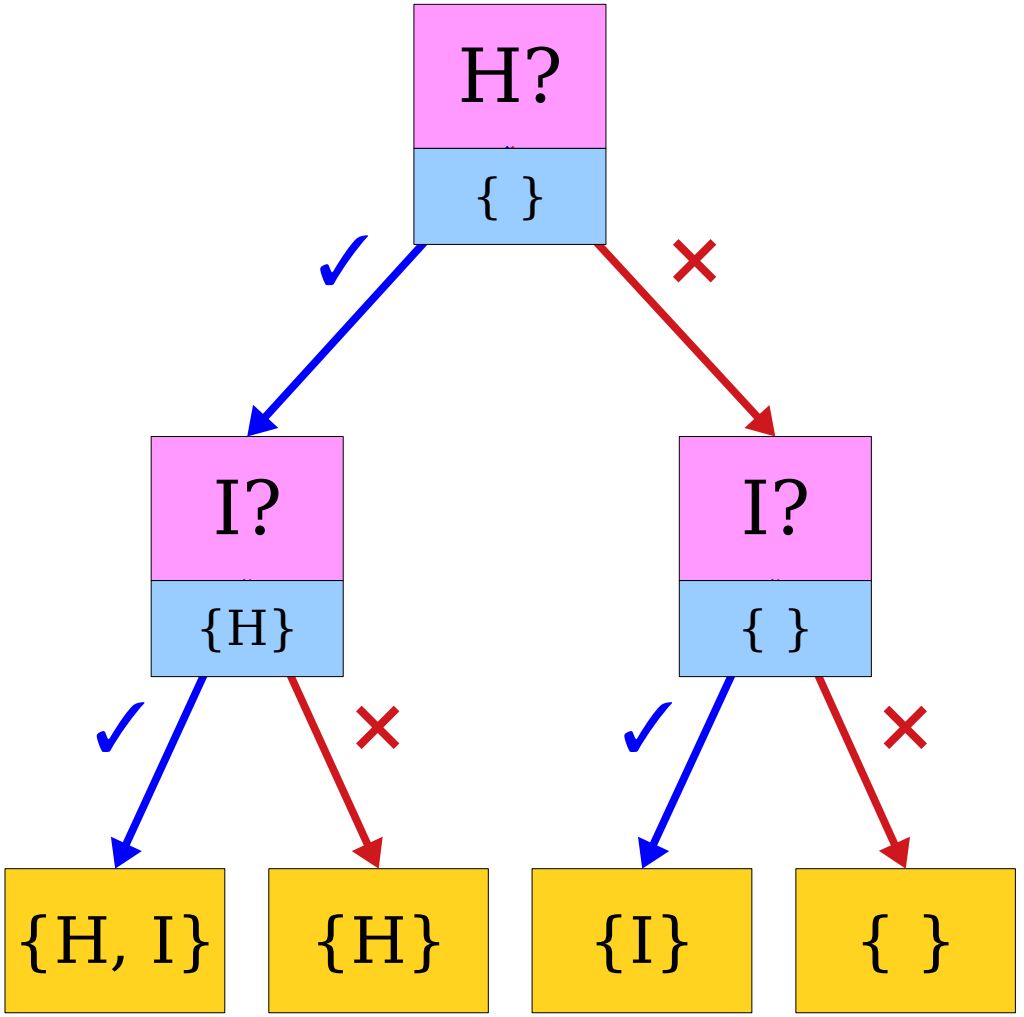
# Two Trees

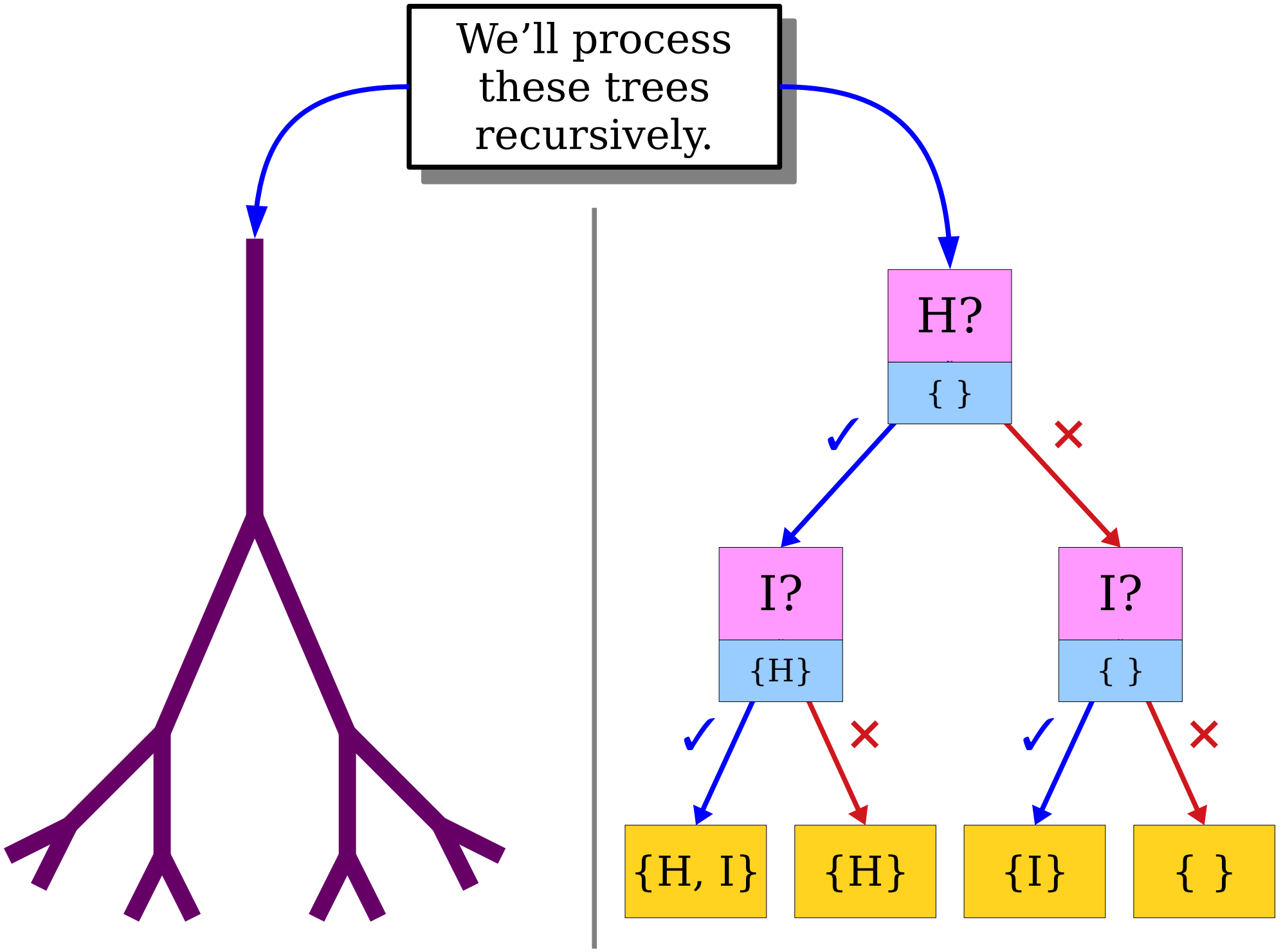


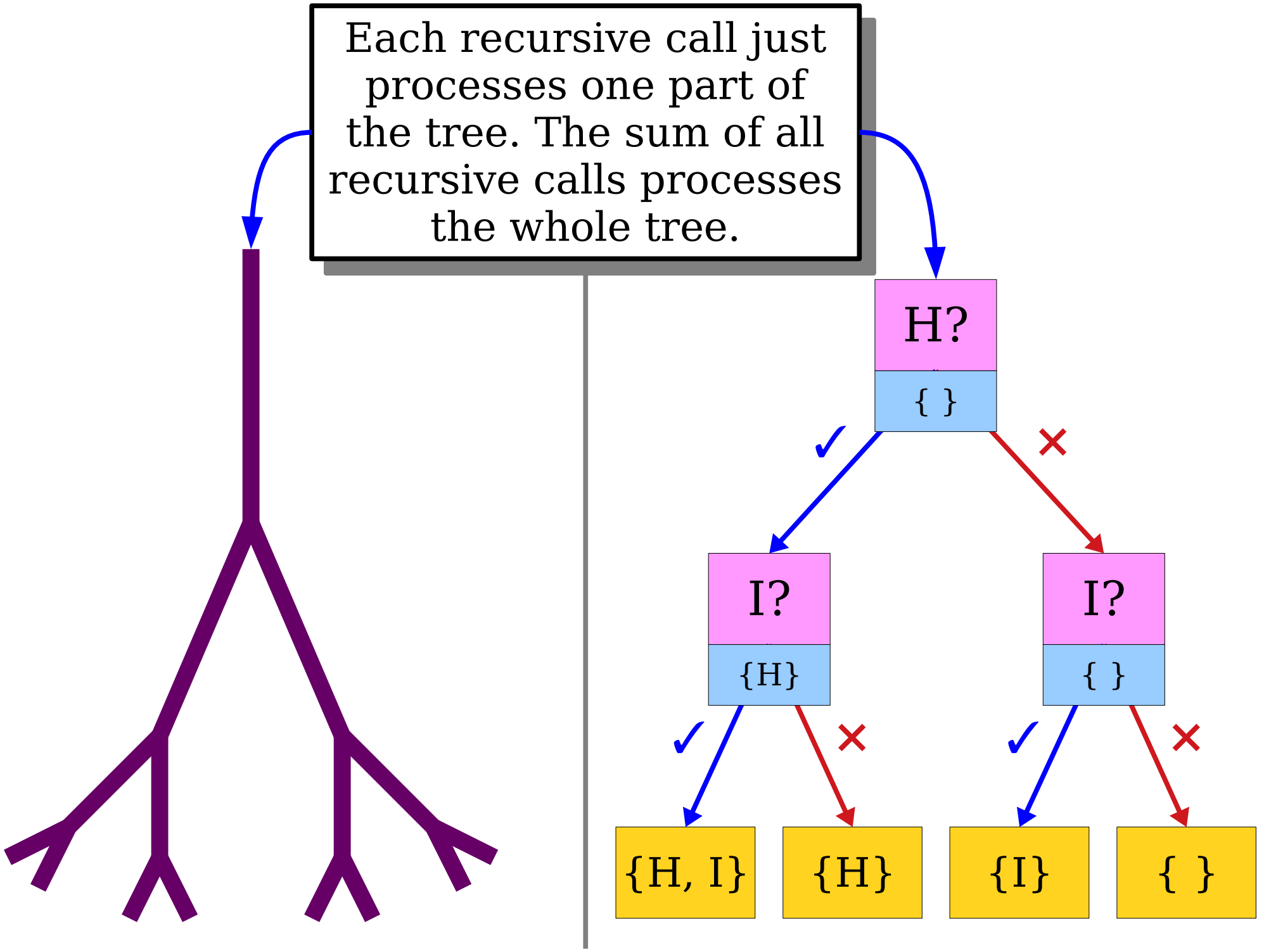


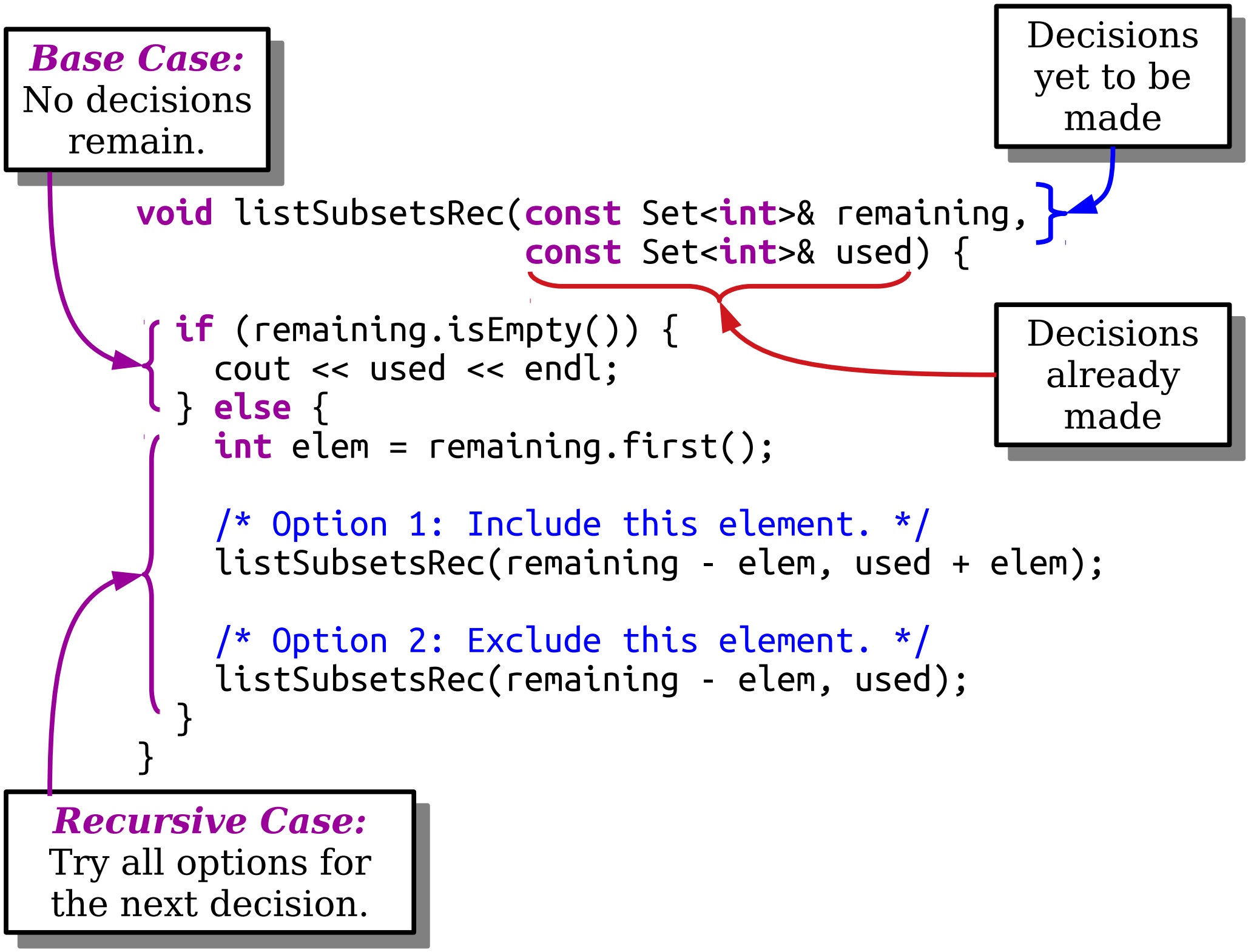
# Two Trees






We'll process these trees recursively.
H?
{ }
✓
✗
I?
{H}
I?
{ }
✓
✗
✓
✗
{H, I}
{H}
{I}
{ }


Each recursive call just processes one part of the tree. The sum of all recursive calls processes the whole tree.
H?
{ }
✓
✗
I?
{H}
I?
{ }
✓
✗
✓
✗
{H, I}
{H}
{I}
{ }


Base Case:
No decisions remain.
Decisions yet to be made
void listSubsetsRec(const Set<int>& remaining,
const Set<int>& used) {
if (remaining.isEmpty()) {
cout << used << endl;
} else {
int elem = remaining.first();
/* Option 1: Include this element. */
listSubsetsRec(remaining - elem, used + elem);
/* Option 2: Exclude this element. */
listSubsetsRec(remaining - elem, used);
}
}
Decisions already made
Recursive Case:
Try all options for the next decision.

# Your Action Items

- ***Work on Assignment 2***
  - It's due on Monday. We hope you've finished Crystals by this point and have started making progress on Evil Hangman.
  - Aim to get Evil Hangman mostly completed by Friday, leaving the weekend as buffer time.
- ***Read Chapter 8 of the Textbook***
  - There's a ton of goodies in there! It'll help you solidify your understanding.

# Next Time

- ***Enumerating Permutations***
  - Finding the best order in which to perform some tasks.
- ***Enumerating Combinations***
  - Finding the right team of the right size.