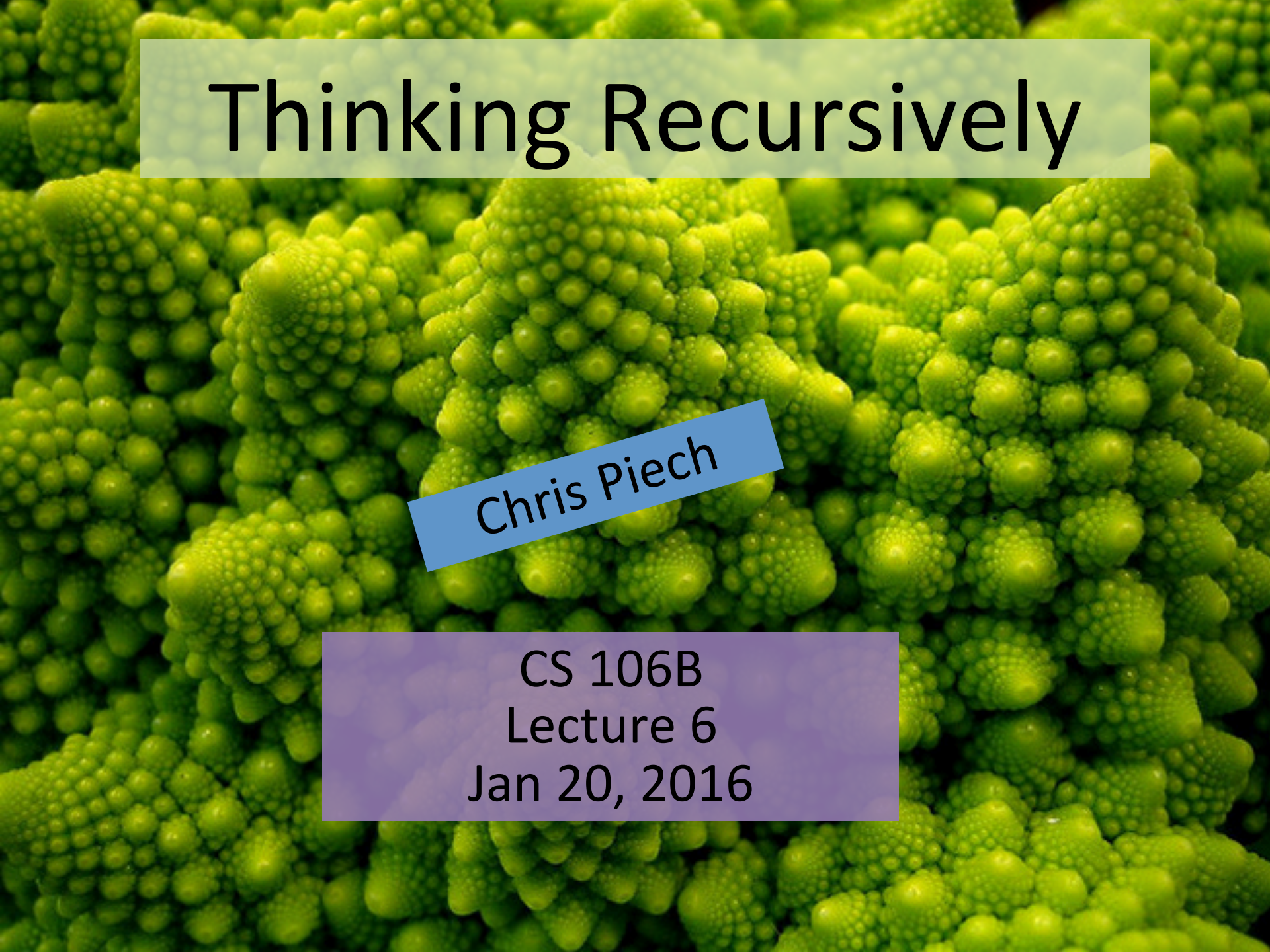

Thinking Recursively
Chris Piech
CS 106B
Lecture 6
Jan 20, 2016

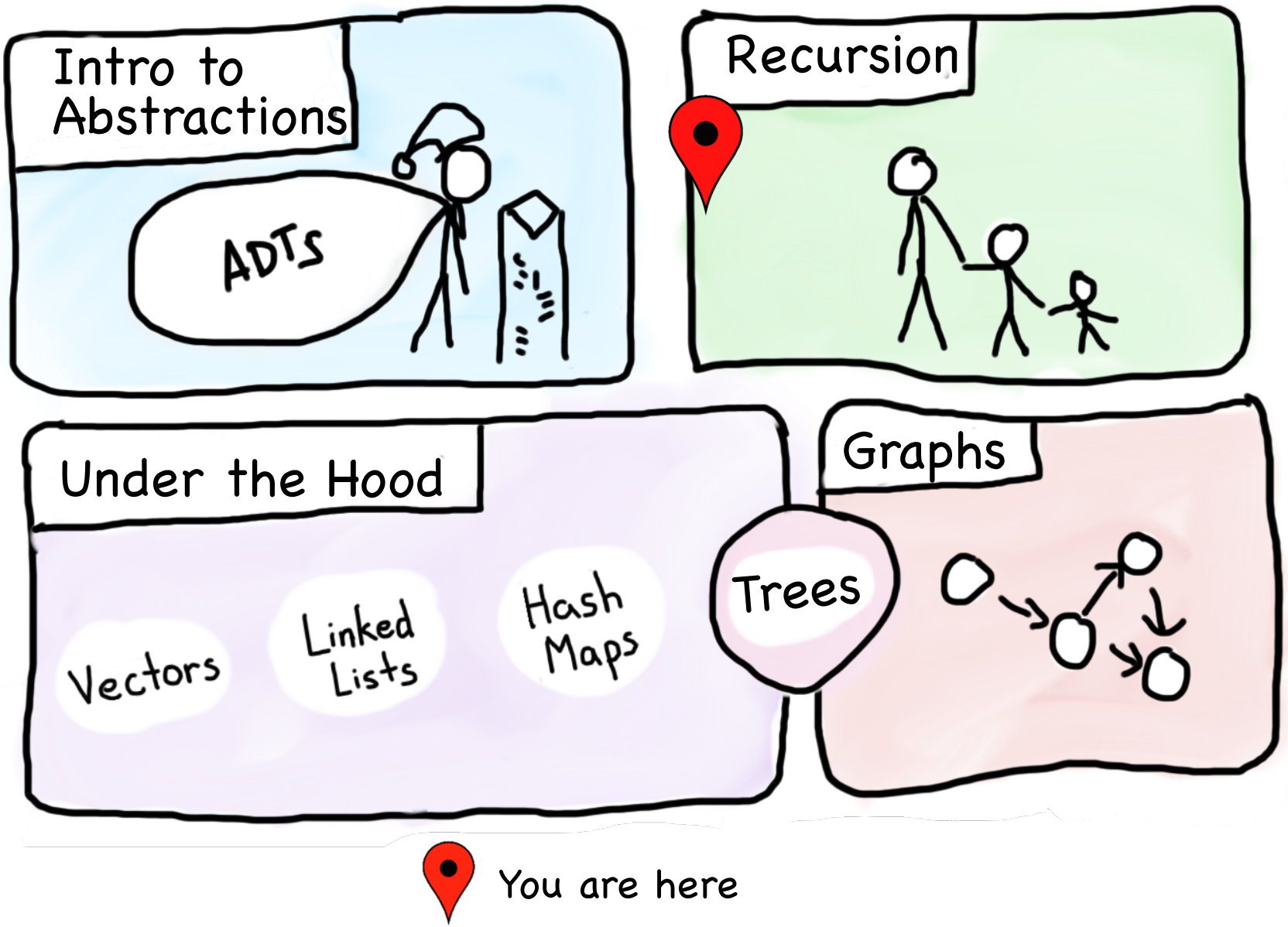
todo

- Post readings
- Post slides
- Rewatch Martys intro

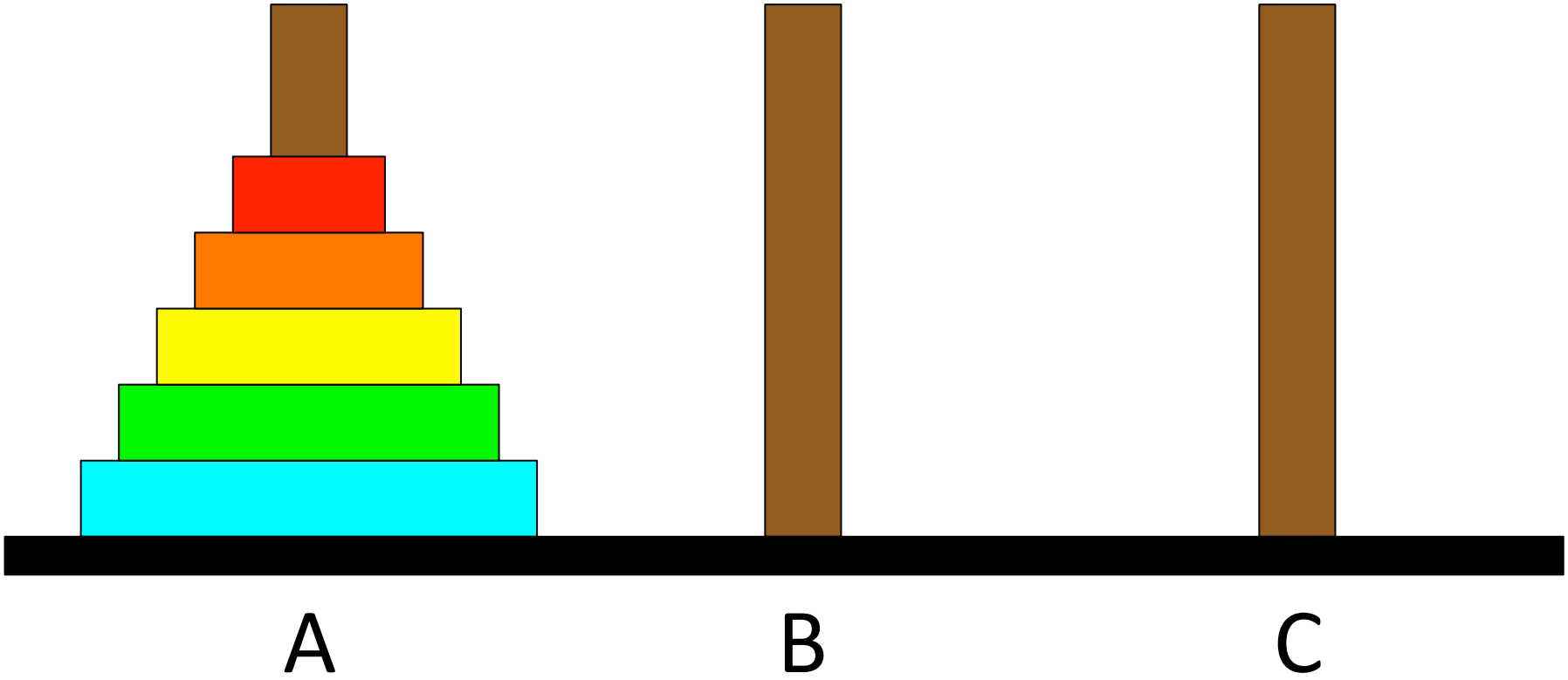
Socrative.com

Room: CS106BWIN16

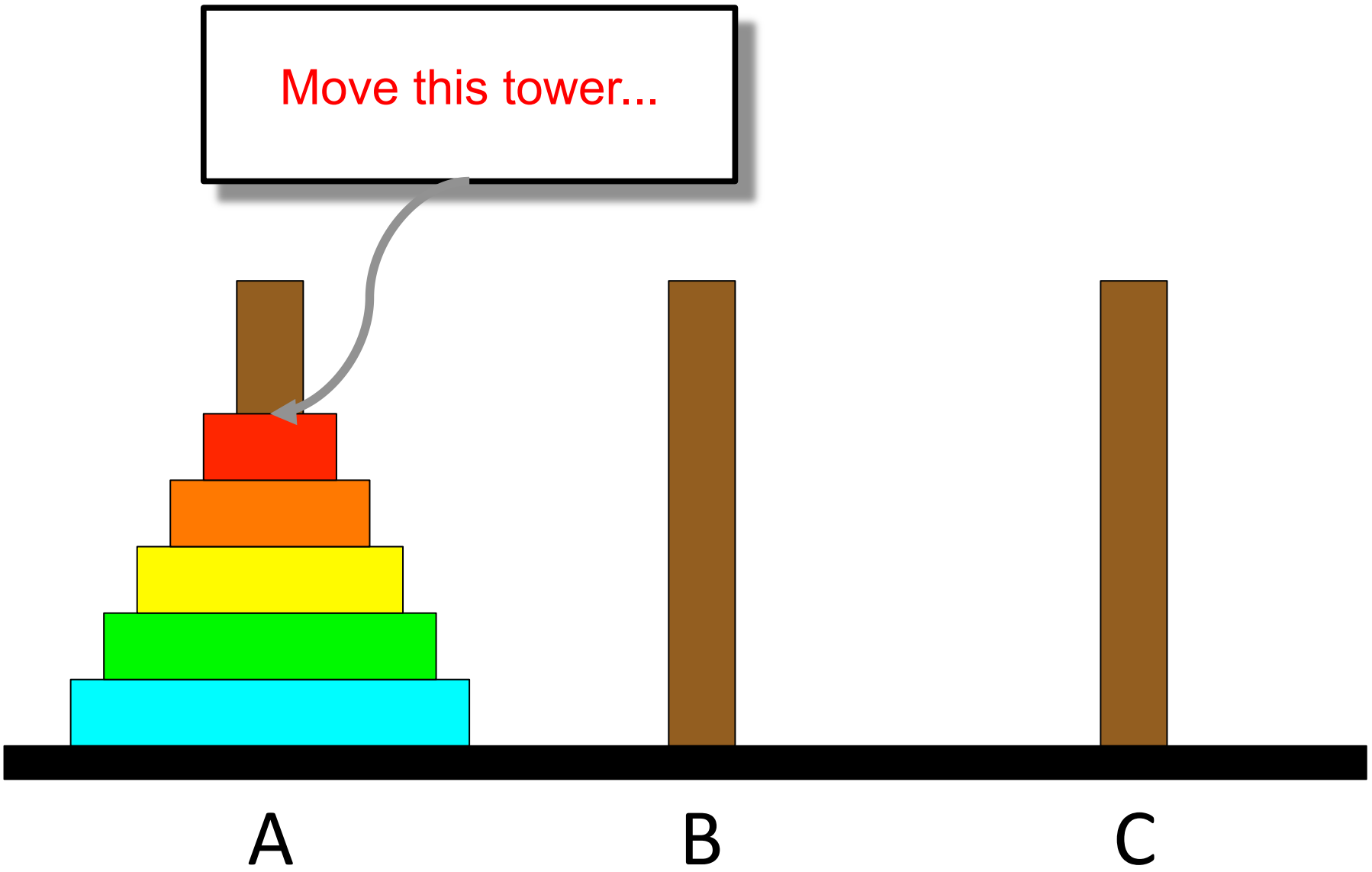
Course Syllabus



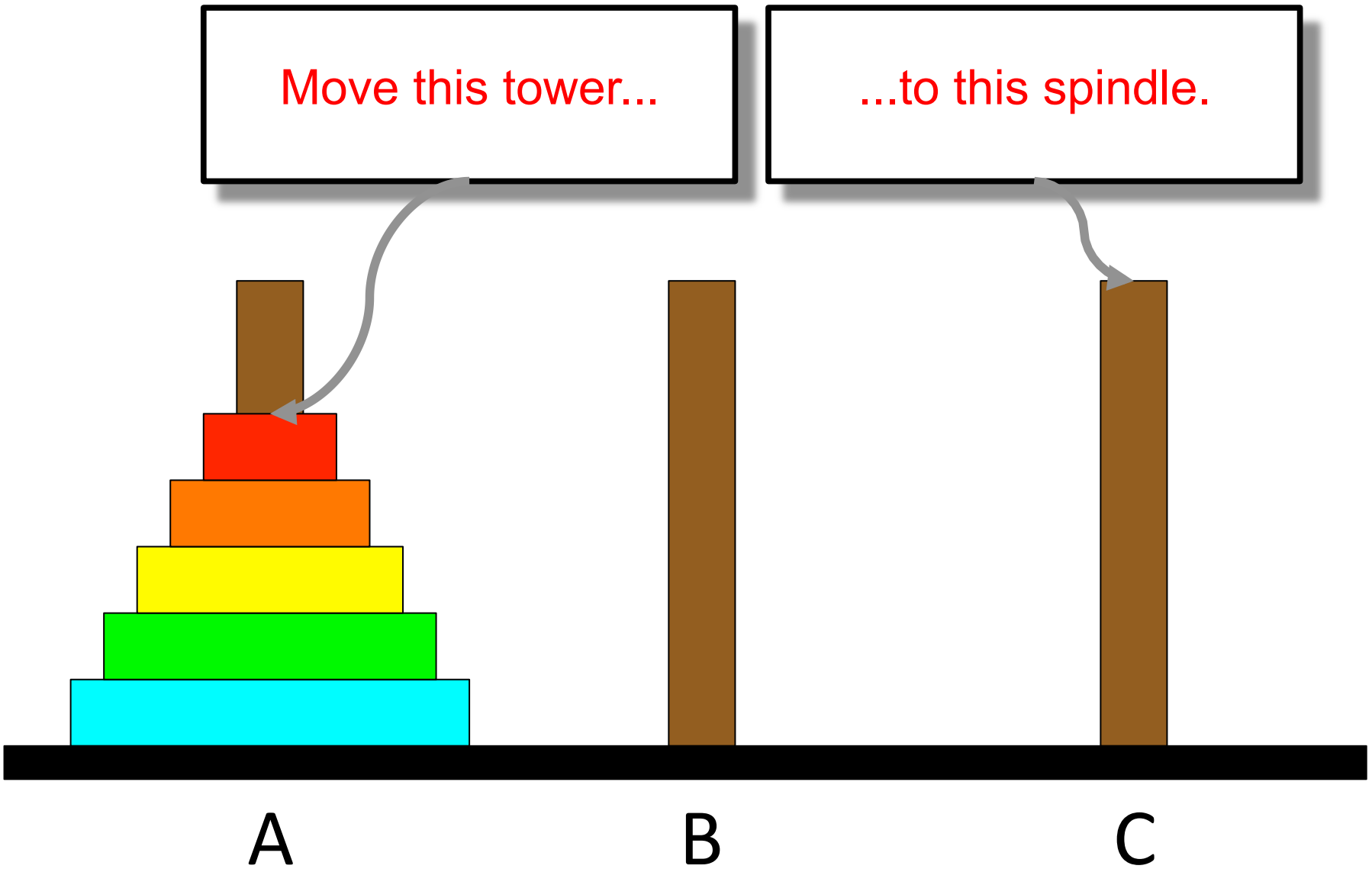
Towers of Hanoi



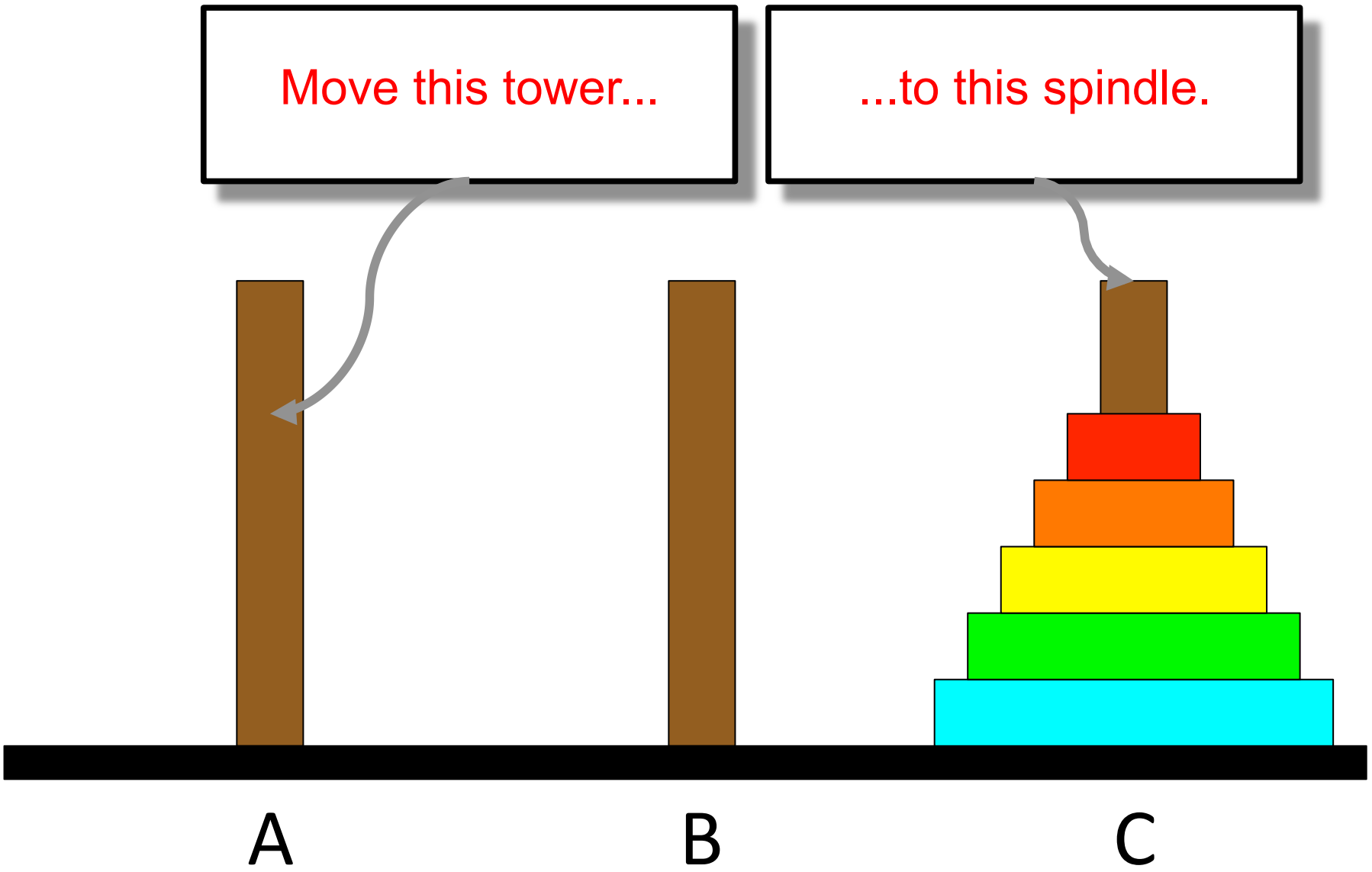
Towers of Hanoi



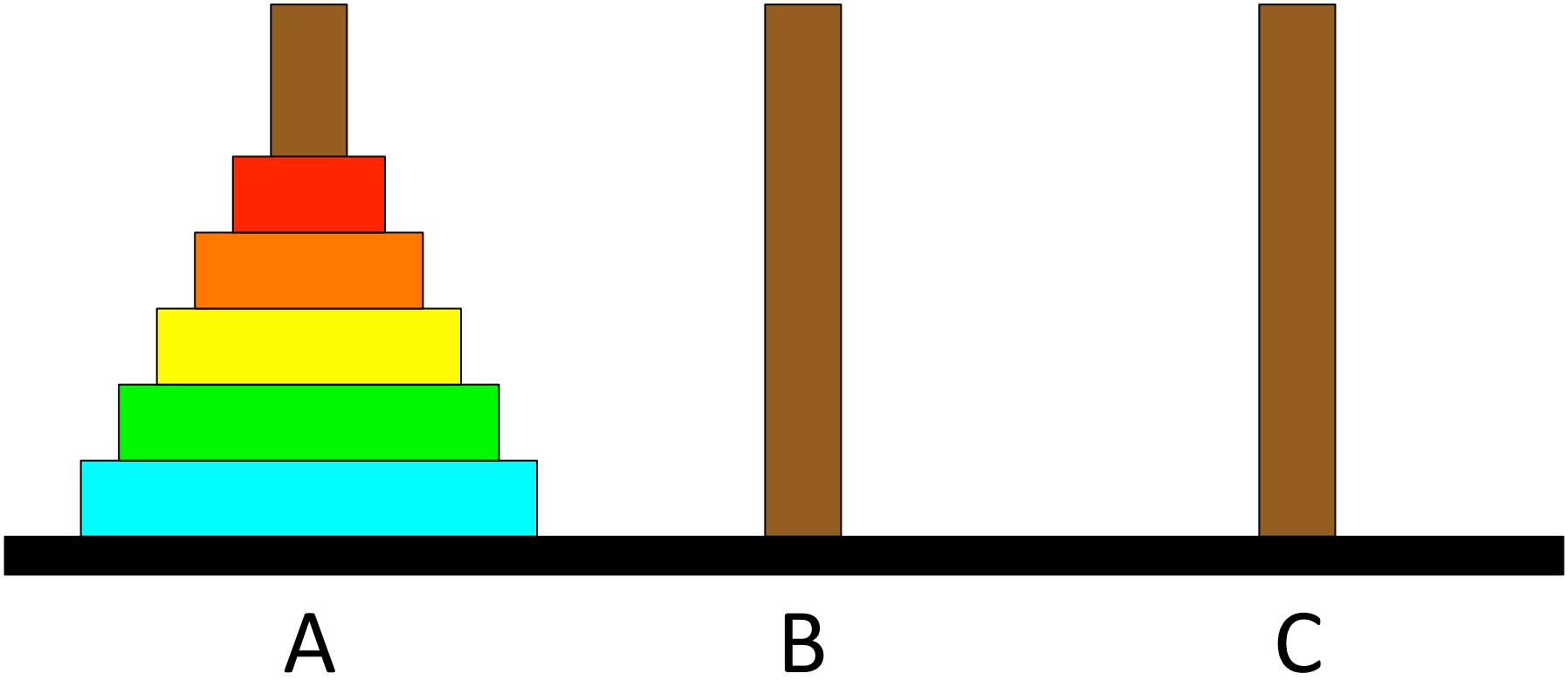
Towers of Hanoi



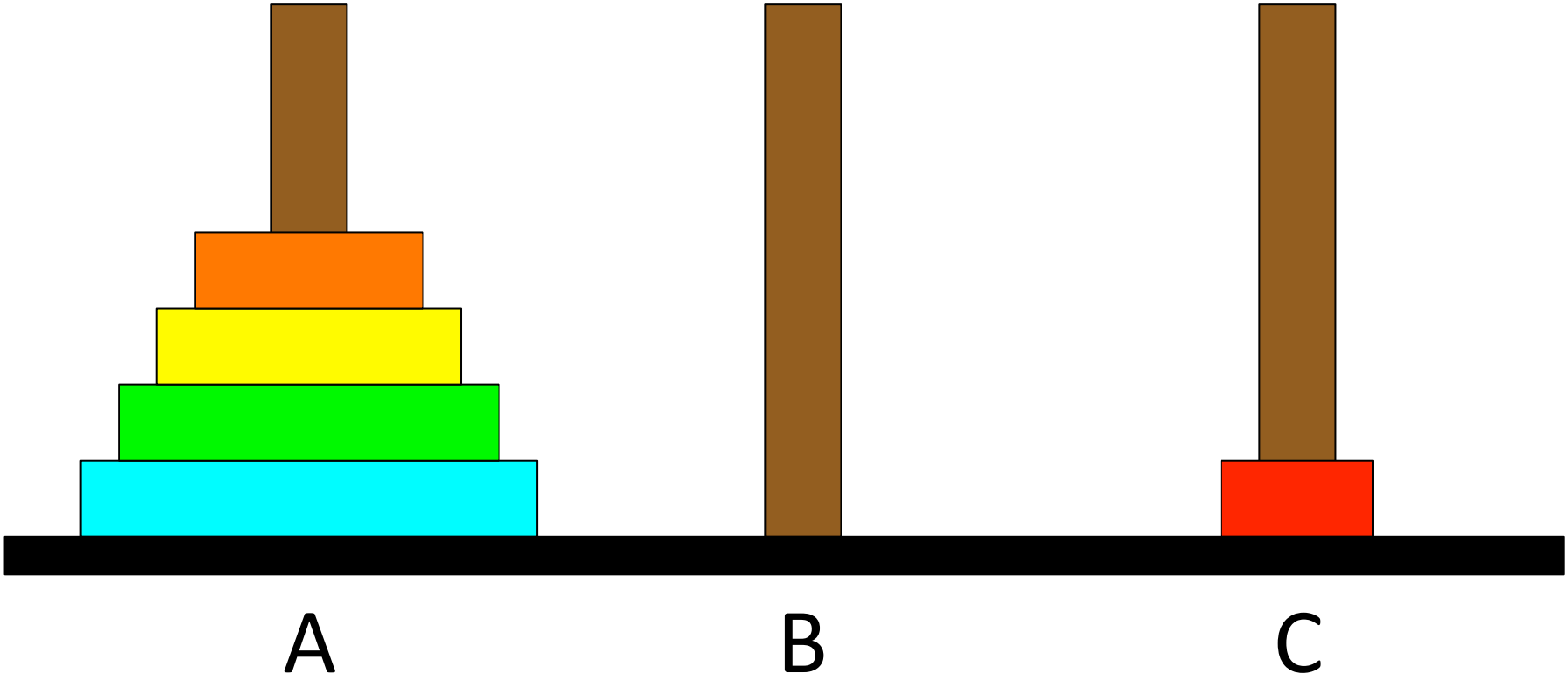
Towers of Hanoi



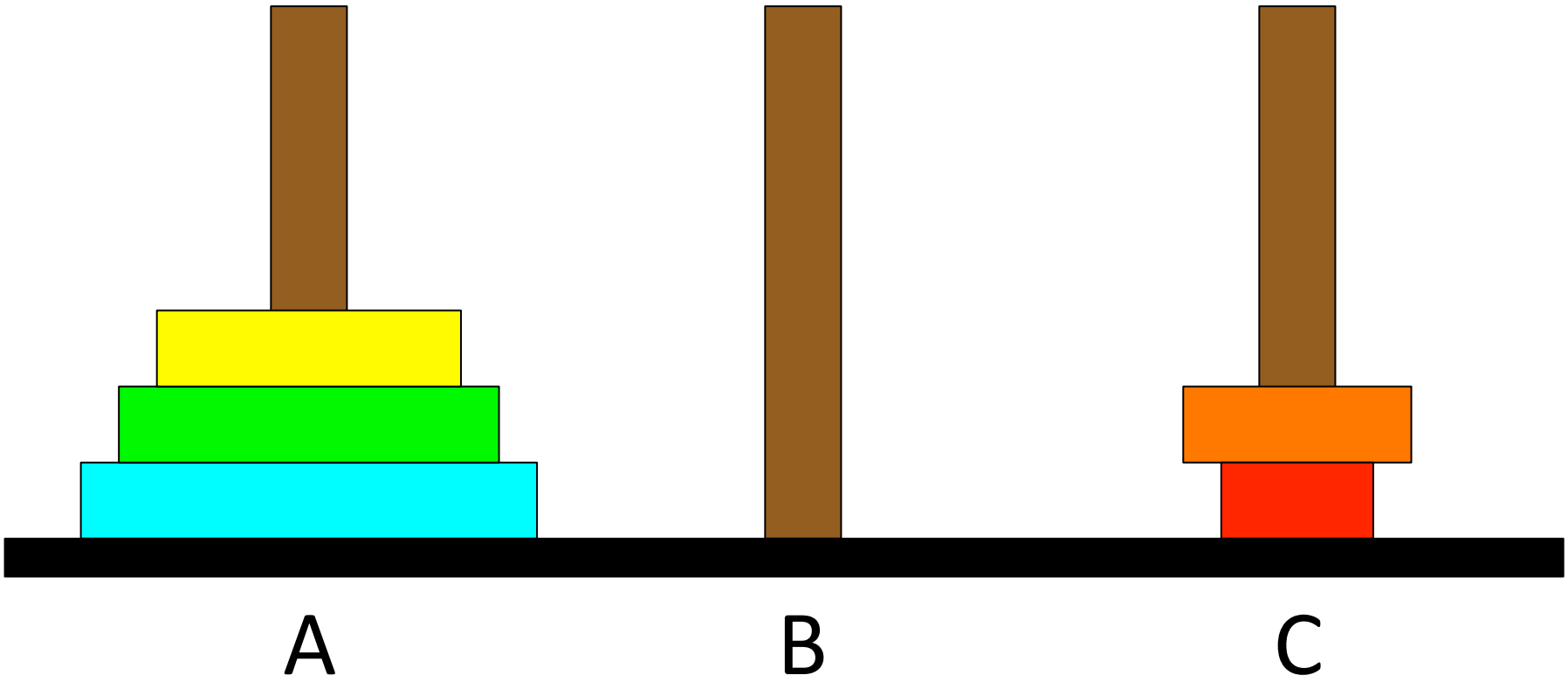
Towers of Hanoi



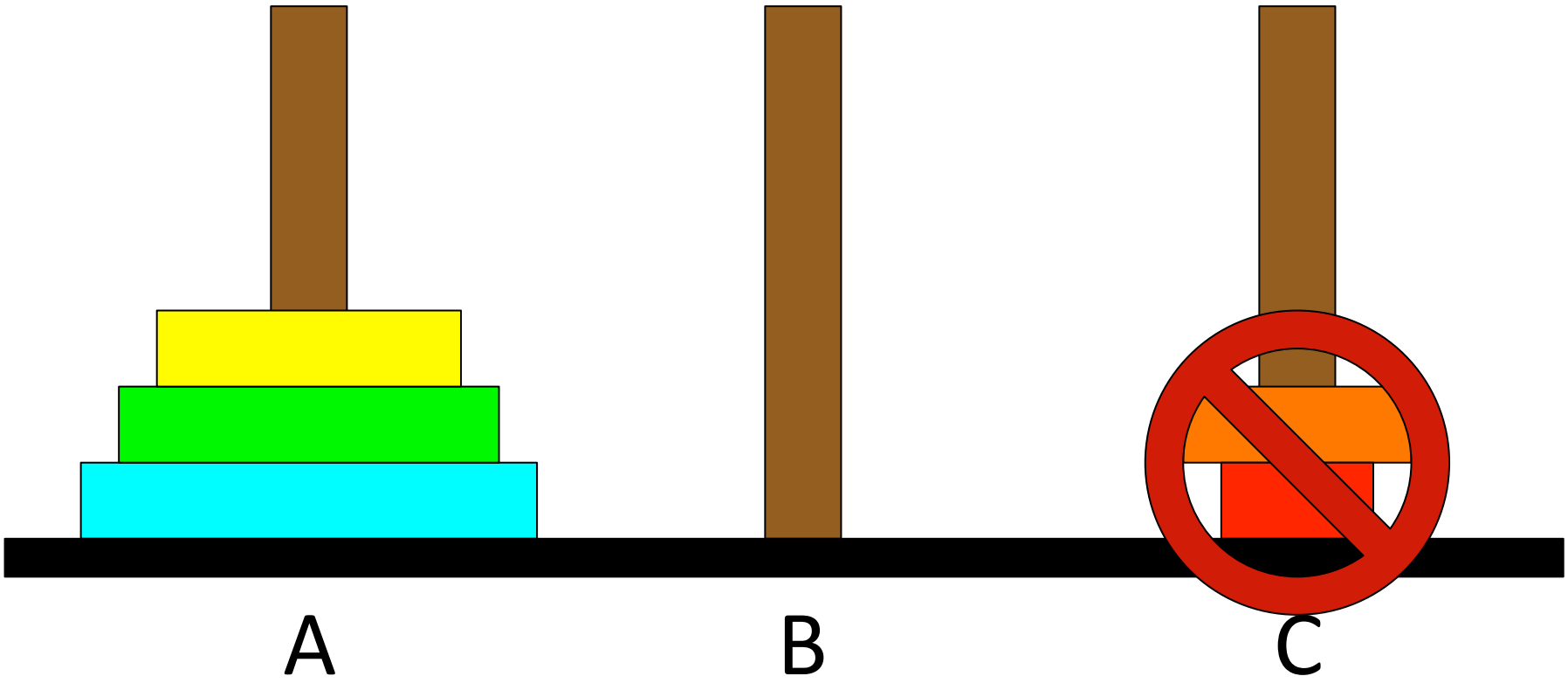
Towers of Hanoi



Towers of Hanoi



Towers of Hanoi




Qt

By the end of today we will write code that solves this problem

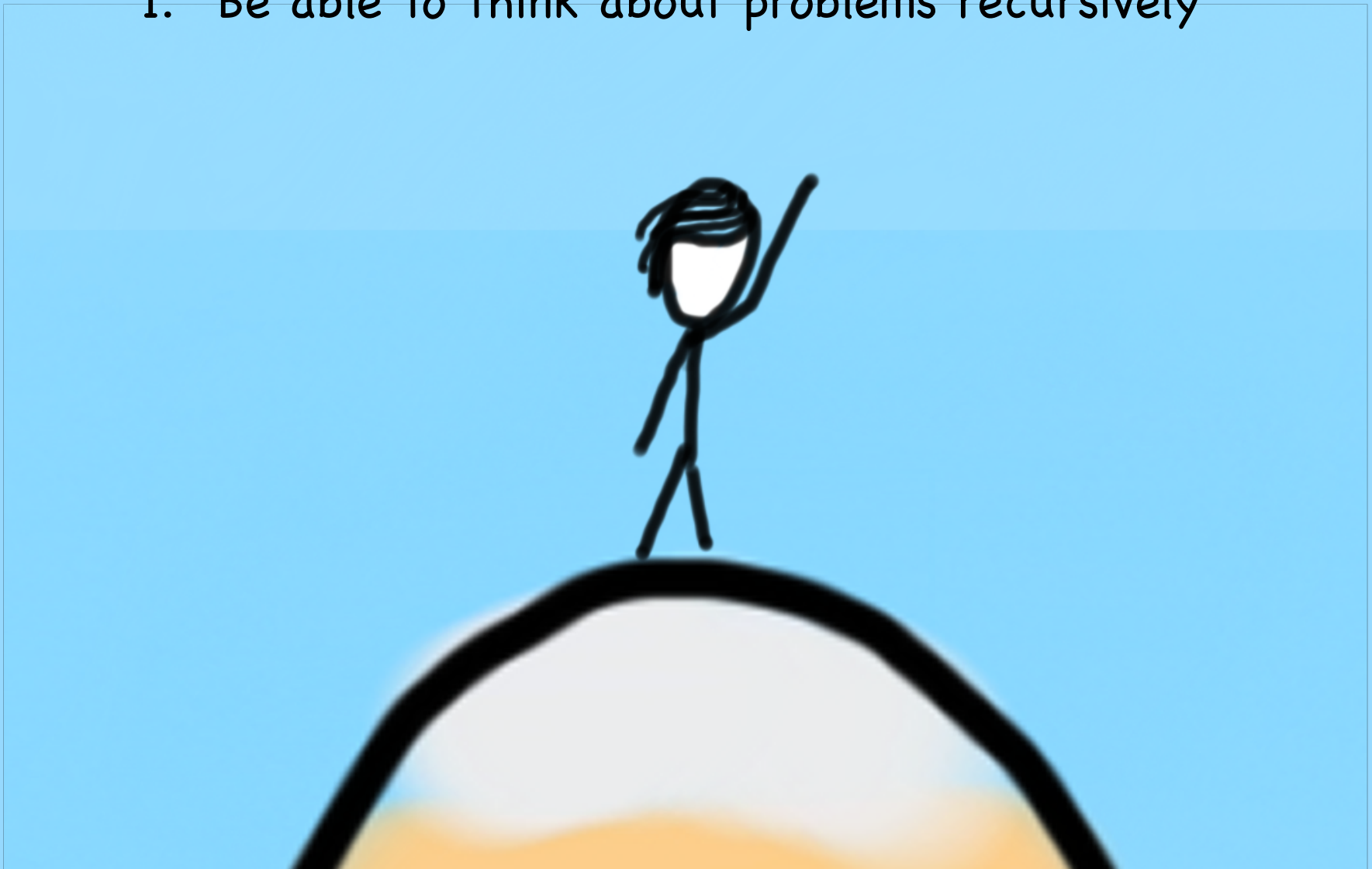
We've Gotten Ahead of Ourselves

Source: The Hobbit

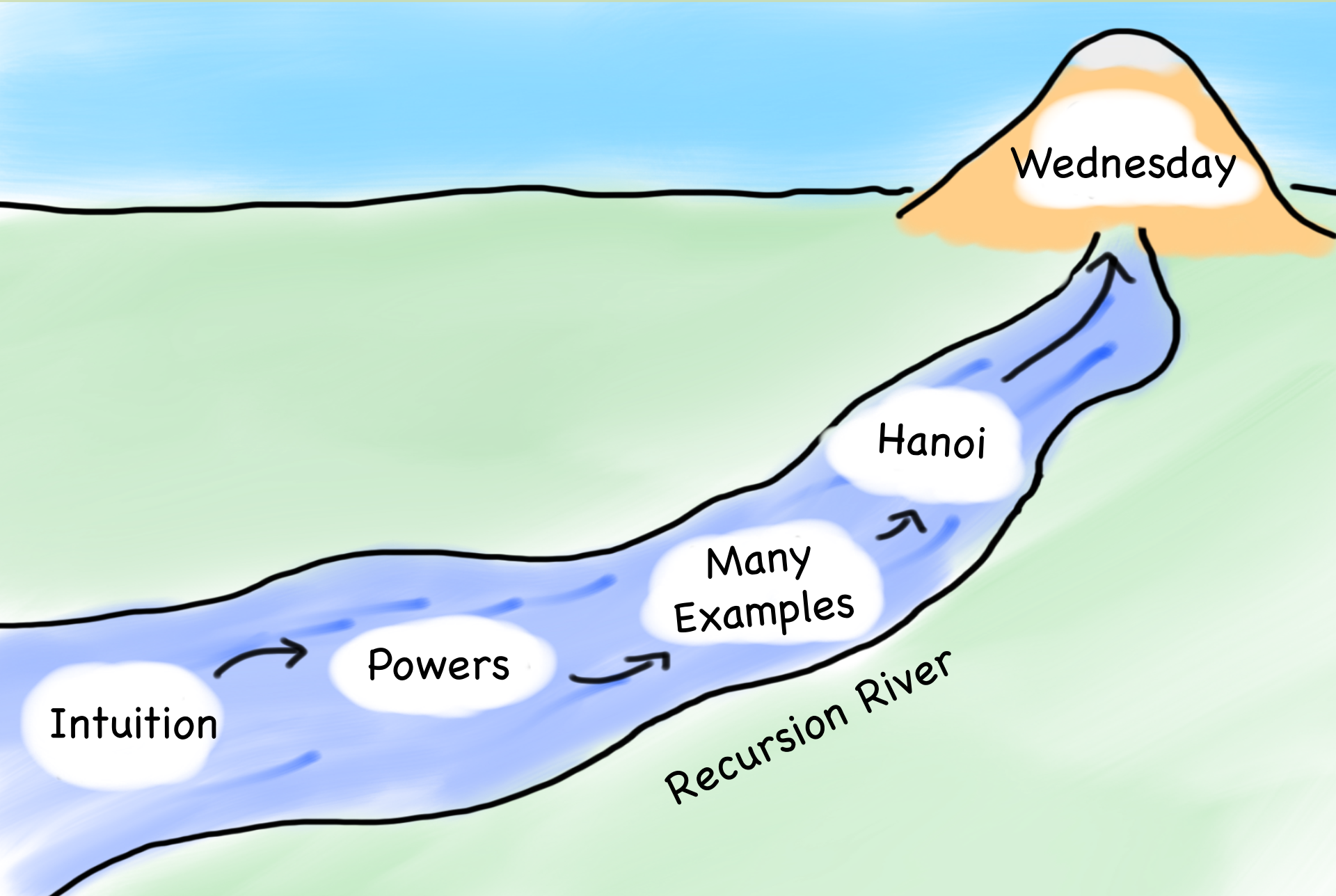
Start at the Beginning

Today's Goal

1. Be able to think about problems recursively



Today's Route



Pedagogy

Start Simple

Build Complexity

Recursion

Solving problems using functions that **call themselves**.

Understanding a Definition

mel·lif·lu·ous

/məˈlifloōəs/

adjective

(of a voice or words) sweet or musical; pleasant to hear.
"the voice was mellifluous and smooth"
synonyms: sweet-sounding, dulcet, honeyed, mellow, soft, liquid, silvery, soothing, rich, smooth, euphonious, harmonious, tuneful, musical
"mellifluous dinner music"

Translations, word origin, and more definitions

How Many Students in Col?

How many students total are directly behind you in your "column" of the classroom?

How Many Students in Col?

How many students total are directly behind you in your "column" of the classroom?

1. You can see only the people right next to you. So you can't just look back and count.

2. But you are allowed to ask questions of the person next to you.

3. How can we solve this problem (*recursively*)

How Many Students in Col?

```
int numStudentsBehind(Student curr) {
    if(lastInRow(curr)) {
        return 0;
    } else {
        Student next = curr.getBehind();
        return numStudentsBehind(next) + 1;
    }
}
```

Anatomy of Recursion

```
int numStudentsBehind(Student curr) {
    if(lastInRow(curr)) {
        return 0;
    } else {
        Student next = curr.getBehind();
        return numStudentsBehind(next) + 1;
    }
}
```

Base Case

```
int numStudentsBehind(Student curr) {
    if(lastInRow(curr)) {
        return 0;
    } else {
        Student next = curr.getBehind();
        return numStudentsBehind(next) + 1;
    }
}
```

Recursive Case

```
int numStudentsBehind(Student curr) {
    if(lastInRow(curr)) {
        return 0;
    } else {
        Student next = curr.getBehind();
        return numStudentsBehind(next) + 1;
    }
}
```

Recursive Call

```
int numStudentsBehind(Student curr) {
    if(lastInRow(curr)) {
        return 0;
    } else {
        Student next = curr.getBehind();
        return numStudentsBehind(next) + 1;
    }
}
```

Recursive Call

```
int numStudentsBehind(Student curr) {
    if(lastInRow(curr)) {
        return 0;
    } else {
        Student next = curr.getBehind();
        return numStudentsBehind(next) + 1;
    }
}
```

Progress towards the base case

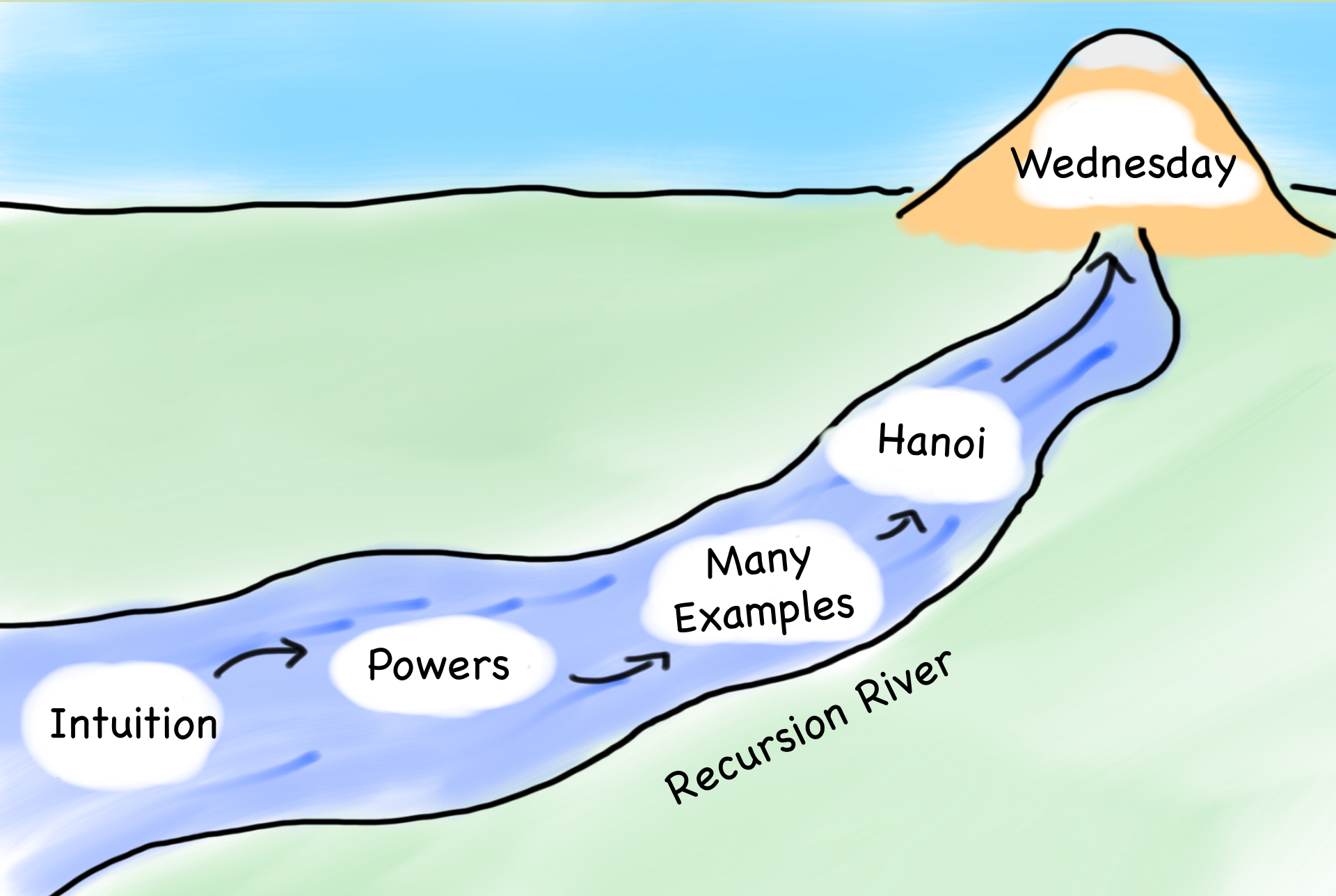
Three Musts of Recursion

1. Your code must have a case for all valid inputs.

2. You must have a base case (makes no recursive calls).

3. When you make a recursive call it should be to a simpler instance (forward progress towards base case)

Today's Route



Recursive Insight

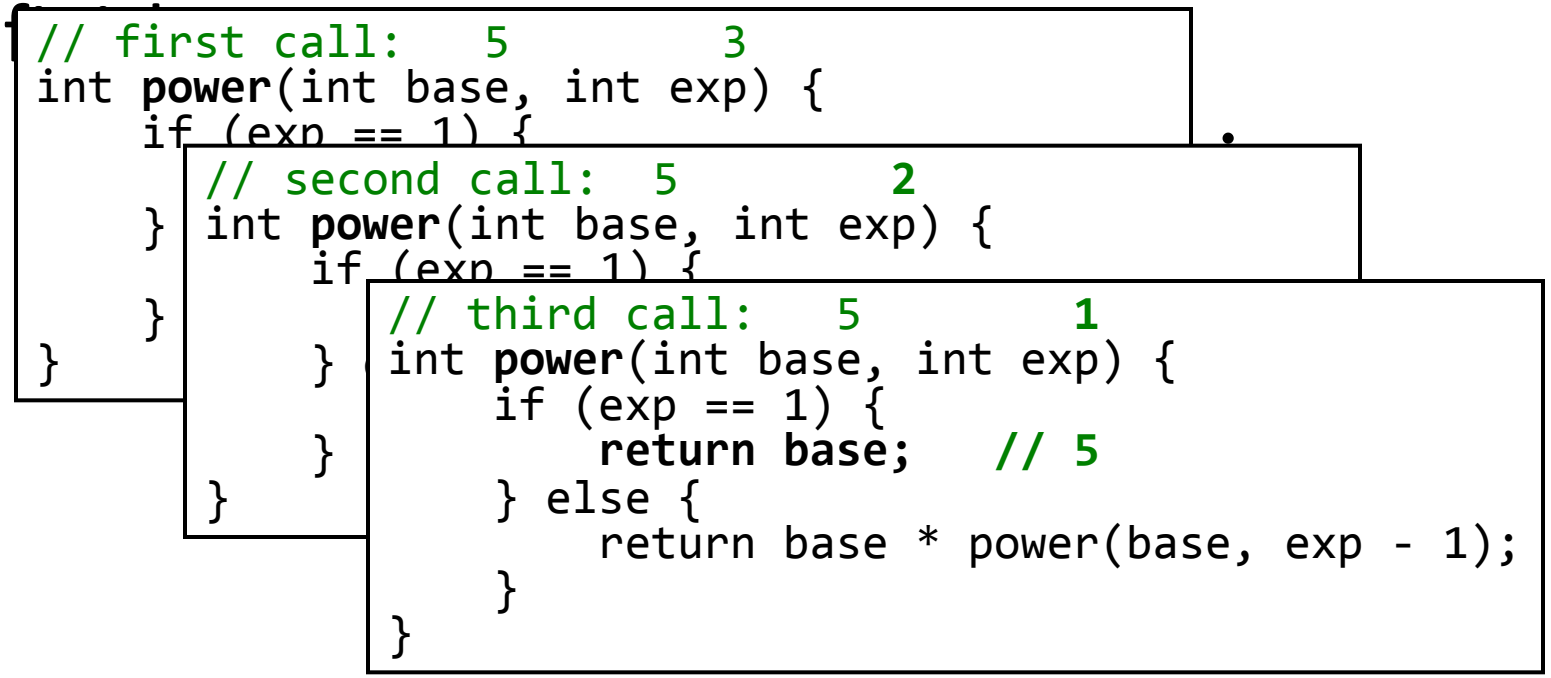
$$x^0 = 1$$

$$x^y = x \times x^{(y-1)}$$


Qt

The Call Stack

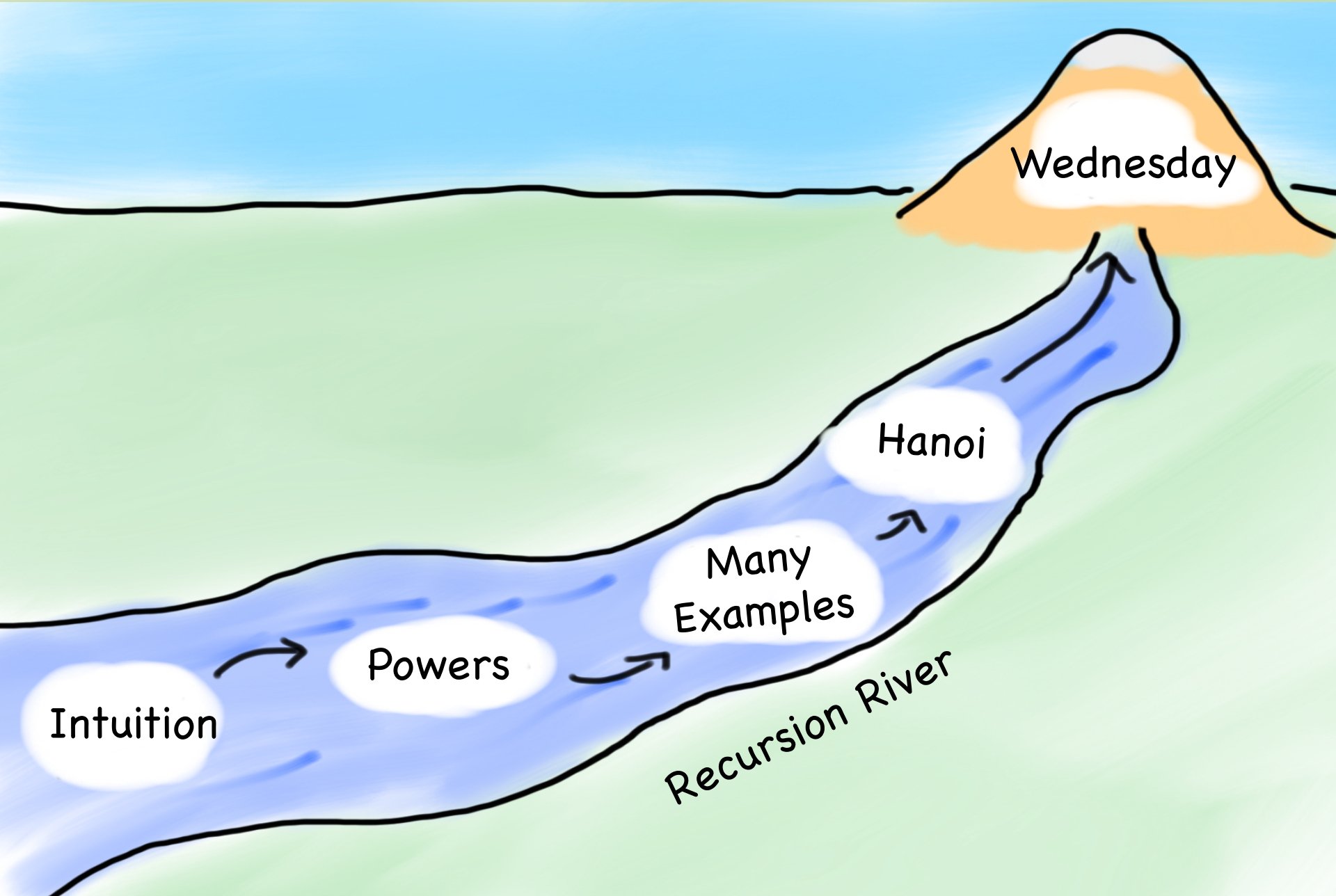
Each previous call waits for the next call to



Even Cooler Recursive Insight

$$x^n = \begin{cases} 1, & \text{if } n = 0 \\ \frac{1}{x}^{-n}, & \text{if } n < 0 \\ x \cdot \left(x^{\frac{n-1}{2}}\right)^2, & \text{if } n \text{ is odd} \\ \left(x^{\frac{n}{2}}\right)^2, & \text{if } n \text{ is even} \end{cases}$$

Today's Route



What Does This Code Do?

```
int mystery(int n) {
    if(n < 10) {
        return n;
    } else {
        int toAdd = n % 10;
        int theRest = mystery(n / 10);
        return theRest + toAdd;
    }
}
```

A: 5

B: 6

C: 7

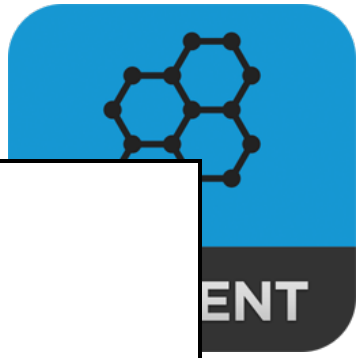
D: 8

What is the result of `mystery(123)`?

_____

What Does This Code Do?

```
int mystery(int n) {
    // first call: 123
    int mystery(int n) {
        if (n < 10) {
            return n;
        } else {
            return (n % 10) + mystery(n / 10);
        }
    }
    ...
}
```



A: 5

B: 6

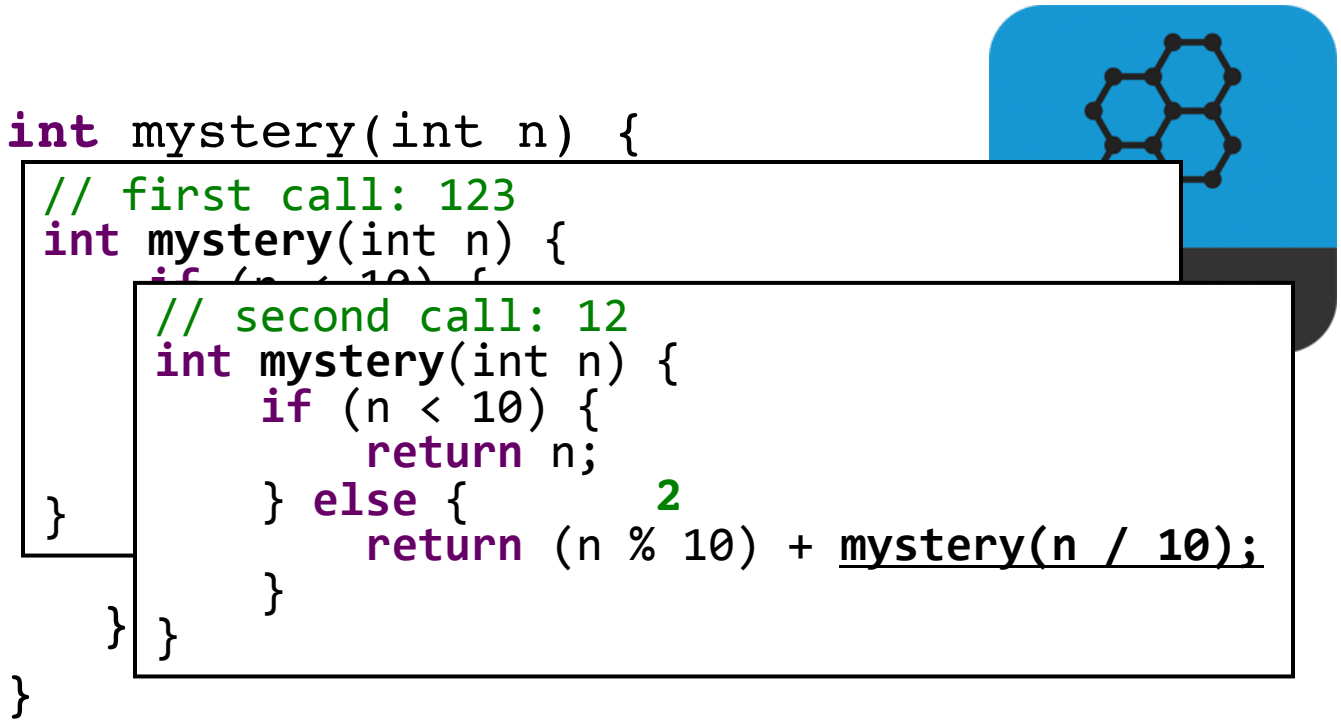
C: 7

D: 8

What is the result of `mystery(123)`?

_____

What Does This Code Do?



A: 5

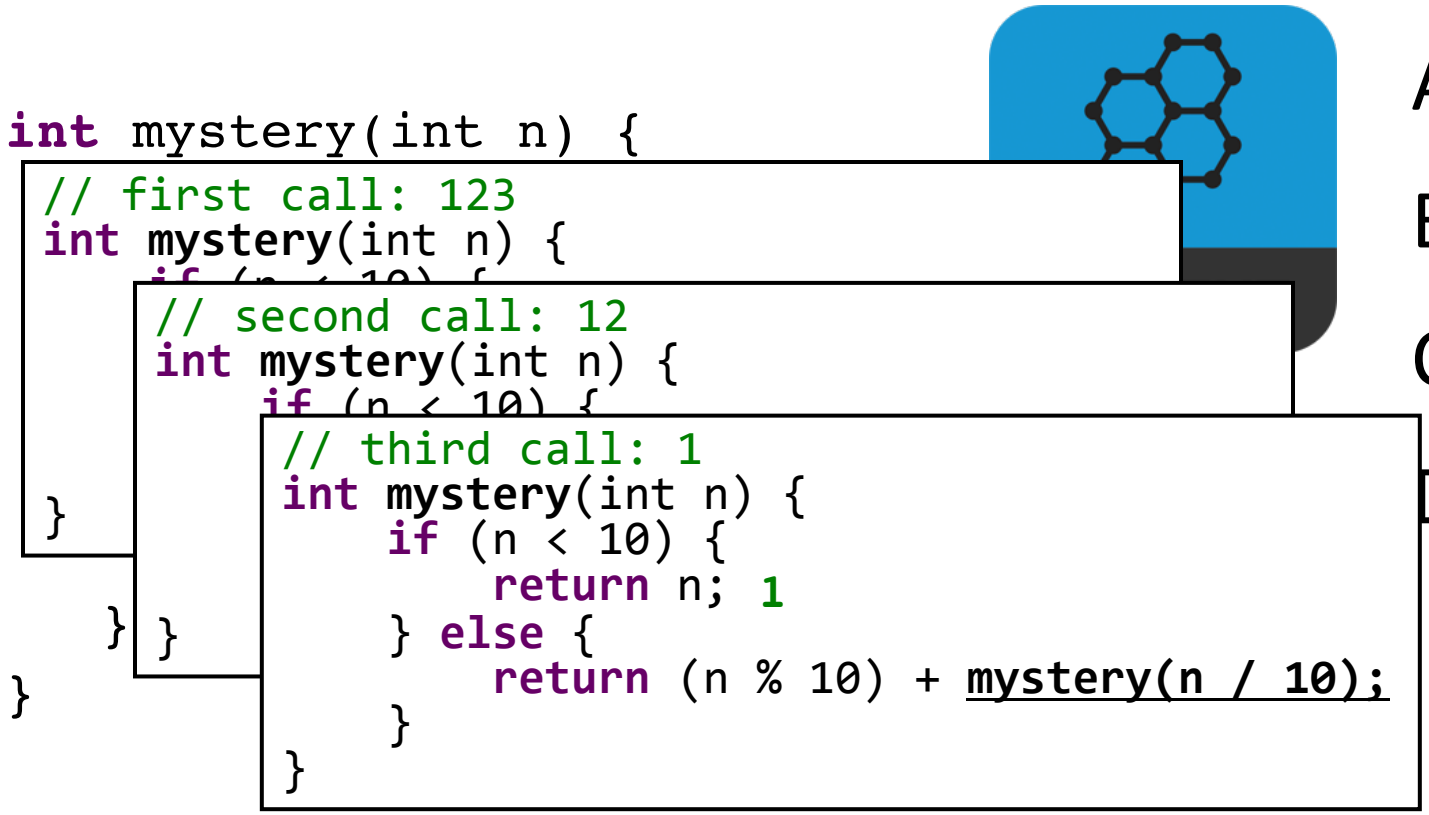
B: 6

C: 7

D: 8

What is the result of `mystery(123)`?

What Does This Code Do?



A: 5

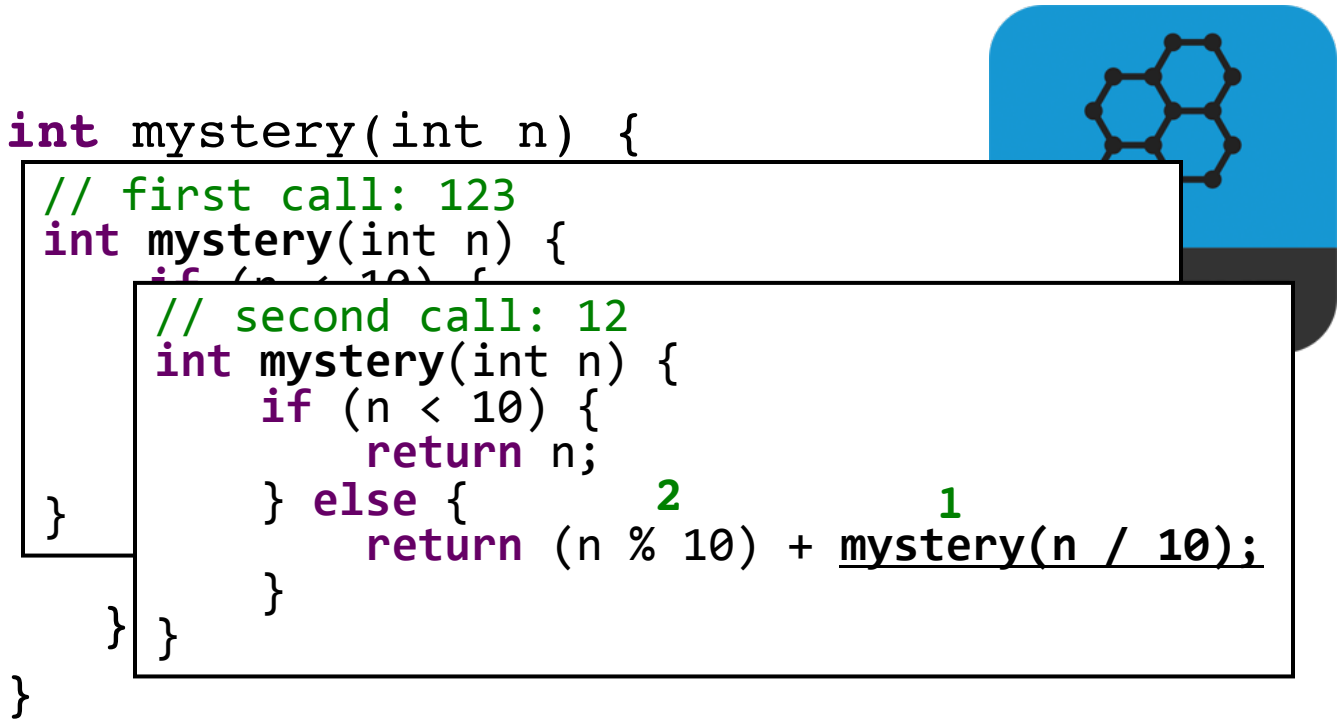
B: 6

C: 7

D: 8

What is the result of `mystery(123)`?

What Does This Code Do?



A: 5

B: 6

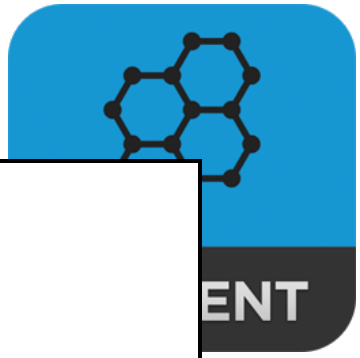
C: 7

D: 8

What is the result of `mystery(123)`?

What Does This Code Do?

```
int mystery(int n) {
    // first call: 123
    int mystery(int n) {
        if (n < 10) {
            return n;
        } else {                3              3
            return (n % 10) + mystery(n / 10);
        }
    }
    return checksum + total;
}
}
```



A: 5

B: 6

C: 7

D: 8

What is the result of `mystery(123)`?

_____

Is Palindrome

Write a recursive function `isPalindrome(string s)` accepts a `string` and returns `true` if it reads the same forwards as backwards.

```
isPalindrome("madam")                          → true
isPalindrome("racecar")                        → true
isPalindrome("step on no pets")                → true
isPalindrome("able was I ere I saw elba")      → true
isPalindrome("Q")                              → true
isPalindrome("Java")                           → false
isPalindrome("rotater")                        → false
isPalindrome("byebye")                         → false
isPalindrome("notion")                         → false
```

What is a good base case?

Is Palendrome

```
// Returns true if the given string reads the same
// forwards as backwards.
// Trivially true for empty or 1-letter strings.
bool isPalindrome(string s) {
    if (s.length() < 2) {   // base case
        return true;
    } else {                // recursive case
        if (s[0] != s[s.length() - 1]) {
            return false;
        }
        string middle = s.substr(1, s.length() - 2);
        return isPalindrome(middle);
    }
}
```

Hailstone

Print the sequences of numbers that you take to get from N until 1, using the Hailstone production rules:

If n == 1, you are done.

If n is odd your next number is 3*n + 1.

If n is even your next number is n / 2.

Hailstone

```
// Couts the sequence of numbers from n to one
// produced by the Hailstone (aka Collatz)
// procedure
void hailstone(int n) {
   cout << n << endl;
   if(n == 1) {
      return;
   } else {
      if(n % 2 == 0) {
         // n is even so we repeat with n/2
         hailstone(n / 2);
      } else {
         // n is odd so we repeat with 3 * n + 1
         hailstone(3 * n + 1);
      }
   }
}
```

Are The Recursive Calls Simpler?

```
// Couts the sequence of numbers from n to one
// produced by the Hailstone (aka Collatz)
```

3. When you make a recursive call it should be to a simpler instance (forward progress towards base case)

Well that seems to be true…

```
}
```

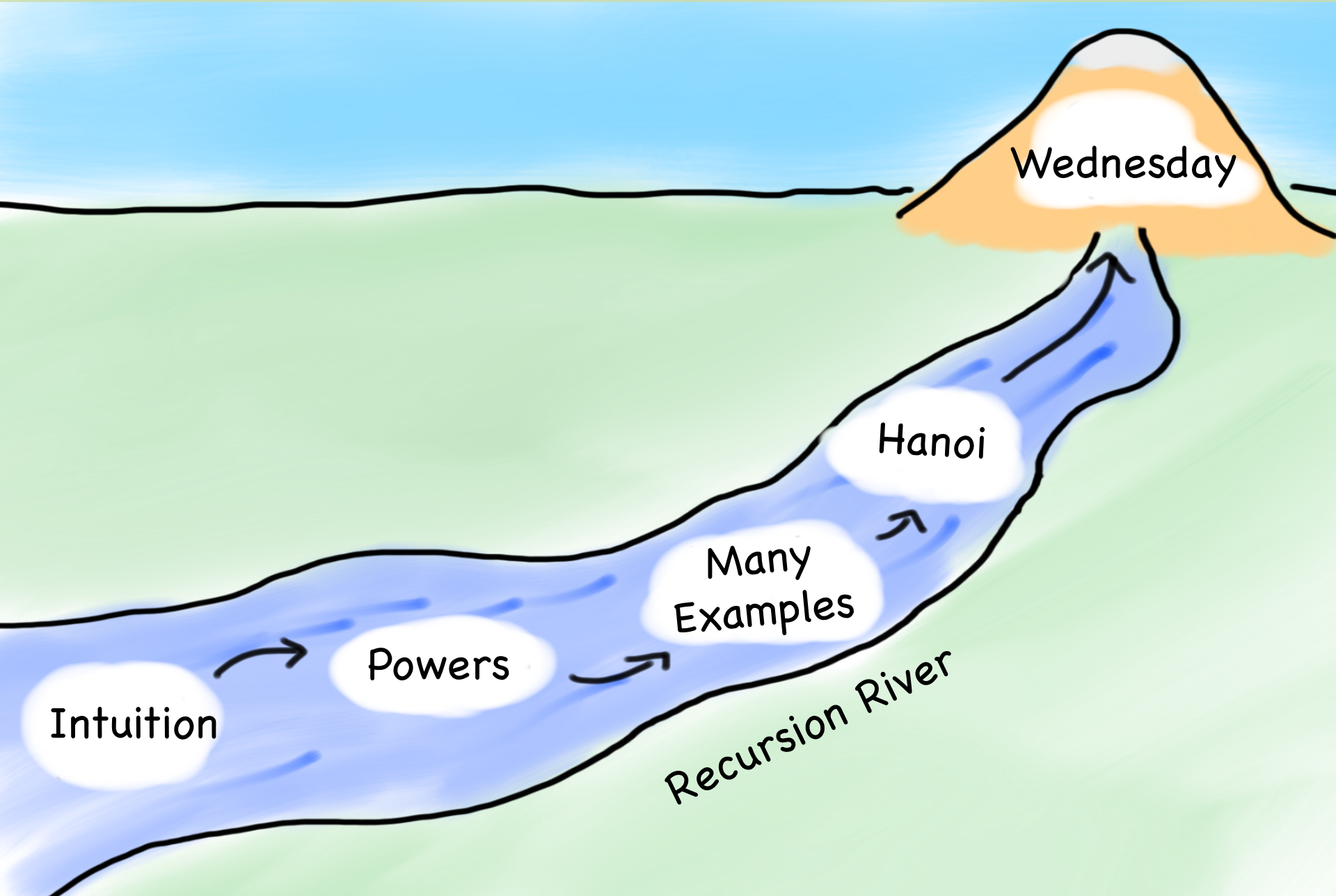
Works for numbers up to 5×10^{18}

Reward for proof

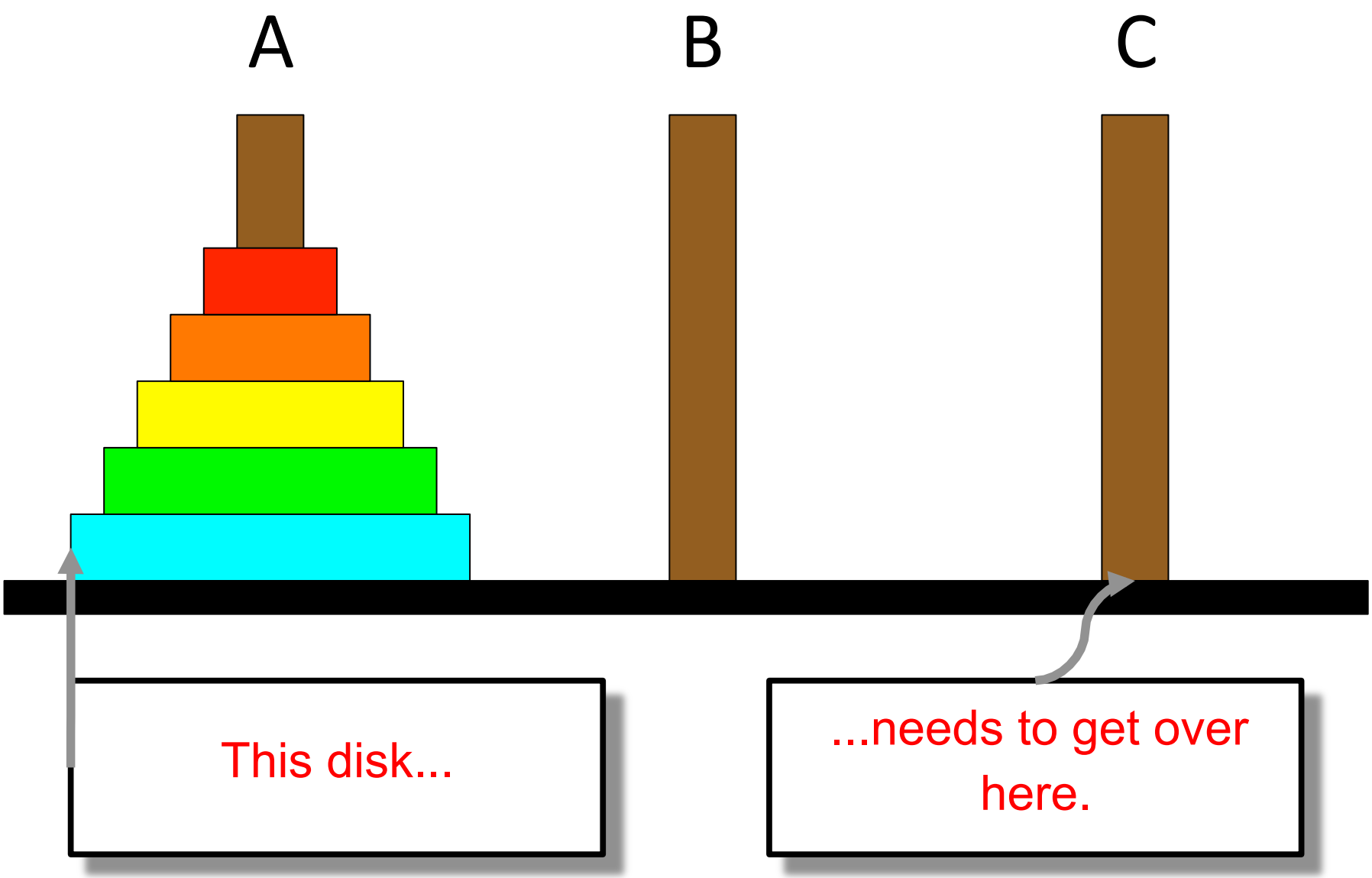
$1,400

And Here We Are

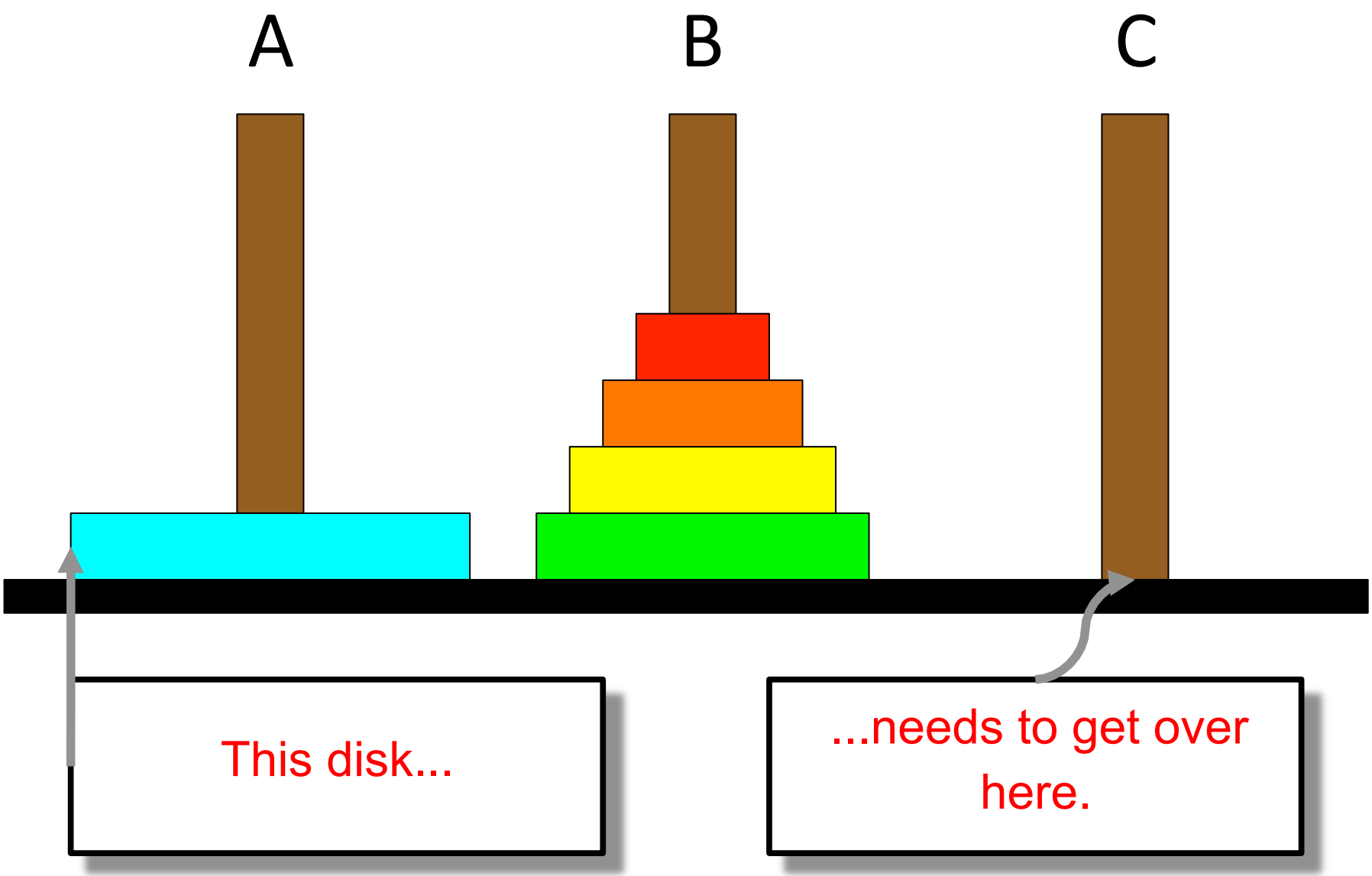
Today's Route



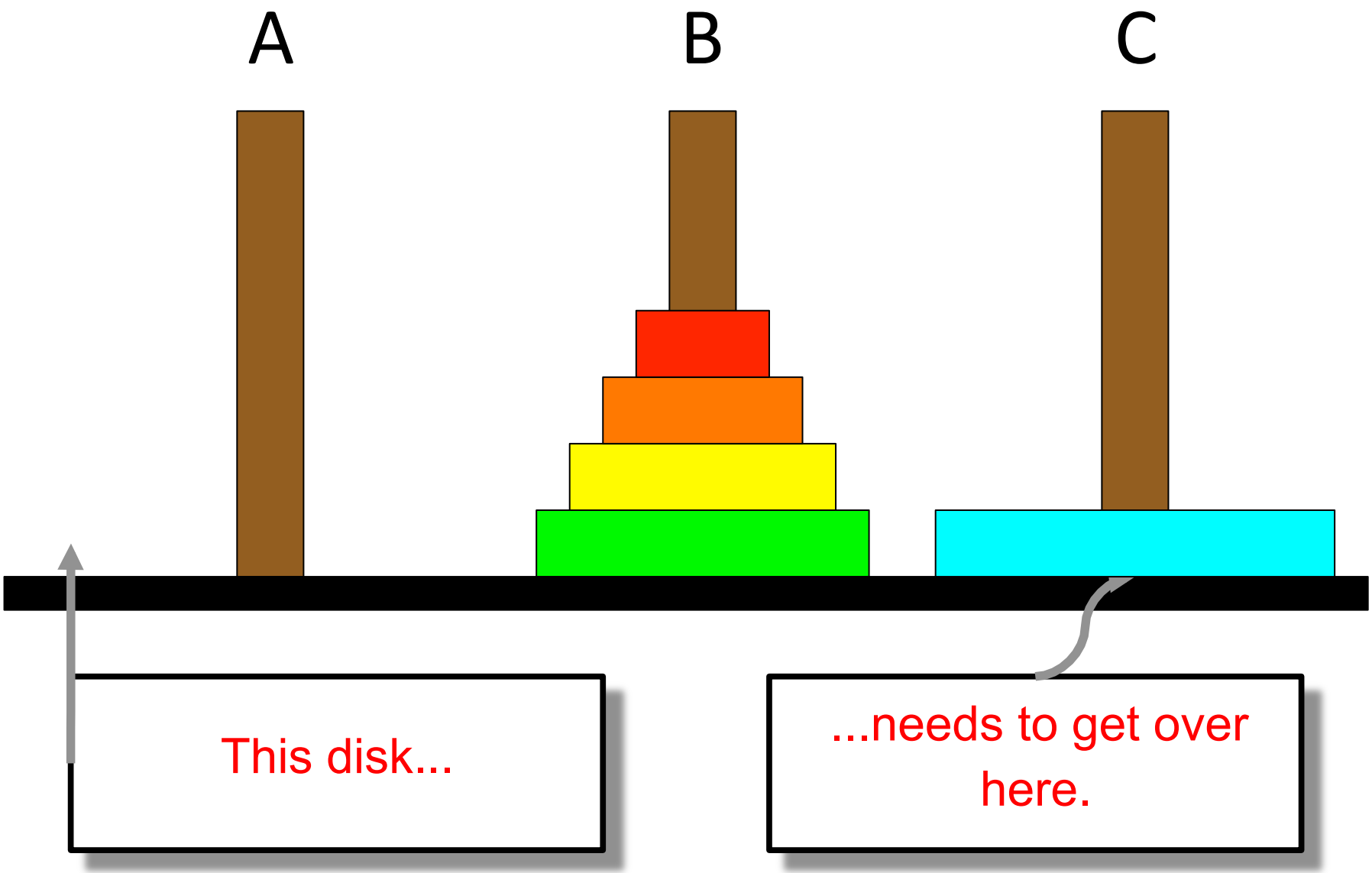
Tower of Hanoi Insight



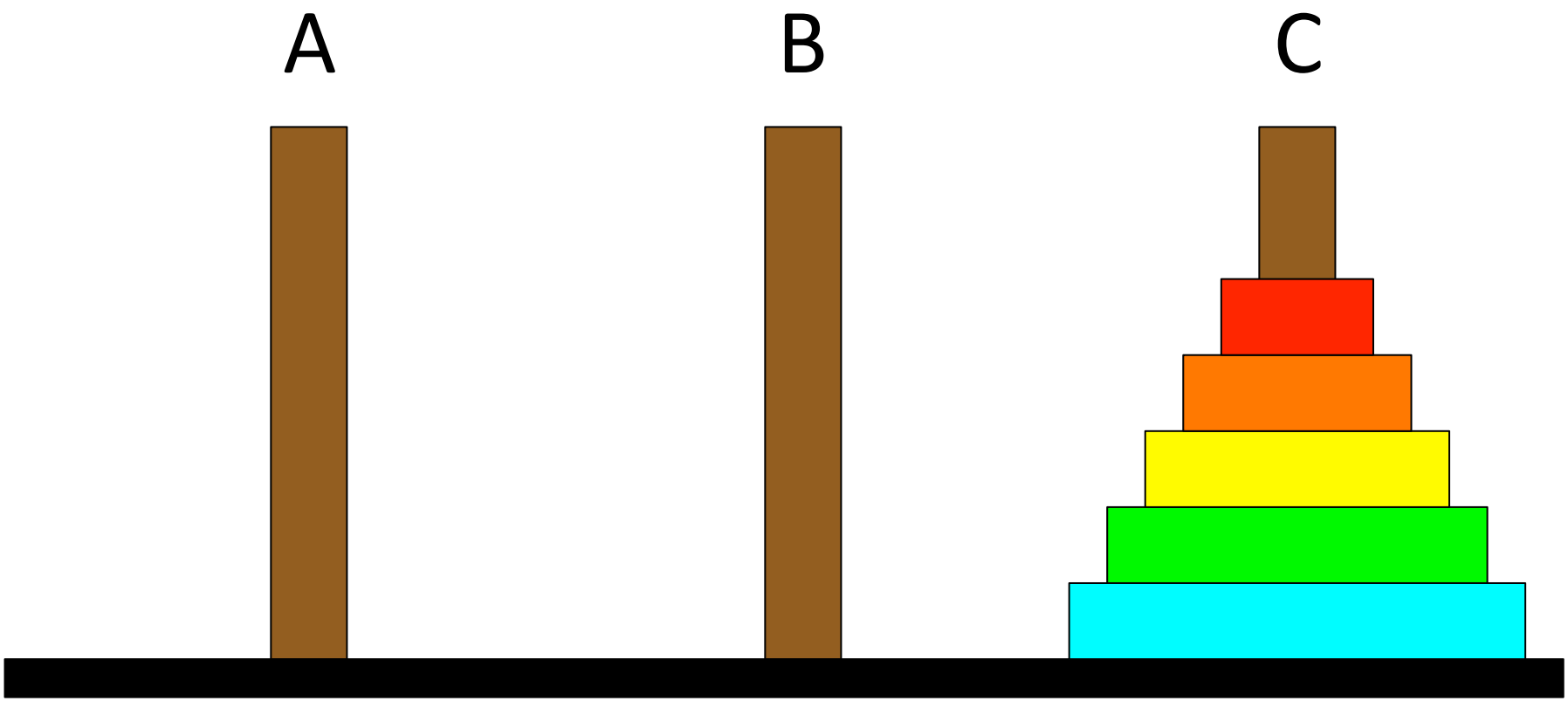
Tower of Hanoi Insight



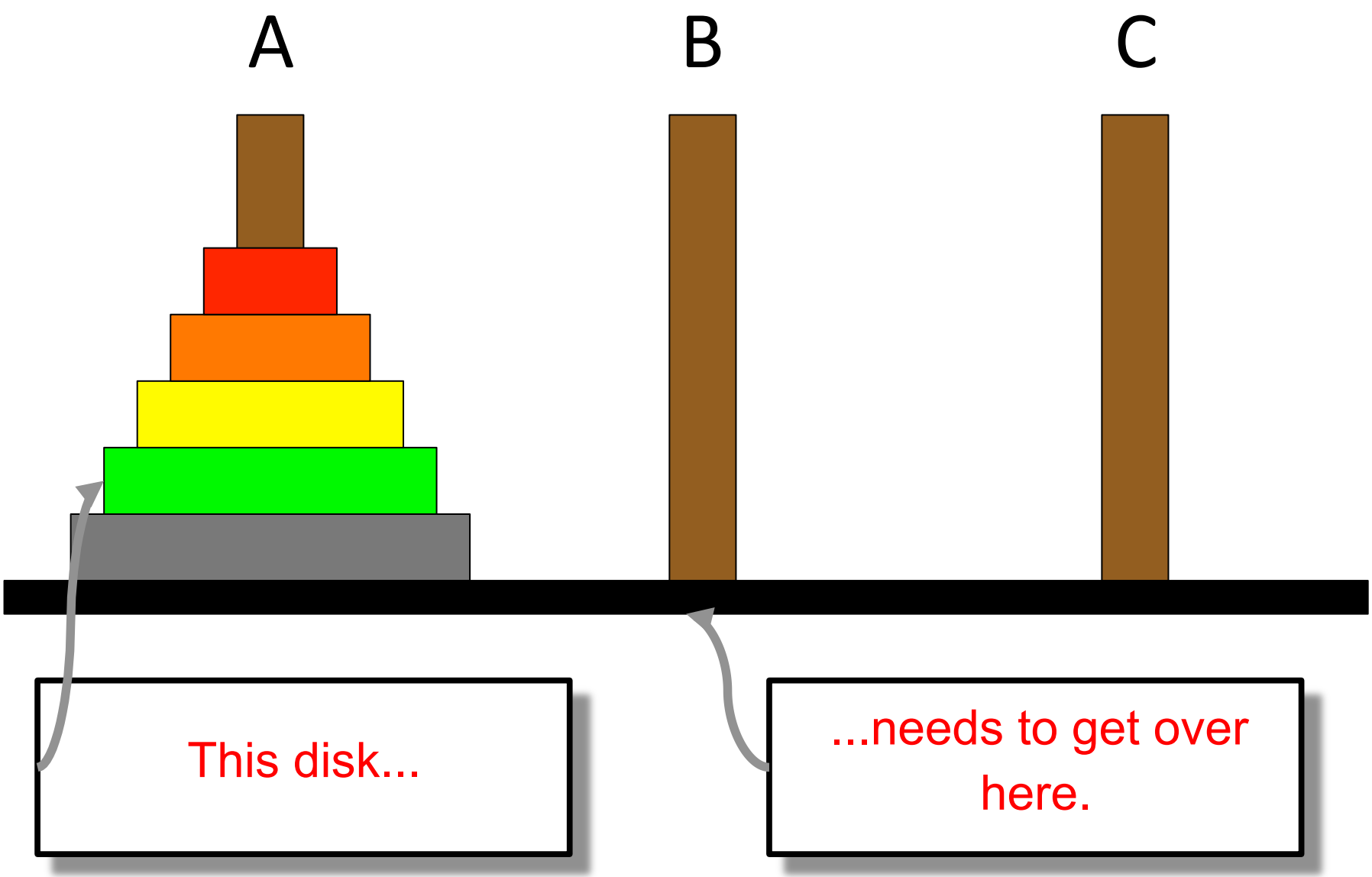
Tower of Hanoi Insight



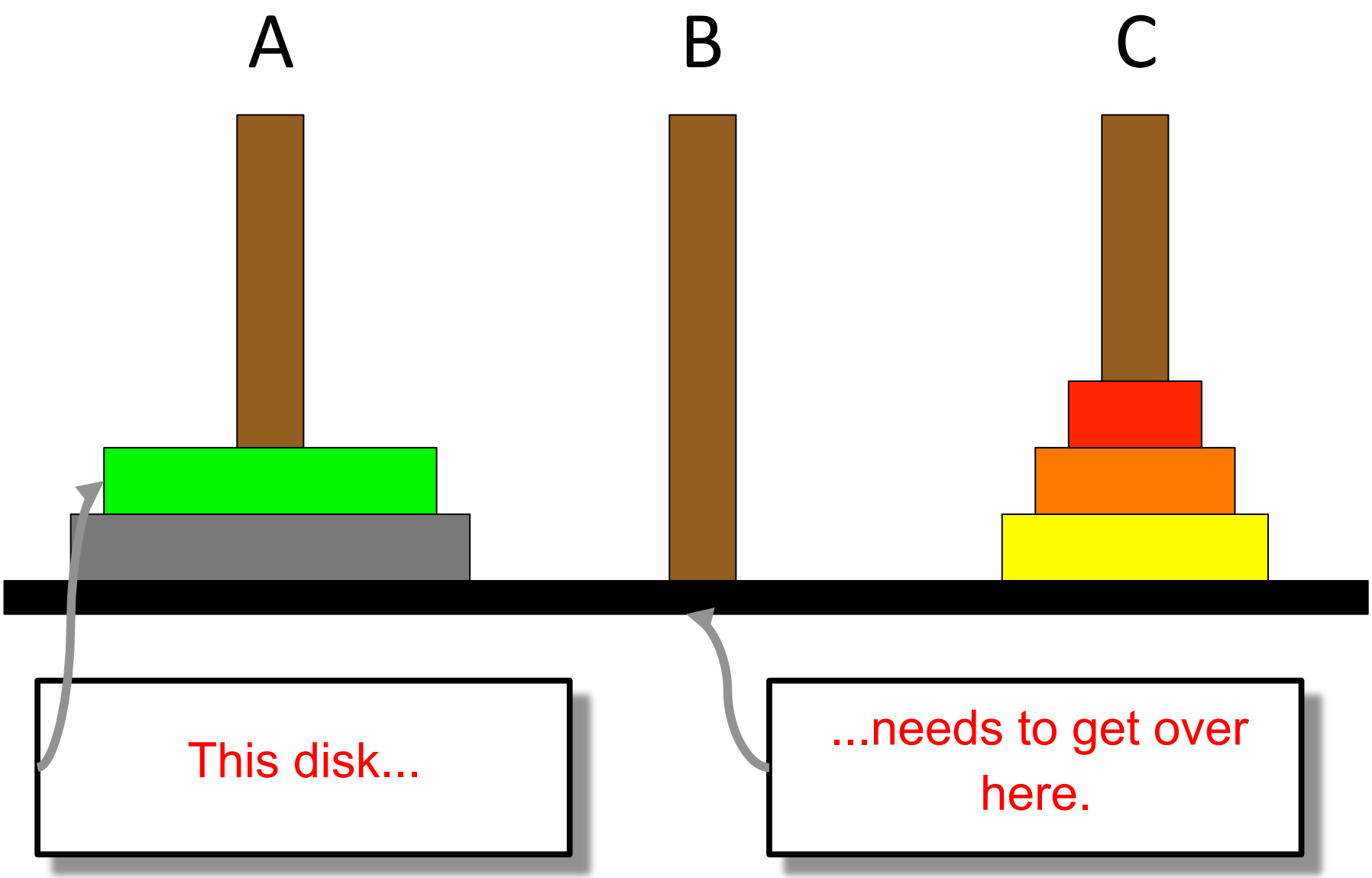
Tower of Hanoi Insight



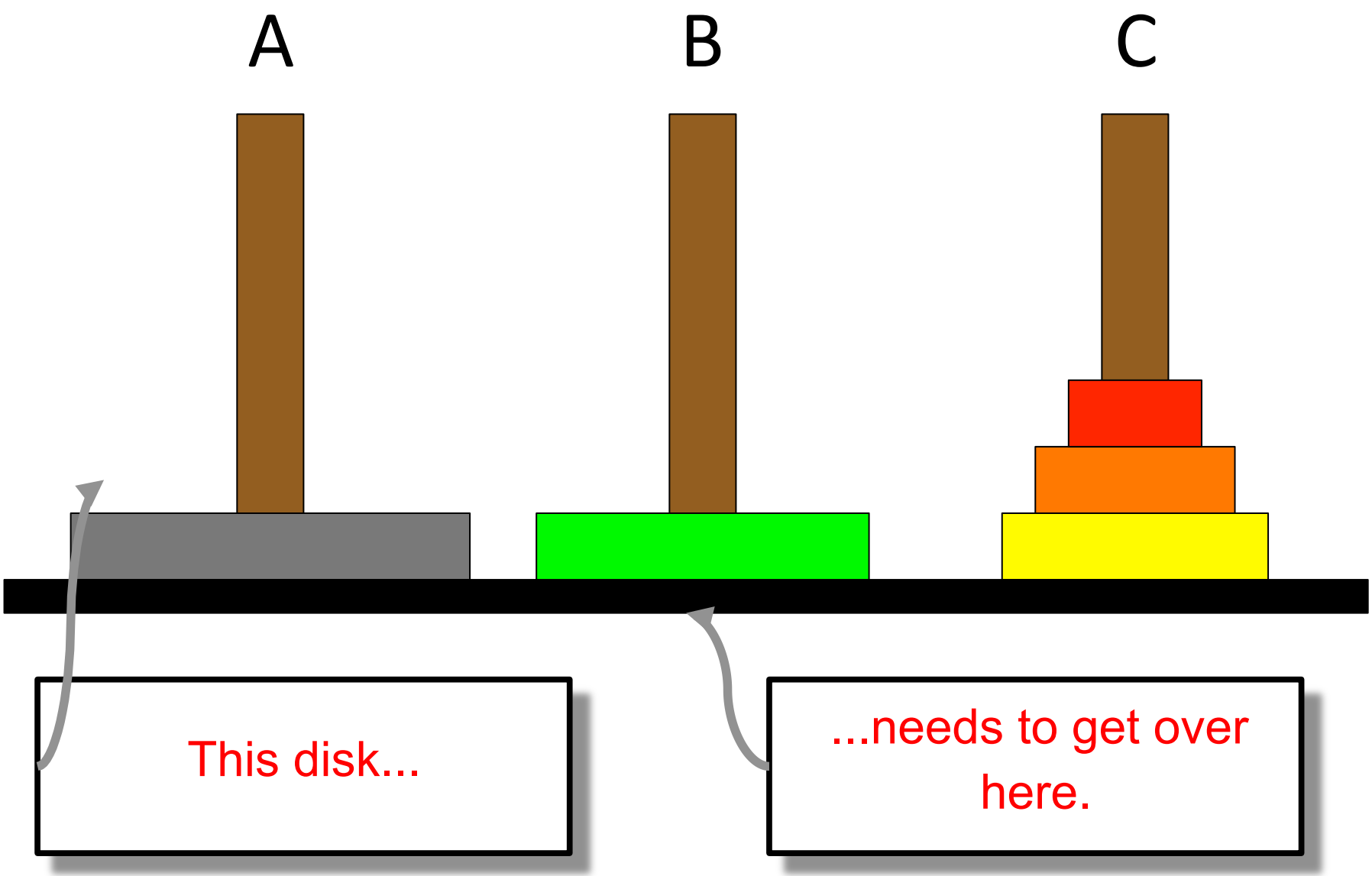
Tower of Hanoi Insight



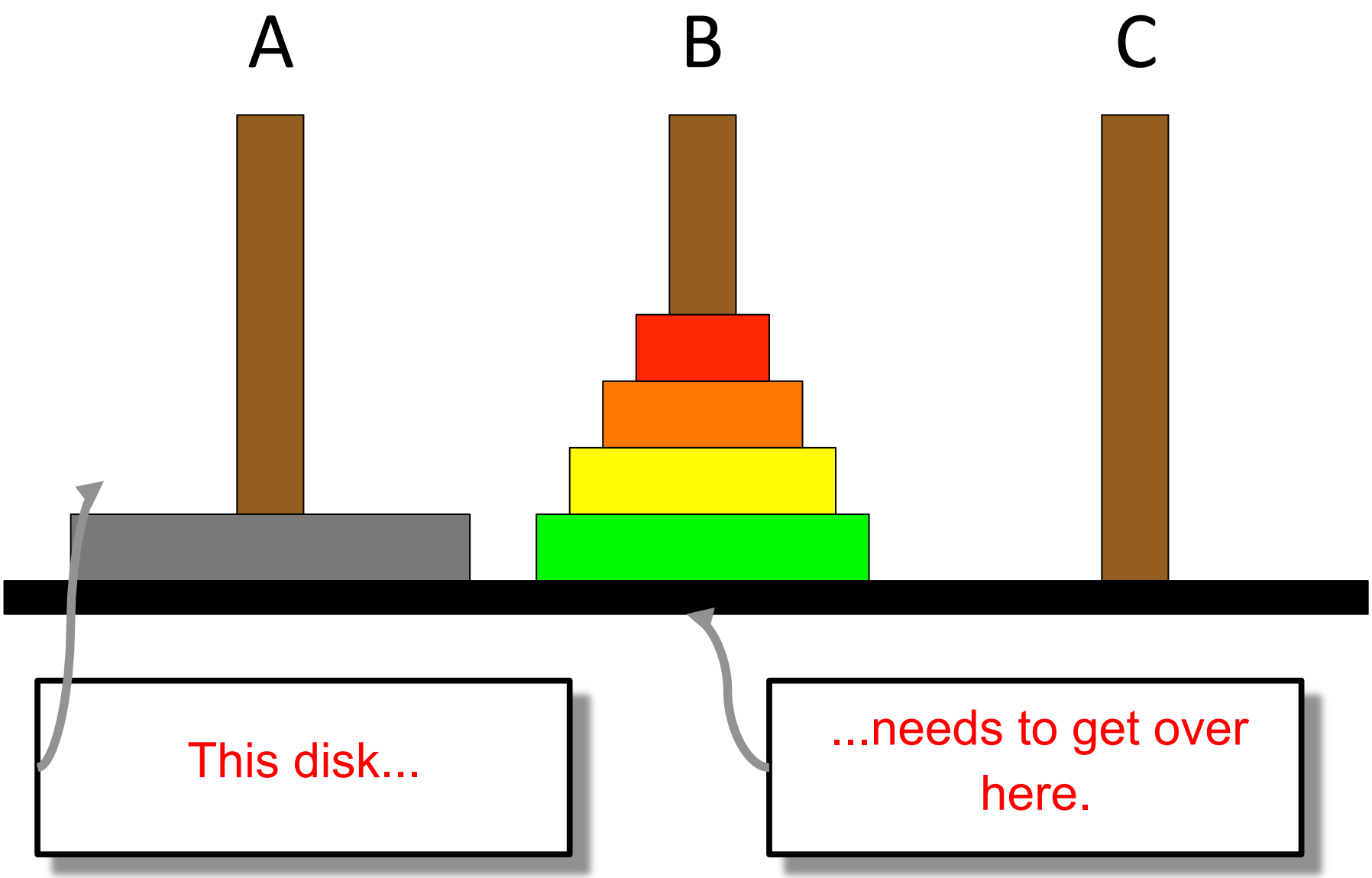
Tower of Hanoi Insight



Tower of Hanoi Insight



Tower of Hanoi Insight

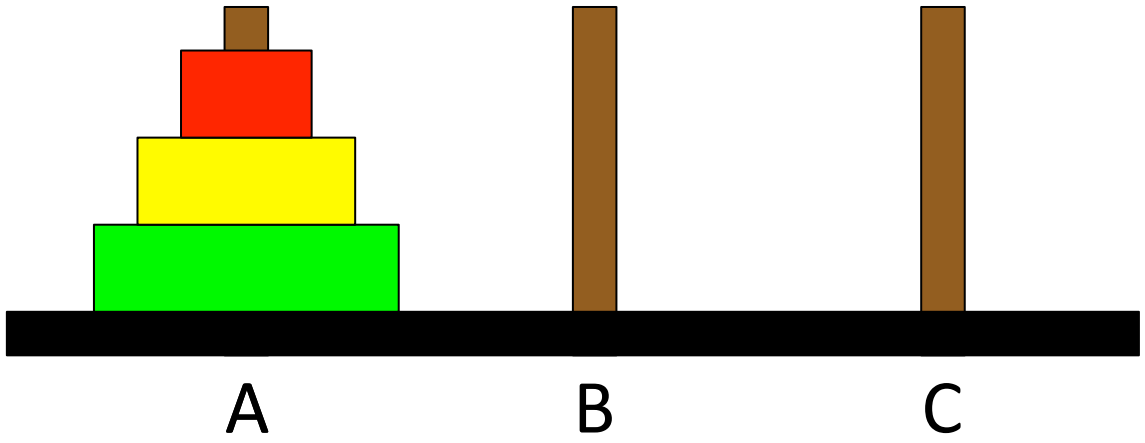


Base Case

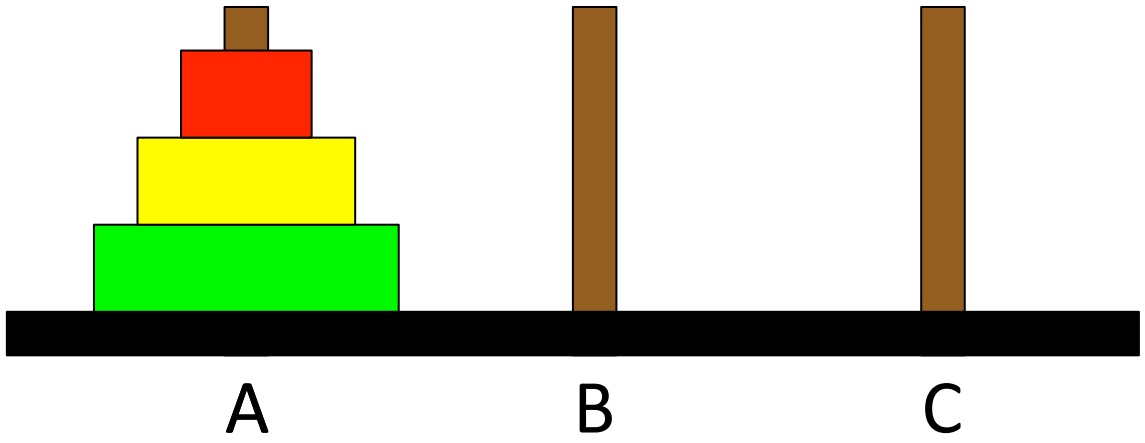
- We need to find a very simple case that we can solve directly in order for the recursion to work.
- If the tower has size one, we can just move that single disk from the source to the destination.


Qt

```
int main() {
    moveTower(3, 'a', 'c', 'b');
}
```



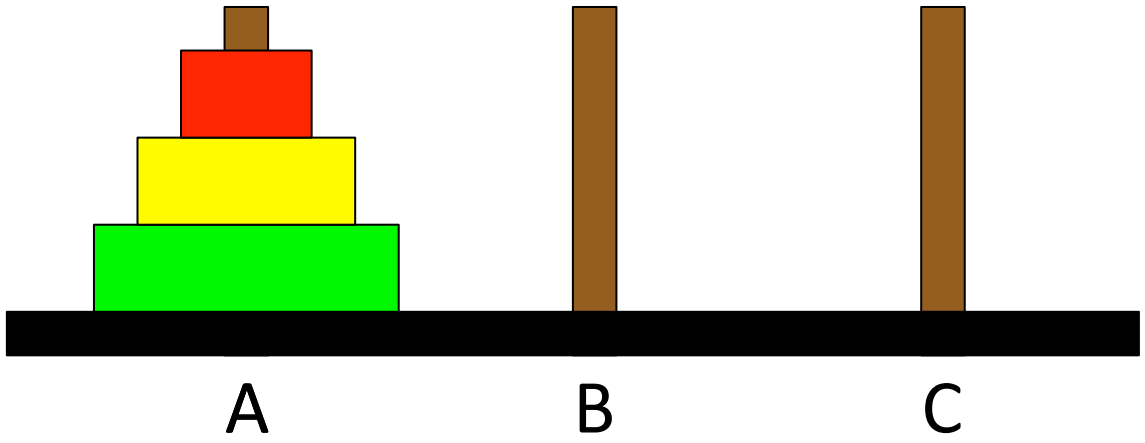
```
int main() {
    moveTower(3, 'a', 'c', 'b');
}
```



```
int main() {
    [illegible]
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

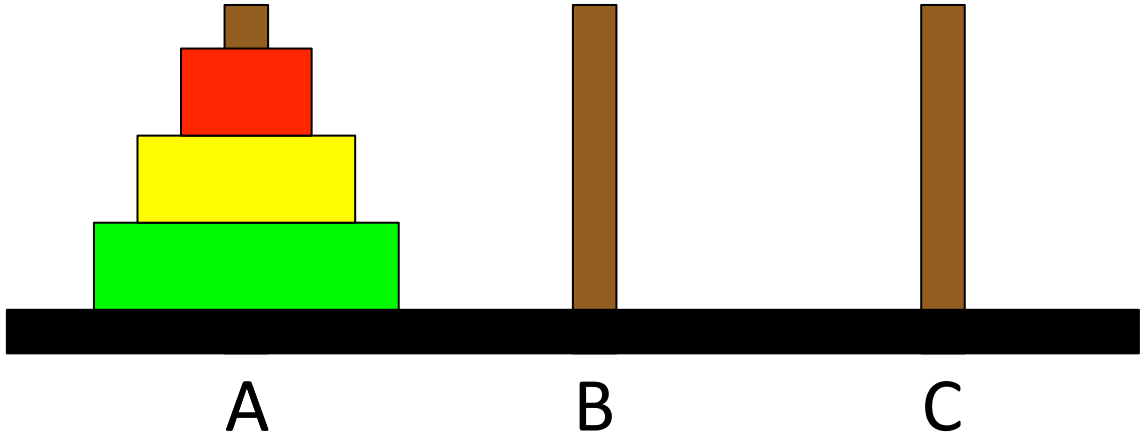
n	3	from	a	to	c	temp	b



```
int main() {
    [illegible]
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

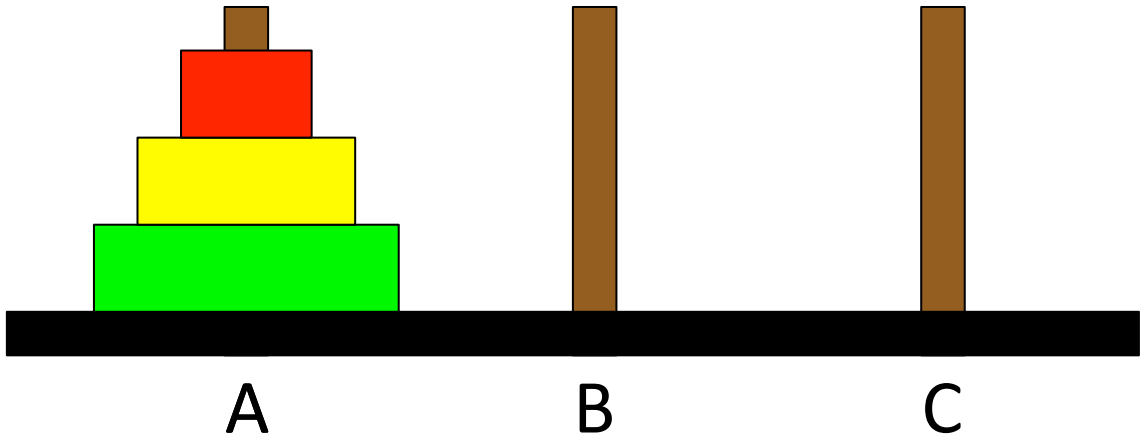
n	3	from	a	to	c	temp	b



```
int main() {
    [illegible]
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

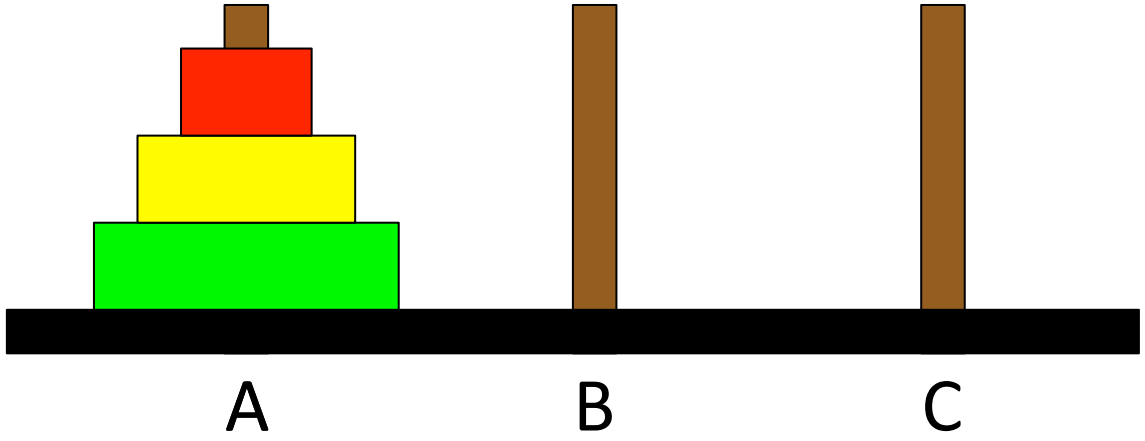
n	3	from	a	to	c	temp	b



```
int main() {
    [illegible]
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	3	from	a	to	c	temp	b

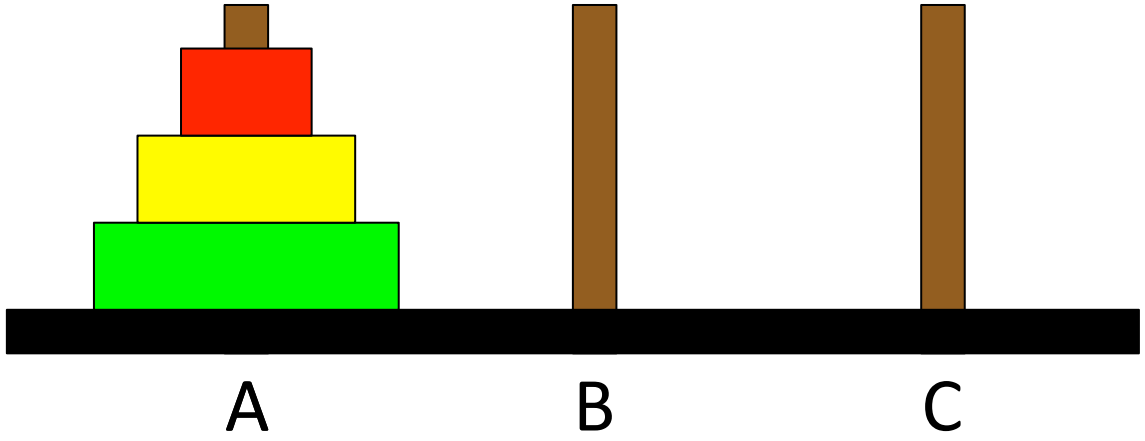


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

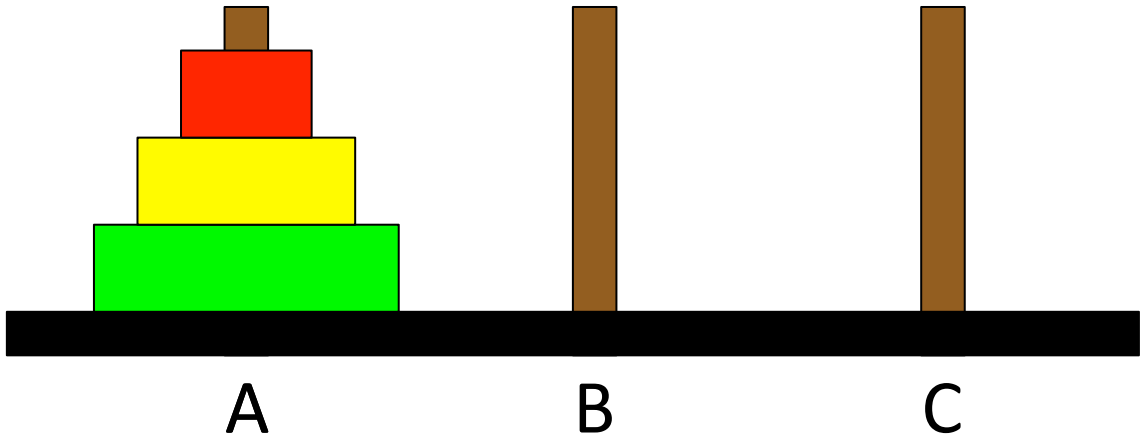


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

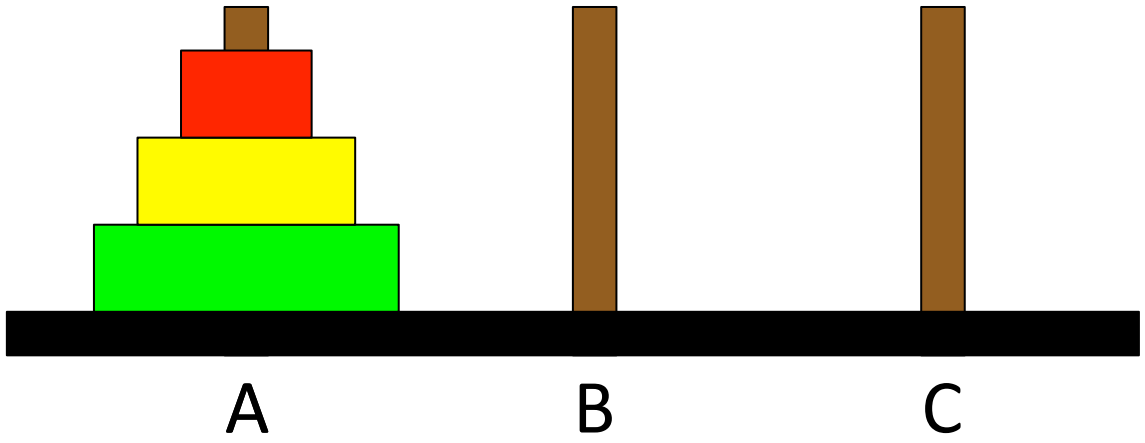


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

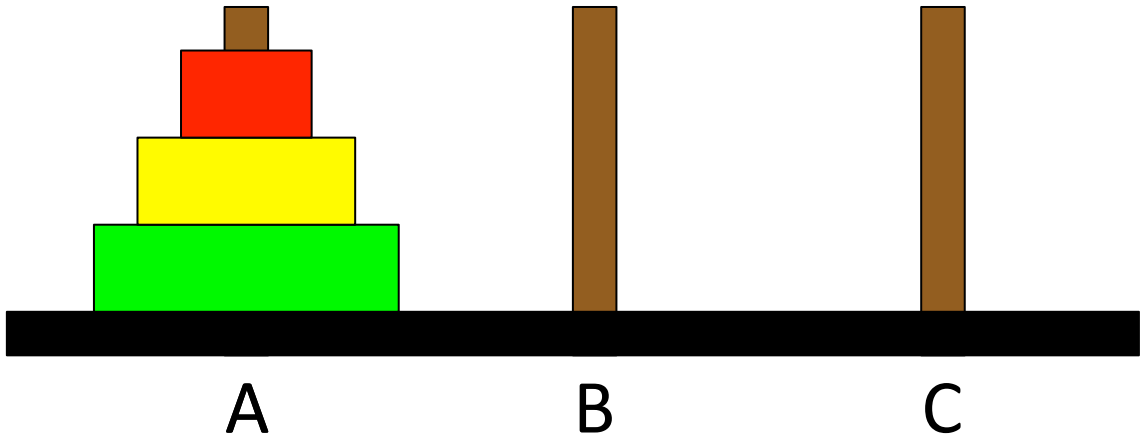


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c



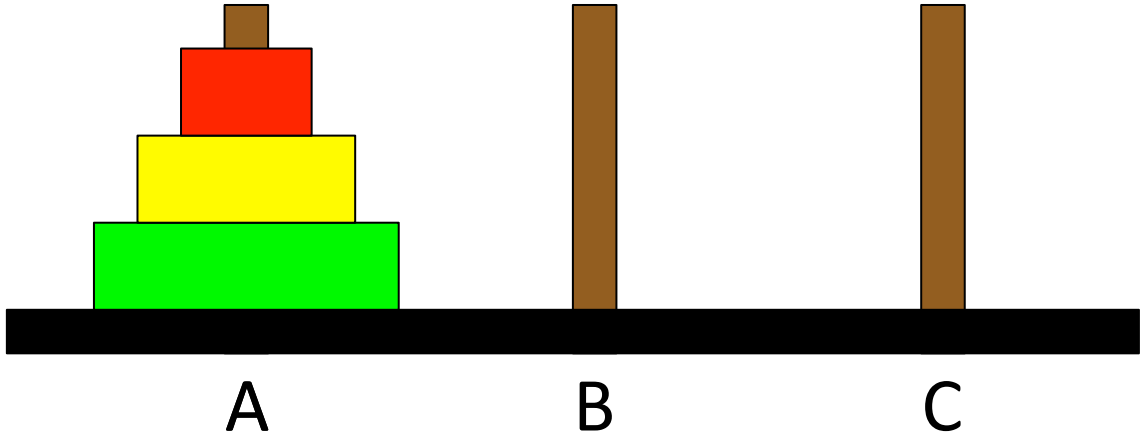
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



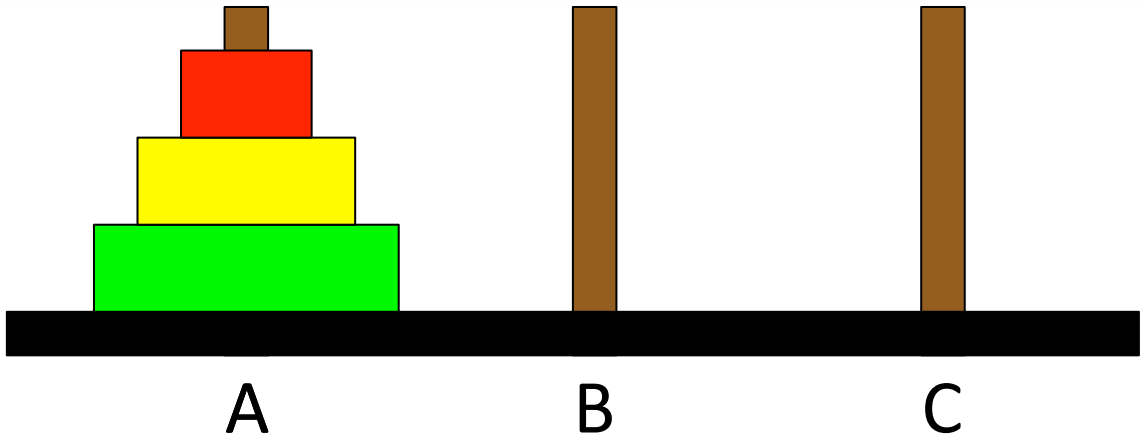
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



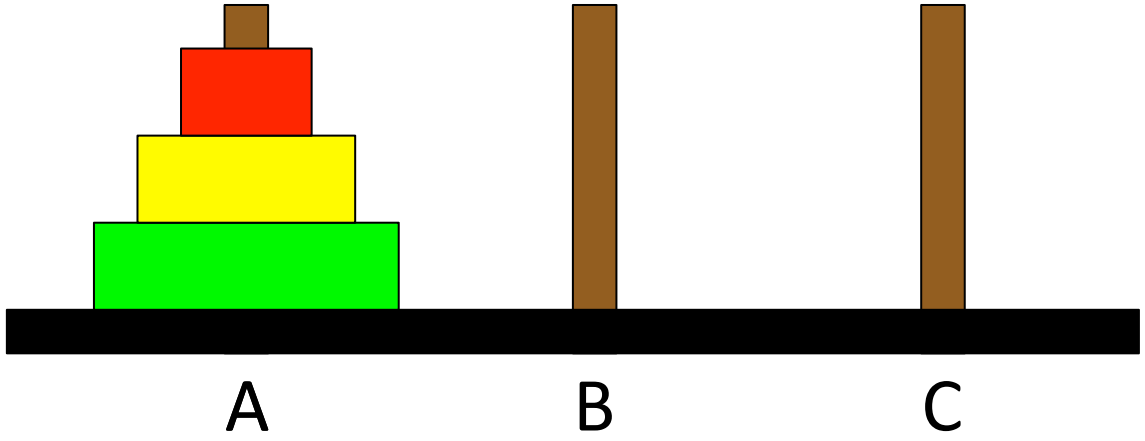
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



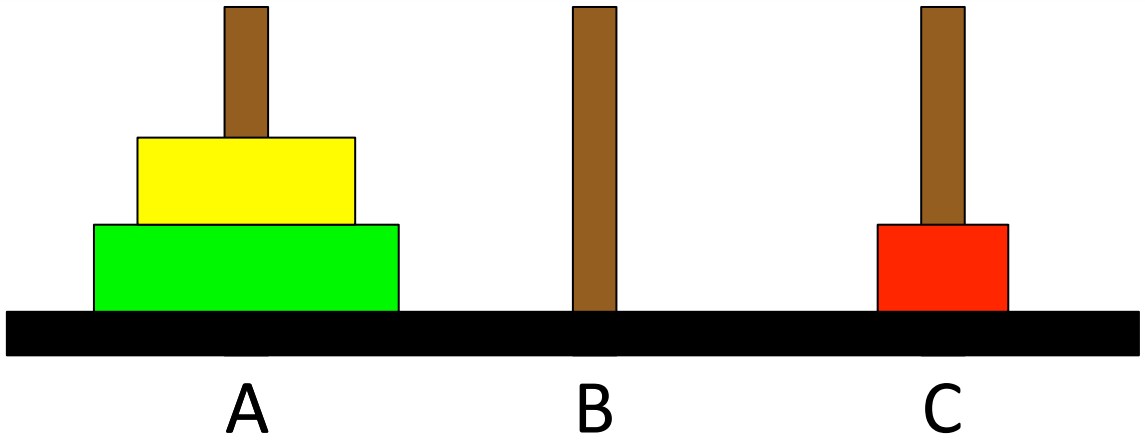
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b

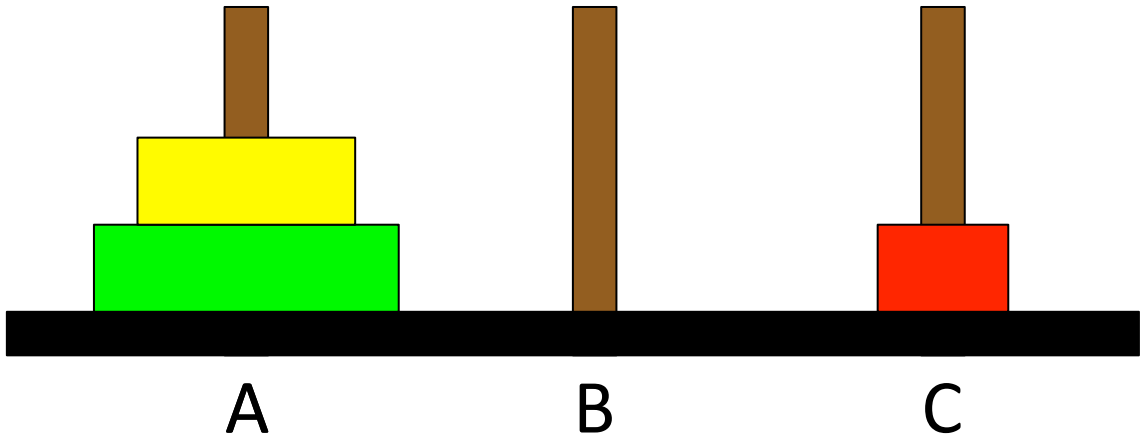


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

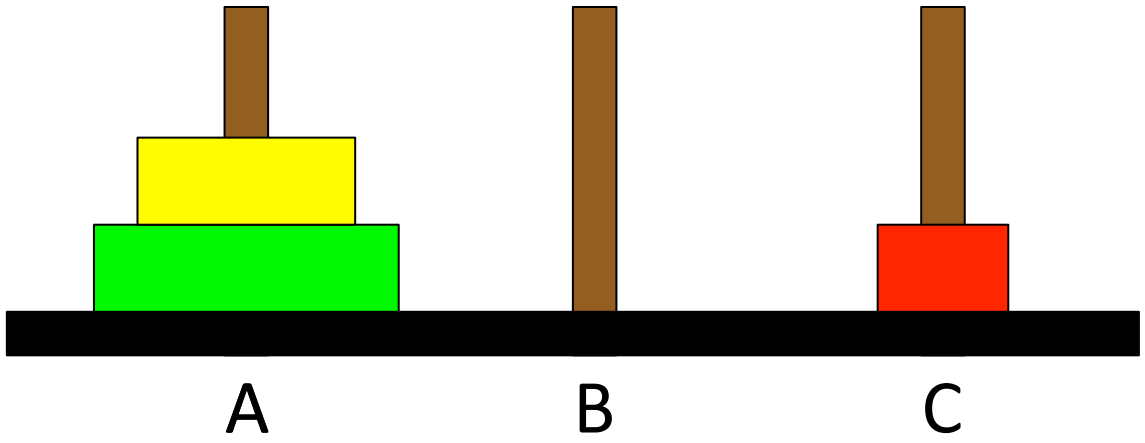


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

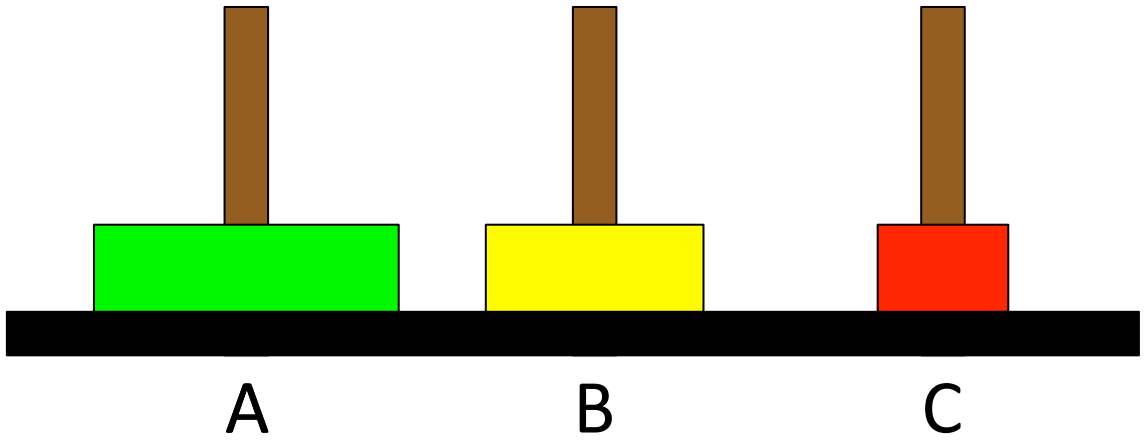


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n 2 from a to b temp c

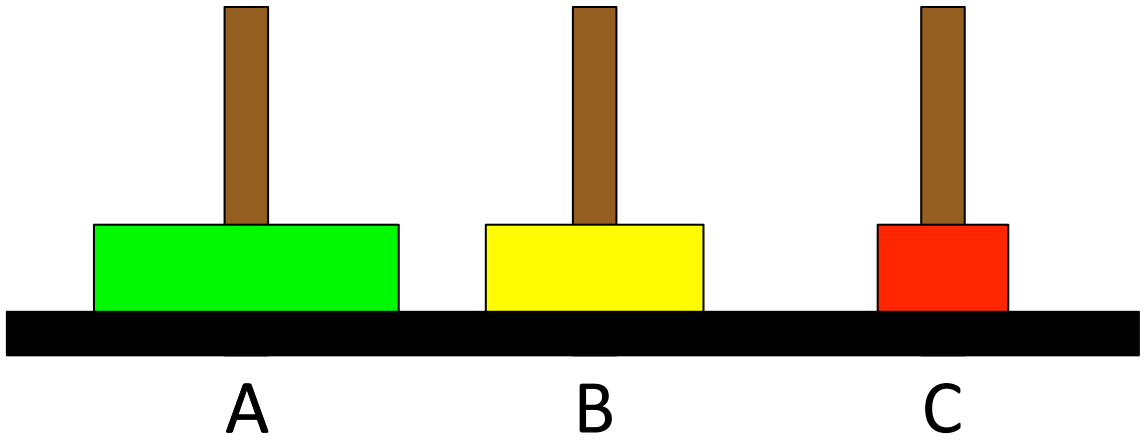


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n 2 from a to b temp c



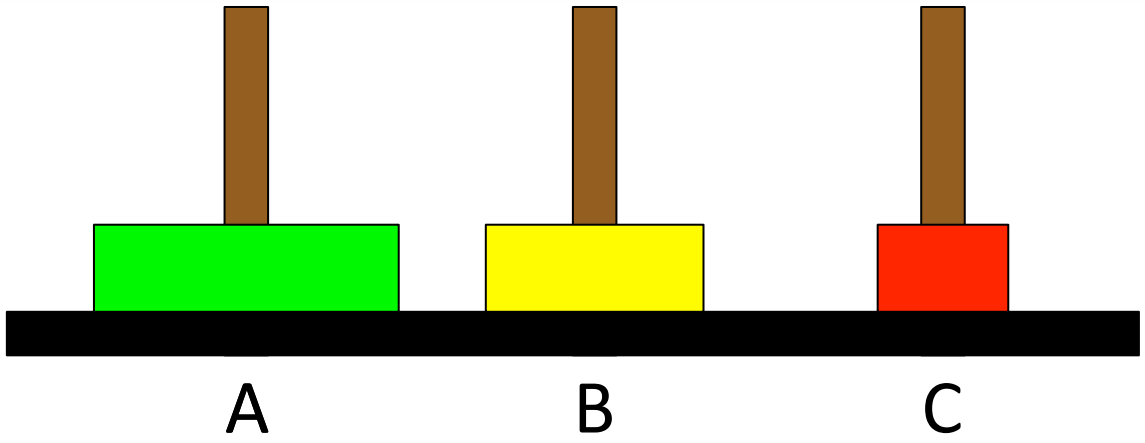
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n 1 from c to b temp a



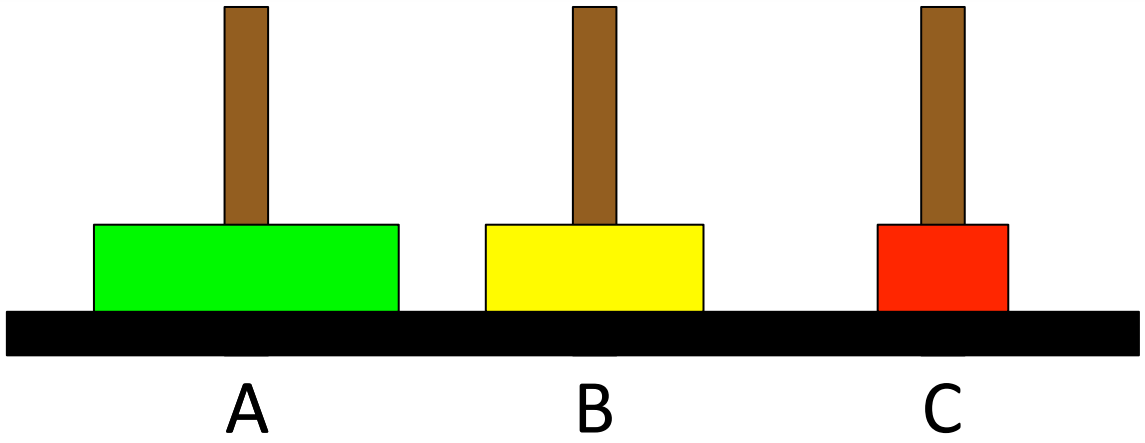
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	c	to	b	temp	a



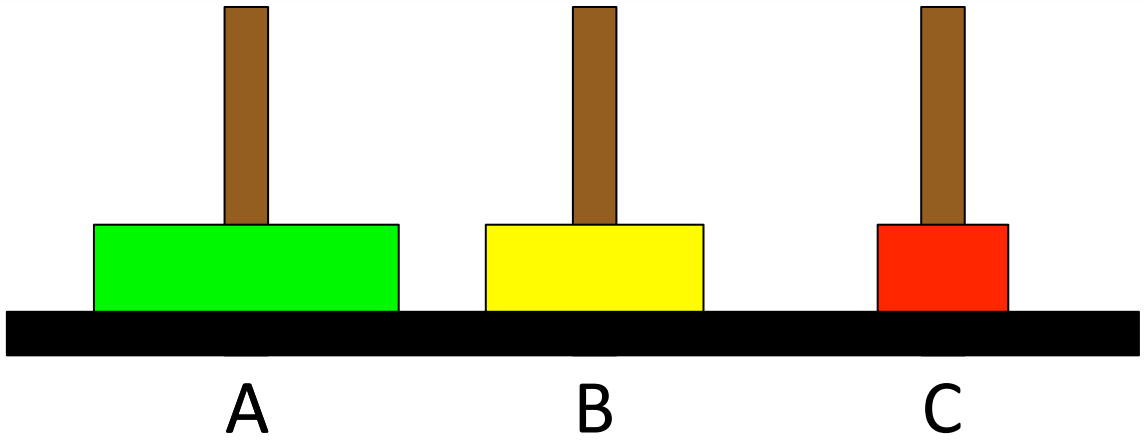
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	c	to	b	temp	a



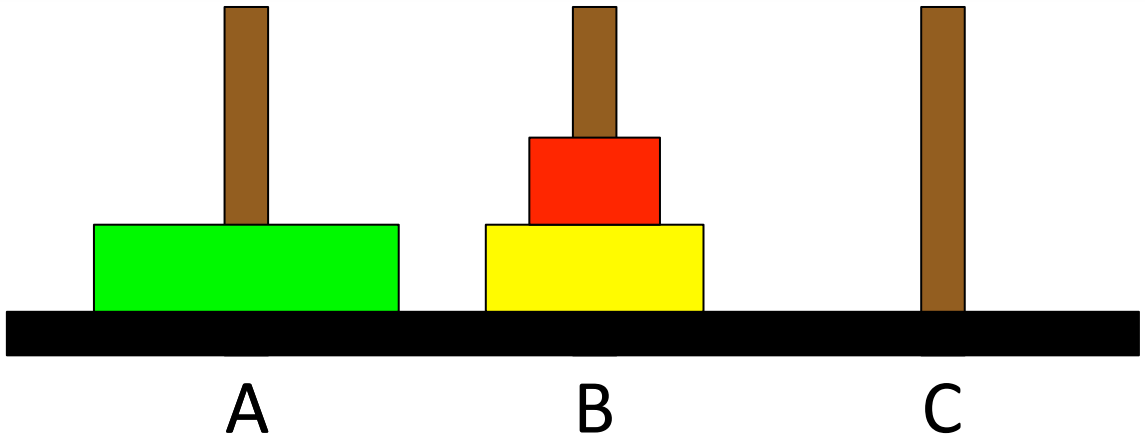
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	c	to	b	temp	a

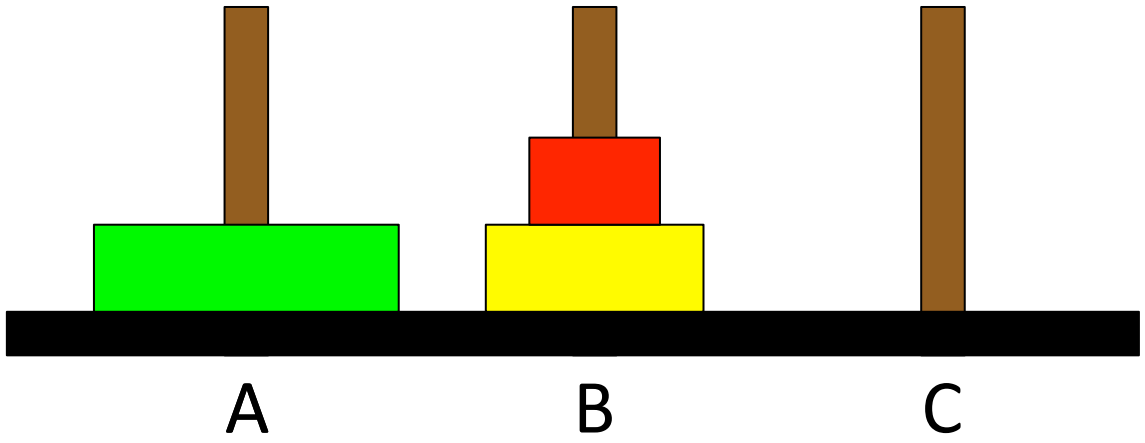


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

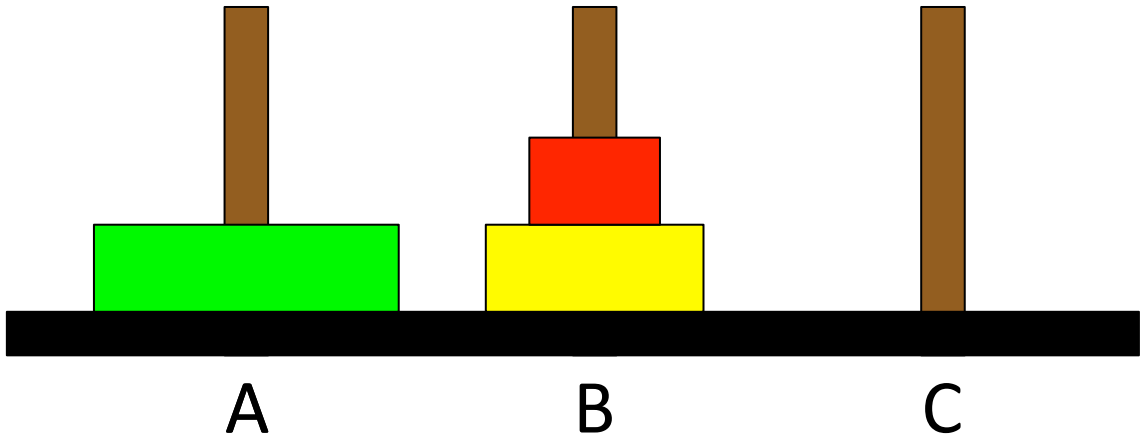
n	2	from	a	to	b	temp	c



```
int main() {
    [illegible]
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

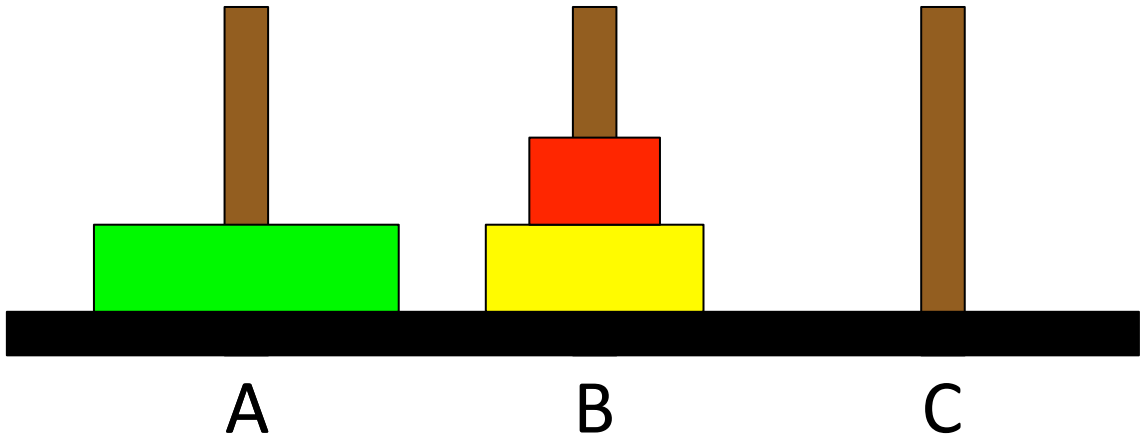
n	3	from	a	to	c	temp	b



```
int main() {
    [illegible]
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

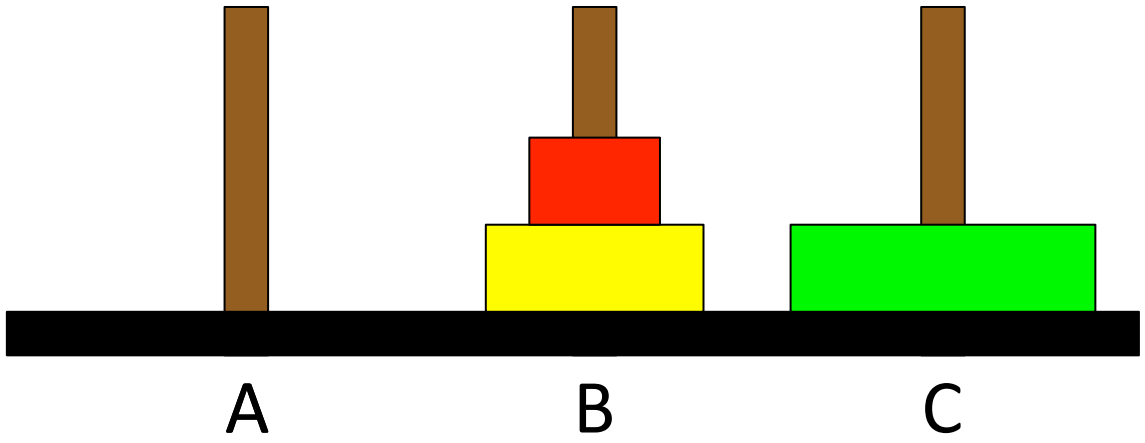
n	3	from	a	to	c	temp	b



```
int main() {
    [illegible]
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

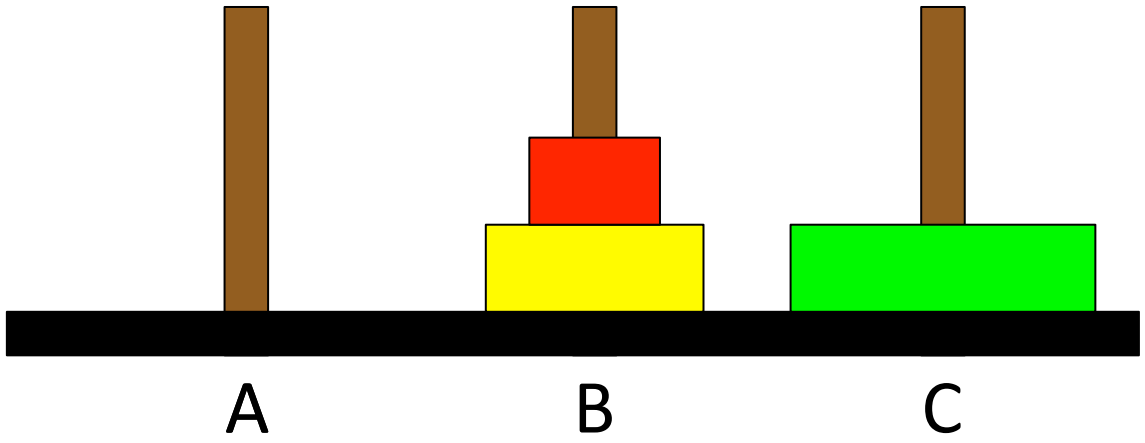
n	3	from	a	to	c	temp	b



```
int main() {
    moveTower(3, 'a', 'b', 'c');
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	3	from	a	to	c	temp	b

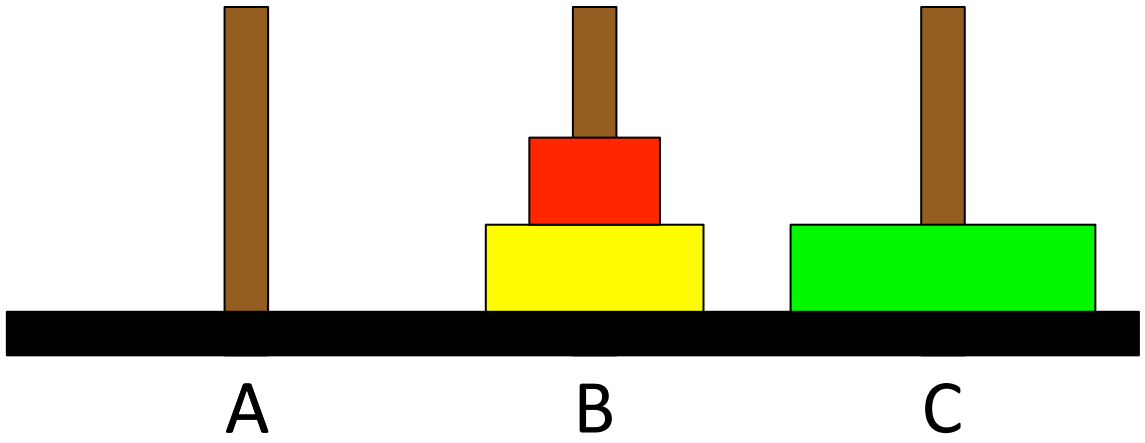


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

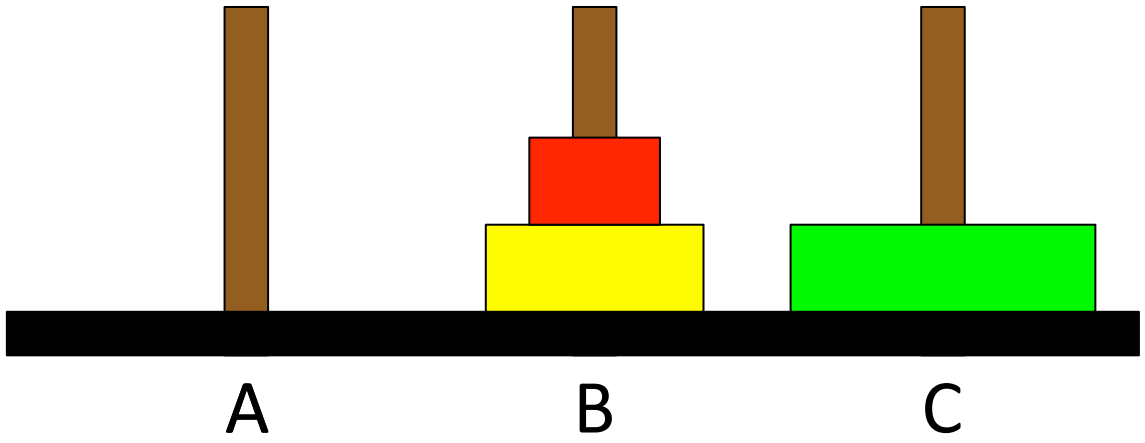


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

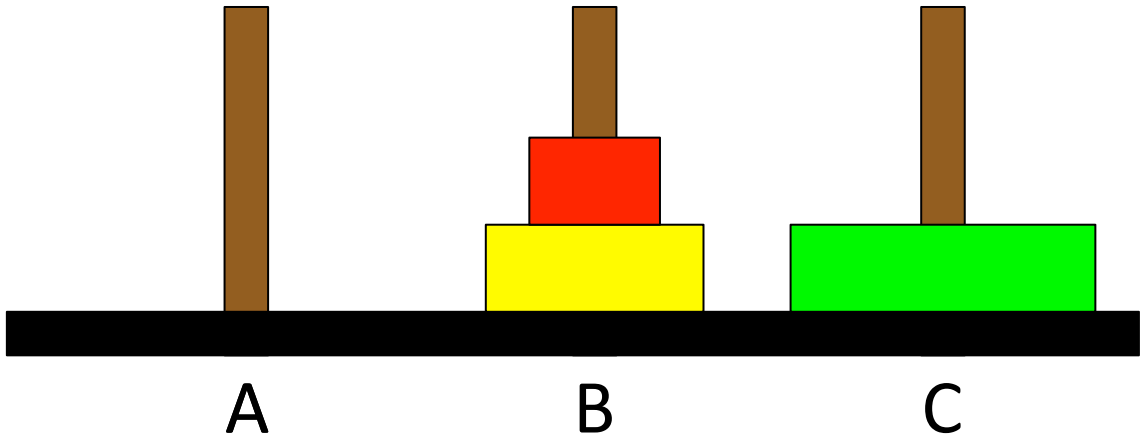


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

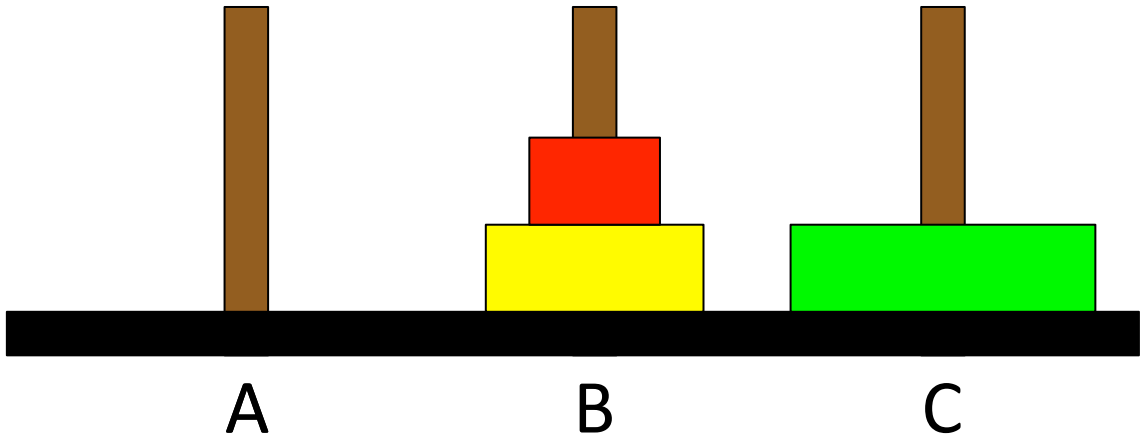


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a



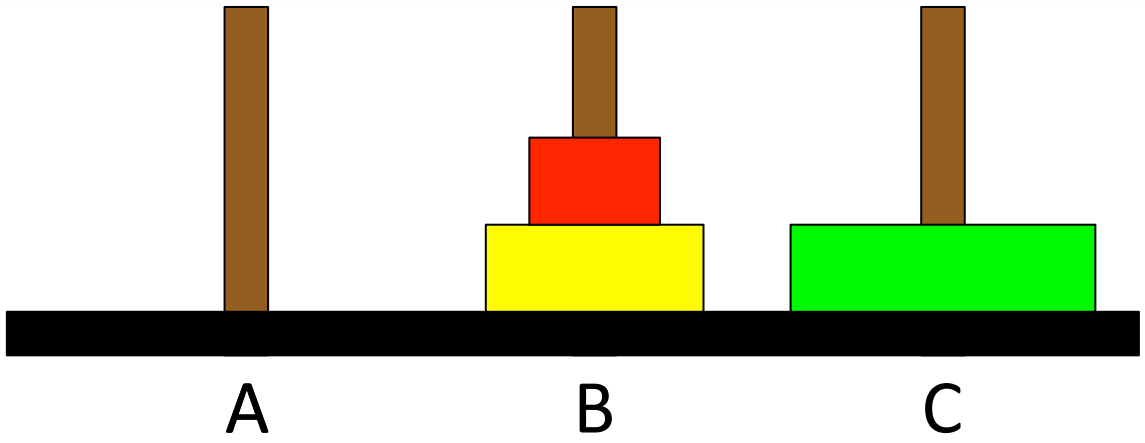
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n 1 from b to a temp c



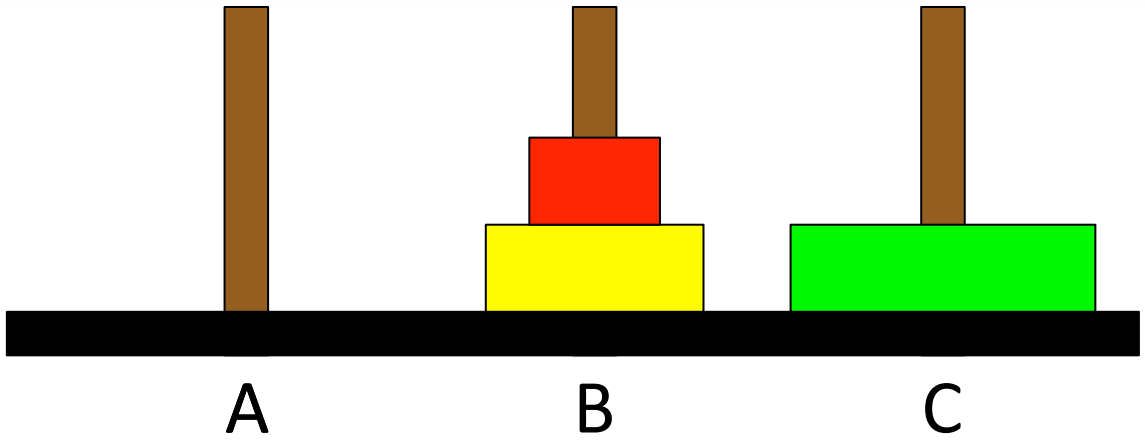
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n 1 from b to a temp c



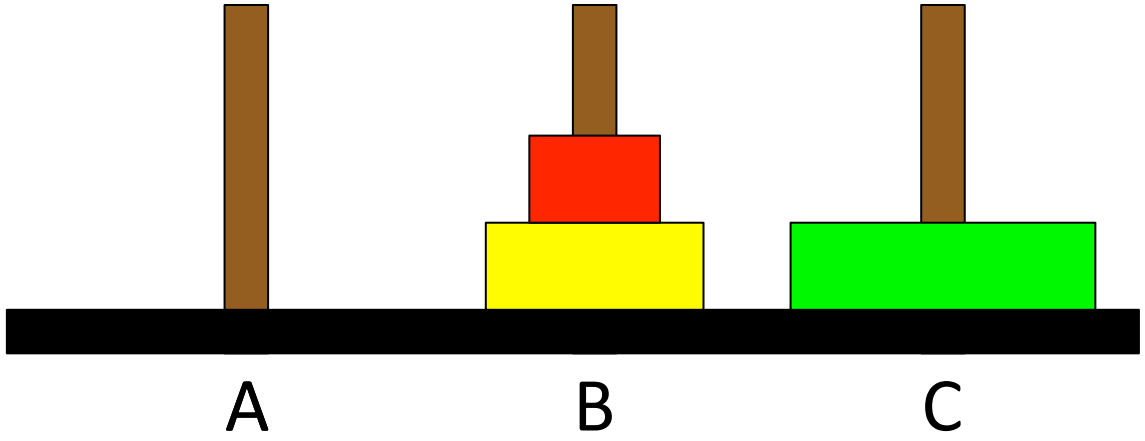
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	b	to	a	temp	c



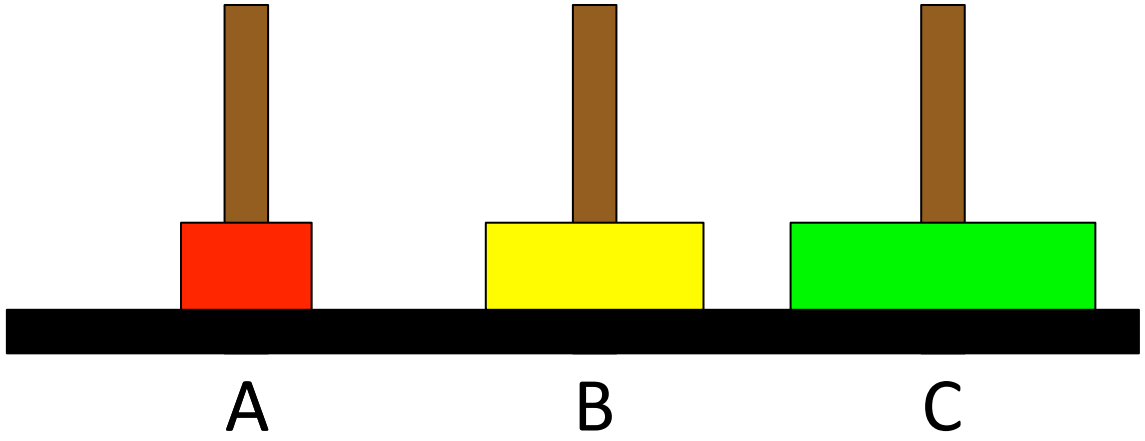
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	b	to	a	temp	c

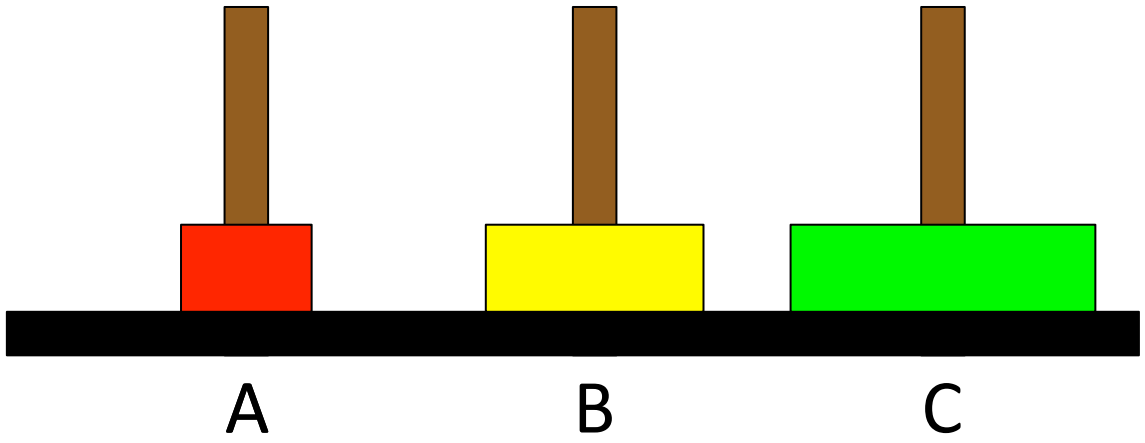


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

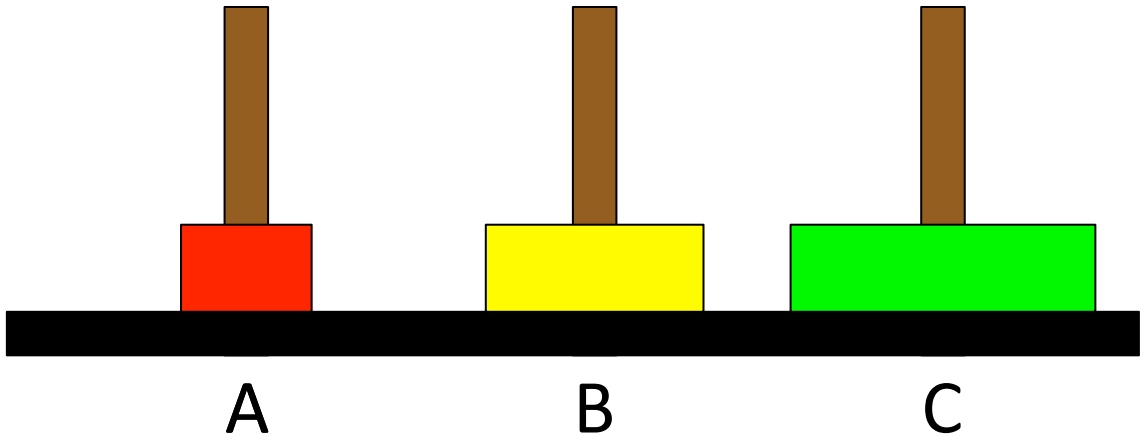


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

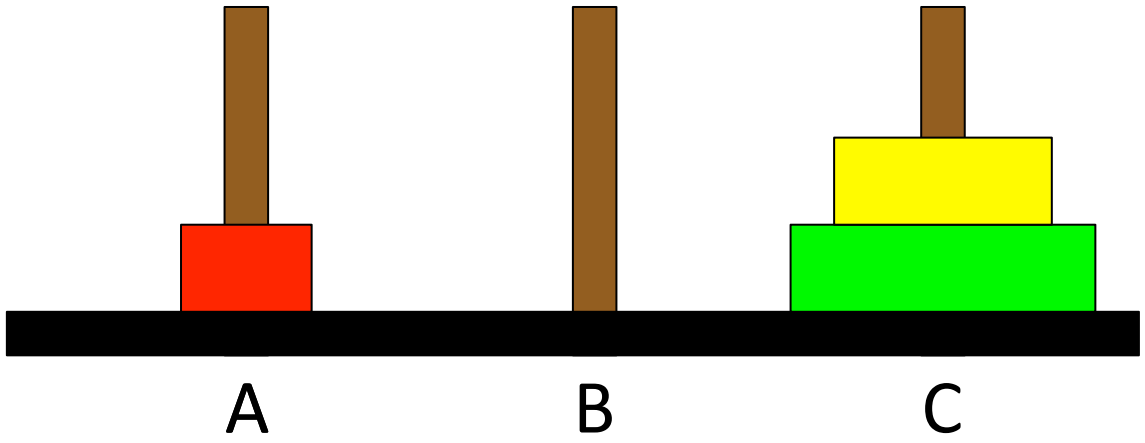


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

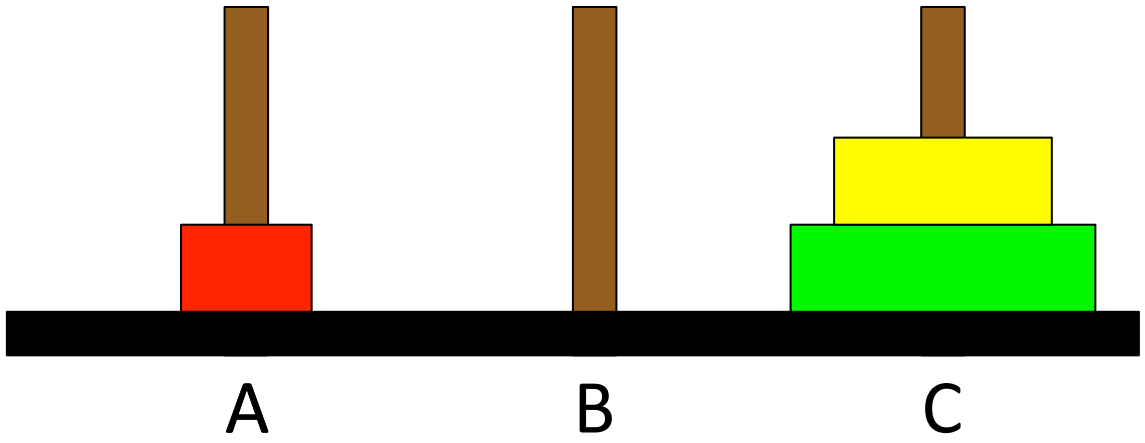


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a



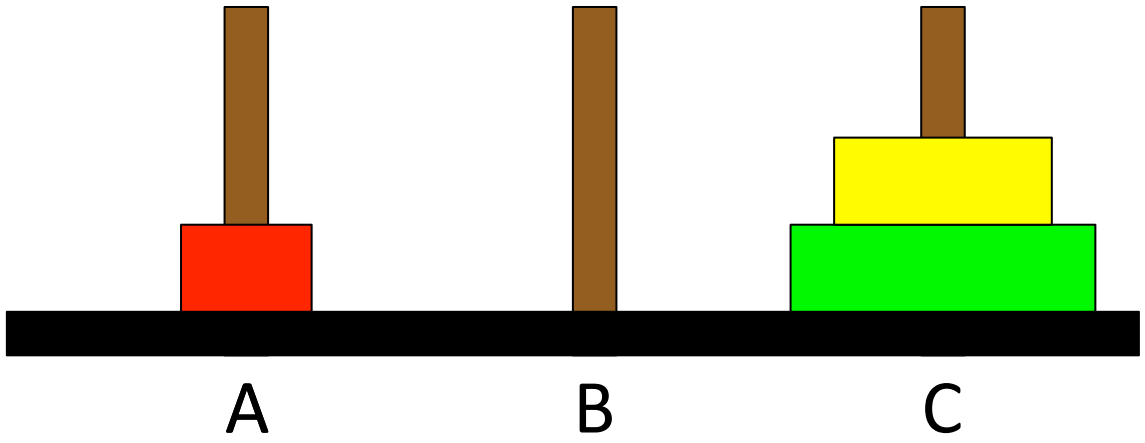
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n 1 from a to c temp b



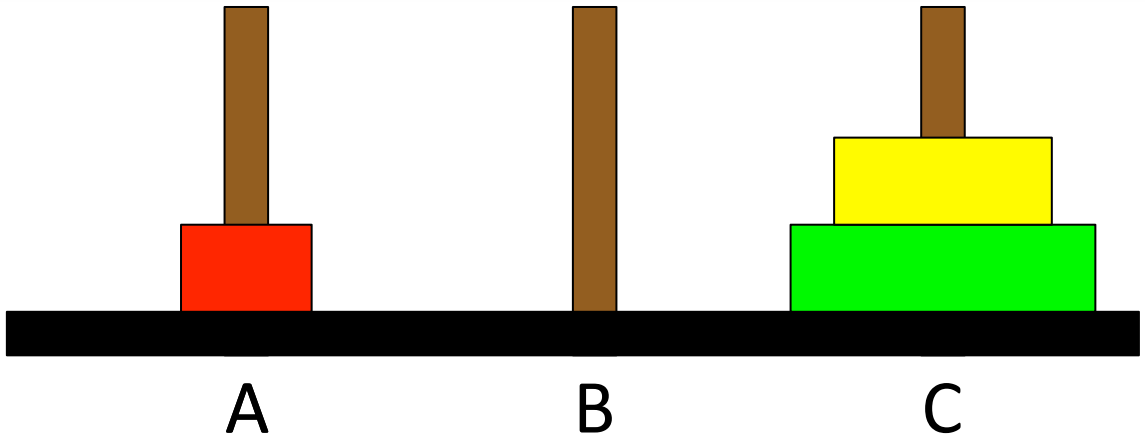
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



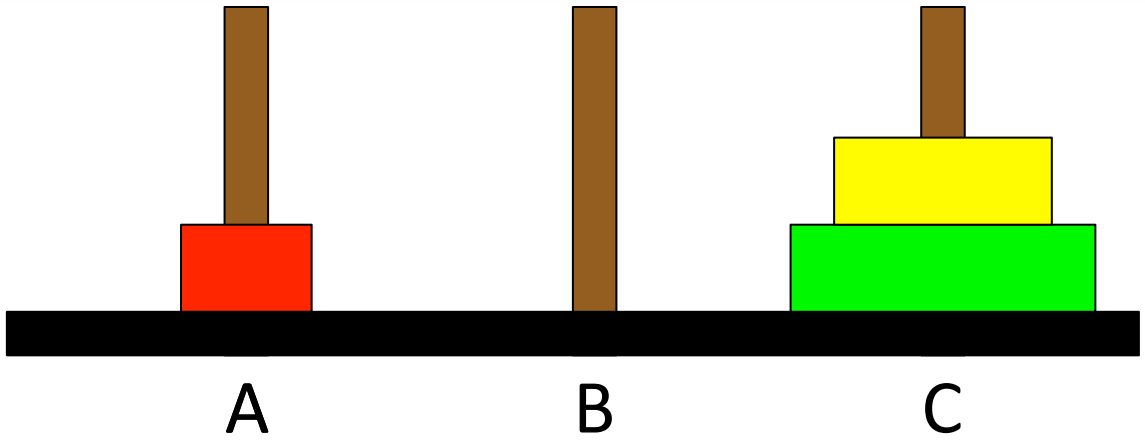
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



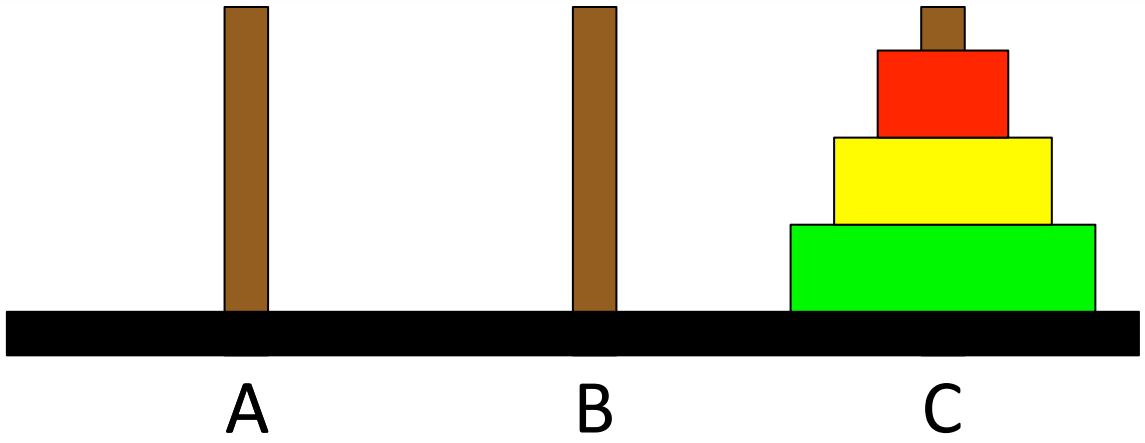
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n 1 from a to c temp b

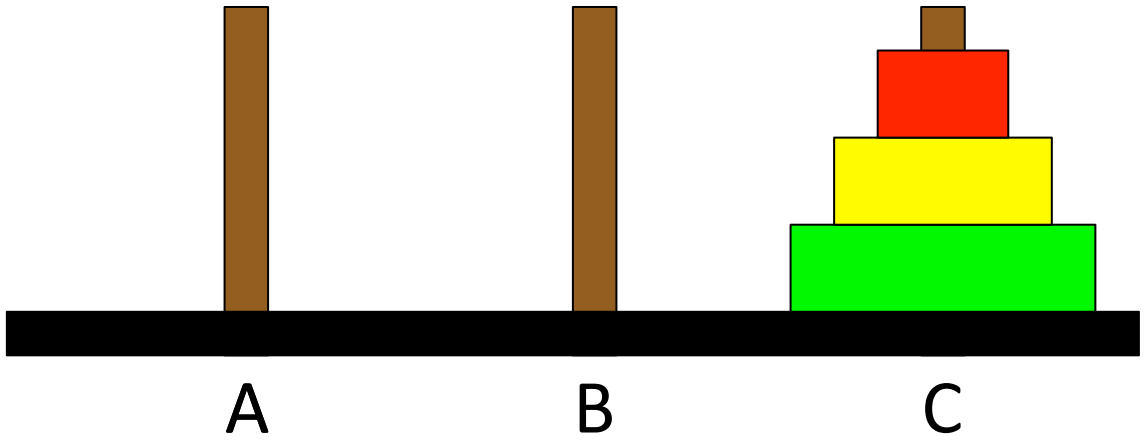


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

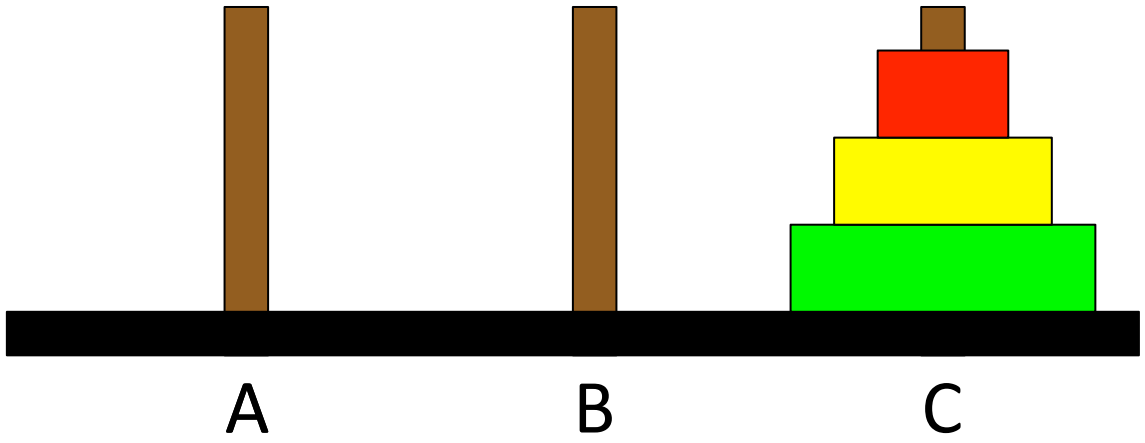
n	2	from	b	to	c	temp	a



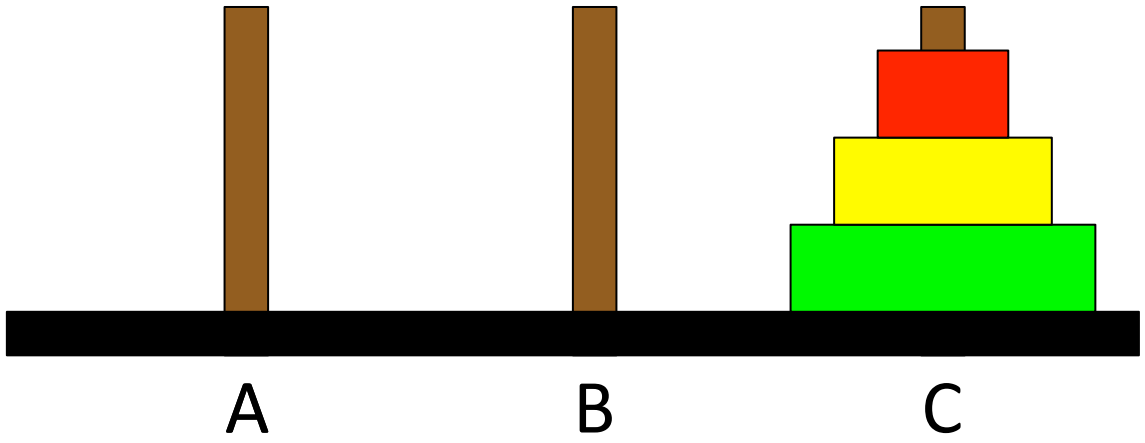
```
int main() {
    moveTower(3, 'a', 'b', 'c');
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	3	from	a	to	b	temp	c



```
int main() {
    moveTower(3, 'a', 'b', 'c');
}
```



Recursive Leap of Faith

The Solution

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

Another Solution

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 0) {
        return;
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

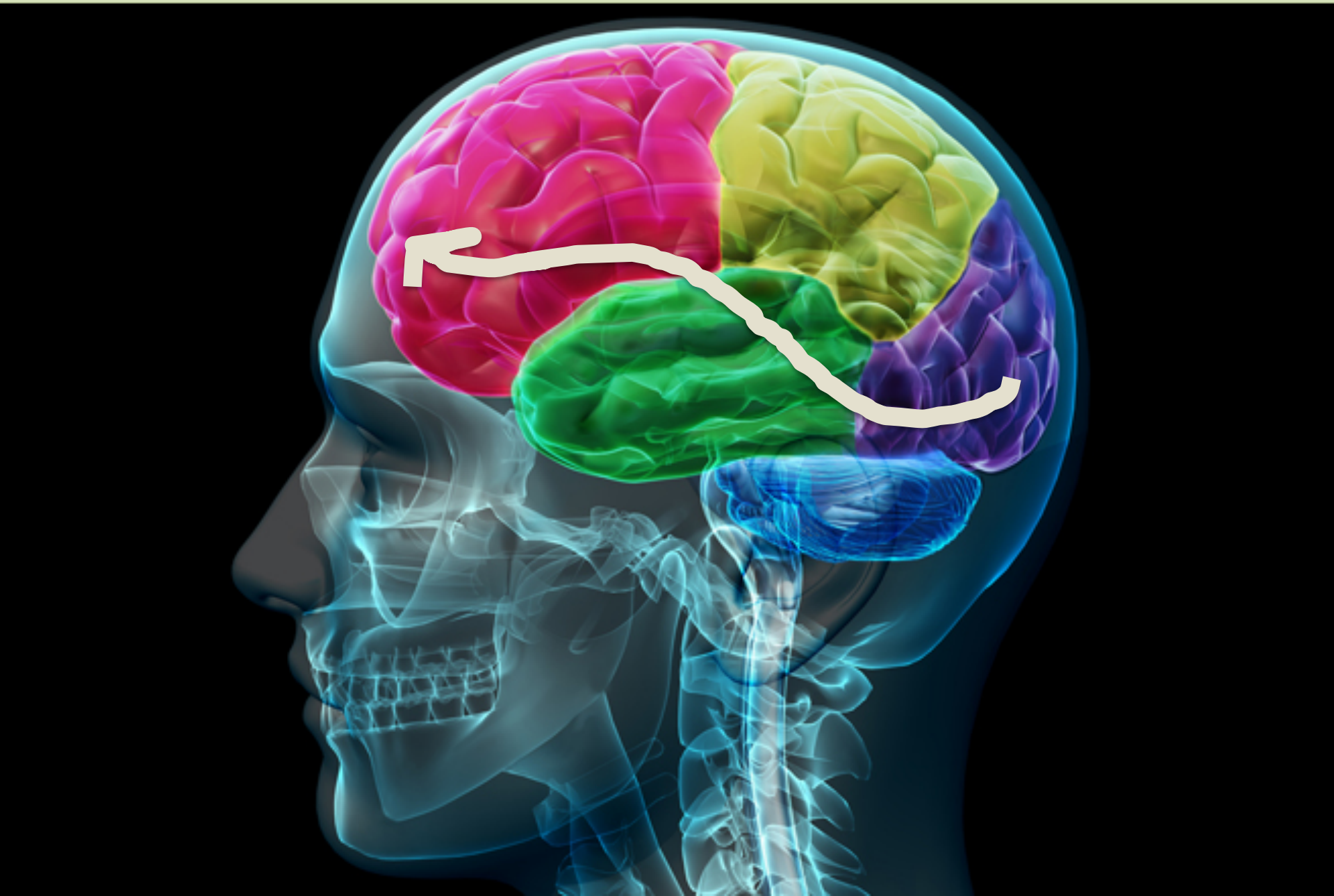
Even Simpler Solution

```
void moveTower(int n, char from, char to, char temp) {
    if (n > 0) {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

You can do recursion.

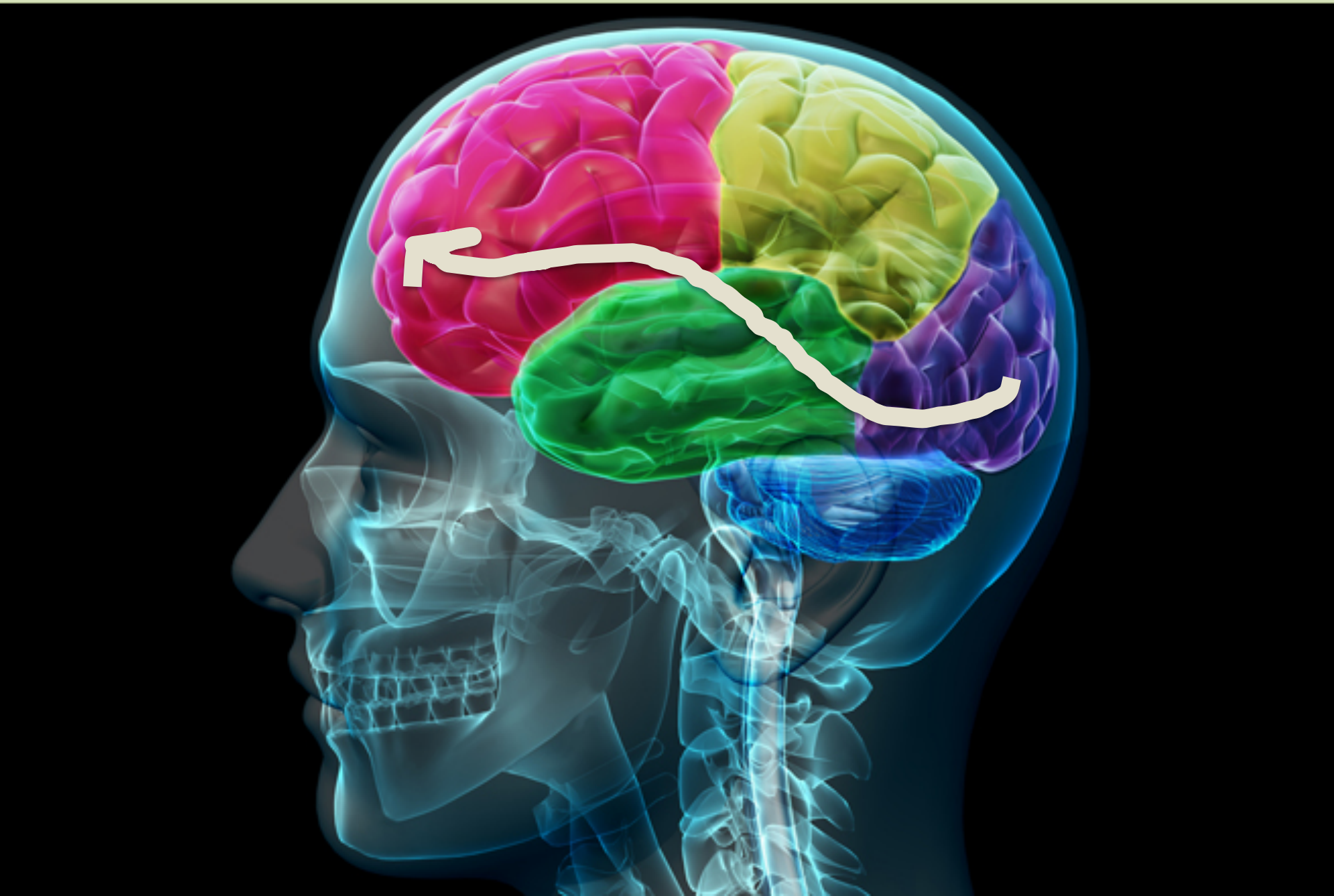
Your Brain is Recursive

There is a pathway in your brain for imagination

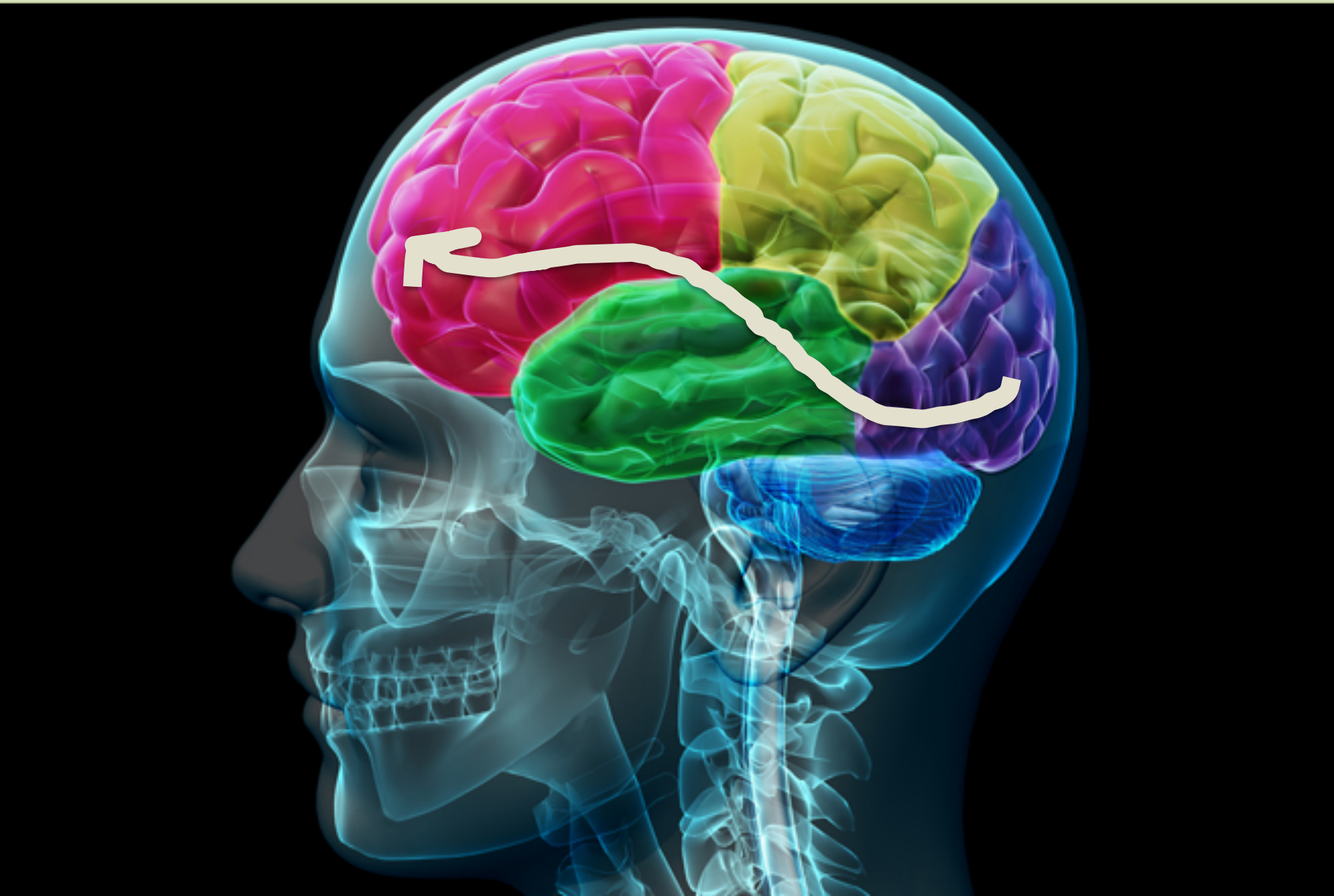


Step 1: Imagine a future at the end of CS106B

Step 1: Imagine a future at the end of CS106B



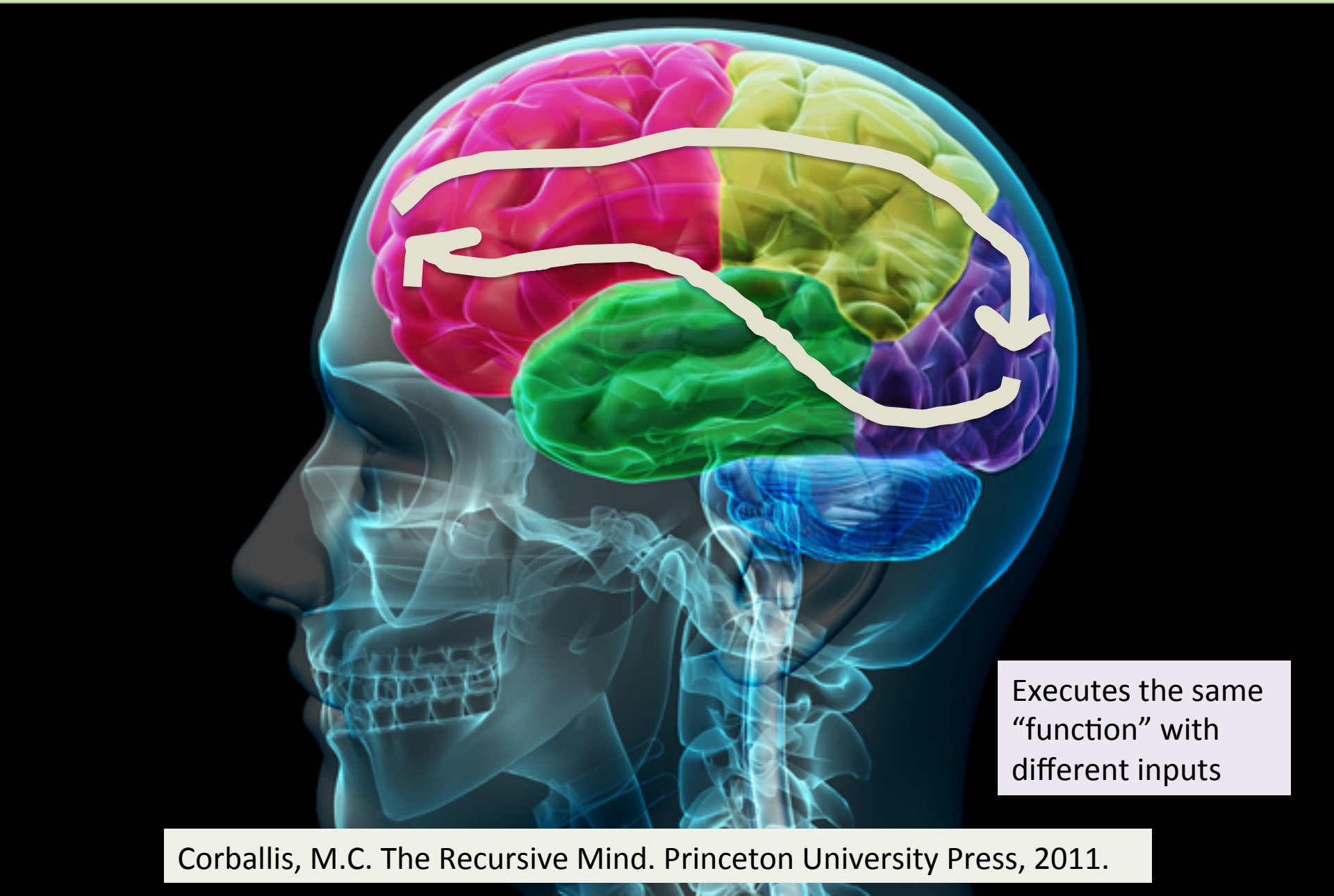
Step 2: Imagine a future after Stanford



Step 2: Imagine a future after Stanford

Executes the same "function" with different inputs

Step 2: Imagine a future after Stanford

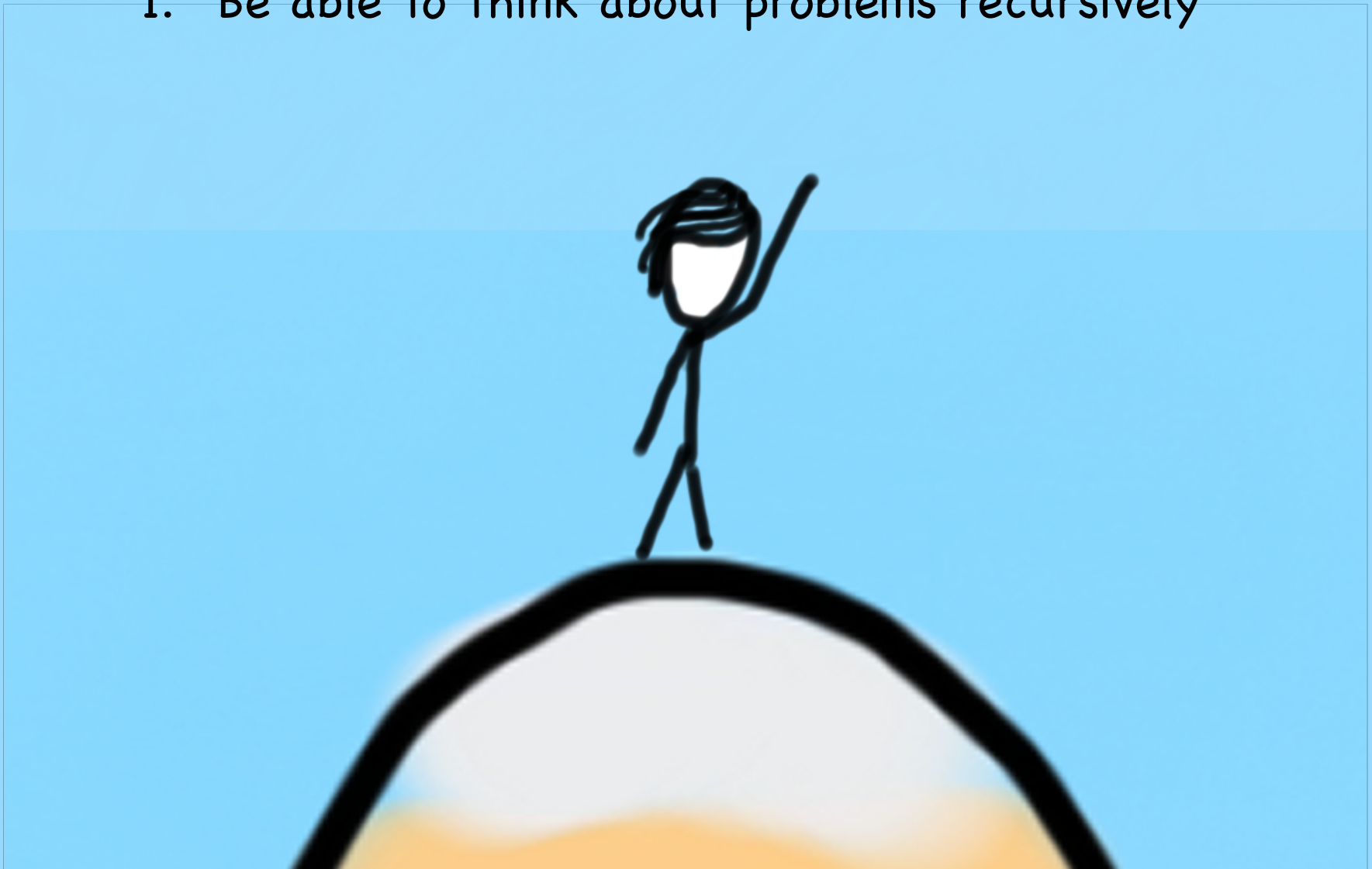


Executes the same "function" with different inputs

Corballis, M.C. The Recursive Mind. Princeton University Press, 2011.

Today's Goal

1. Be able to think about problems recursively



Three Musts of Recursion

1. Your code must have a case for all valid inputs.

2. You must have a base case.

3. When you make a recursive call it should be to a simpler instance (forward progress towards base case)