Complexity, Induction, and Recurrence Relations

CSE 373 Help Session 4/7/2016

Big-O Definition

- Definition:
  - $g(n)$ is in $O(f(n))$ if there exist positive constants c and n_0 such that $g(n) \leq c\, f(n)$ for all $n \geq n_0$
- Upper Bound
  - $T(n) = O(f(n))$ means $T(n)$ grows at a rate no faster than $f(n)$, $f(n)$ is upper bound on $T(n)$.
  - Tightest upper bound of $T(n)$

General Rules for Big-O Analysis

- For loops
  - The running time of a for loop is at most the running time of the statements inside the for loop (including tests) times the number of iterations
- Example:

```
int sum = 0;
for( int i = 0; i < n; i++) {
    sum += i;   // O(1)
}
```

- Runtime? O(1) * n = O(n)

General Rules for Big-O Analysis

- Nested loops
  - Analyze inside out.
  - Running time of the statement multiplied by the product of the sizes of all the loops
- Example:

```
for( int i = 0; i < n; i++) {     // O(1) * n * m = O(m*n)
      for (int j = 0; j < m; j++) {
            sum += i*j;     // O(1)
      }
}
```

General Rules for Big-O Analysis

- Consecutive Statements
  - Just add them up
- Conditional if/else
  - Running time of test plus the larger of the running time between S1 and S2
  - if(condition)
    - S1
  - else
    - S2

Example Problem

Write an algorithm/method *calculatePositiveSum* that takes in a two dimensional integer array arr that has length n and width m, find out the sum of every positive integer in arr. And explain the running time of your method.

For example, if a two dimensional integer array 1, -2, 3 of length 3, width 2 is passed in, -1, 6, -5

The method should return 1 + 3 + 6 = 10

Induction (Slide from Class)

- Type of mathematical proof
- Typically used to establish a given statement for all natural numbers (e.g. integers > 0)
- Proof is a sequence of deductive steps
    1. Show the statement is true for the first number.
    2. Show that if the statement is true for any one number, this implies the statement is true for the next number.
    3. If so, we can infer that the statement is true for all numbers.

Induction (Slide from Class)

- Type of mathematical proof
- Typically used to establish a given statement for all natural numbers (e.g. integers > 0)
- Proof is a sequence of deductive steps
    1. Show the statement is true for the first number.
    2. Show that if the statement is true for any one number, this implies the statement is true for the next number.
    3. If so, we can infer that the statement is true for all numbers.

It's not always obvious how your problem fits into this framework.

Induction

1. Show the statement is true for the first number.
2. Show that if the statement is true for any one number, this implies the statement is true for the next number.
3. If so, we can infer that the statement is true for all numbers.

It's not always obvious how your problem fits into this framework.

Q: What are the "numbers" in my problem?

Q: Which one is first?

Q: What if I can't come up with an equation?

Examples

- Difference of successive squares are the odd numbers.
- Towers of Hanoi and Proofs about Programs.
- A few more if we have time.

A Problem

- Show that the difference between each successive square are the odd numbers.

A Problem

- Show that the difference between each successive square are the odd numbers.

- Equivalent formulation to find the “numbers”
  - Show that the sum of the first k odd numbers is k^2.

A Problem

- Show that the difference between each successive square are the odd numbers.

- Equivalent formulation to find the "numbers"
  - Show that the sum of the first k odd numbers is k^2.

- Make it numerical
  - The kth odd number is:

A Problem

- Show that the difference between each successive square are the odd numbers.

- Equivalent formulation to find the “numbers”
  - Show that the sum of the first k odd numbers is k^2.

- Make it numerical
  - The kth odd number is (2k-1).
    - (2(1)-1) = 1, (2(2)-1) = 3, (2(3)-1) = 5, …

As Equations

- Prove by induction on k that

As Equations

- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$

As Equations

- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Base Case

As Equations

- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Base Case
  - $\sum_{n=1}^{1} 2n - 1 = 2(1) - 1 = 1 = 1^2$.

As Equations

- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Base Case
  - $\sum_{n=1}^{1} 2n - 1 = 2(1) - 1 \ = 1 = 1^2$.
- Inductive Hypothesis

As Equations

- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Base Case
  - $\sum_{n=1}^{1} 2n - 1 = 2(1) - 1 \ = 1 = 1^2.$
- Inductive Hypothesis
  - $\sum_{n=1}^{k} 2n - 1 = k^2$

As Equations

- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Base Case
  - $\sum_{n=1}^{1} 2n - 1 = 2(1) - 1 \ = 1 = 1^2$.
- Inductive Hypothesis
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Inductive Step

As Equations

- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Base Case
  - $\sum_{n=1}^{1} 2n - 1 = 2(1) - 1 = 1 = 1^2$.
- Inductive Hypothesis
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Inductive Step
  - $\sum_{n=1}^{k+1} 2n - 1$

As Equations

- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Base Case
  - $\sum_{n=1}^{1} 2n - 1 = 2(1) - 1 \ = 1 = 1^2$.
- Inductive Hypothesis
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Inductive Step
  - $\sum_{n=1}^{k+1} 2n - 1 = 2(k + 1) - 1 + \sum_{n=1}^{k}(2n - 1)$

As Equations

- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Base Case
  - $\sum_{n=1}^{1} 2n - 1 = 2(1) - 1 \ = 1 = 1^2.$
- Inductive Hypothesis
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Inductive Step
  - $\sum_{n=1}^{k+1} 2n - 1 = 2(k+1) - 1 + \sum_{n=1}^{k}(2n - 1) = 2(k+1) - 1 + k^2$

As Equations

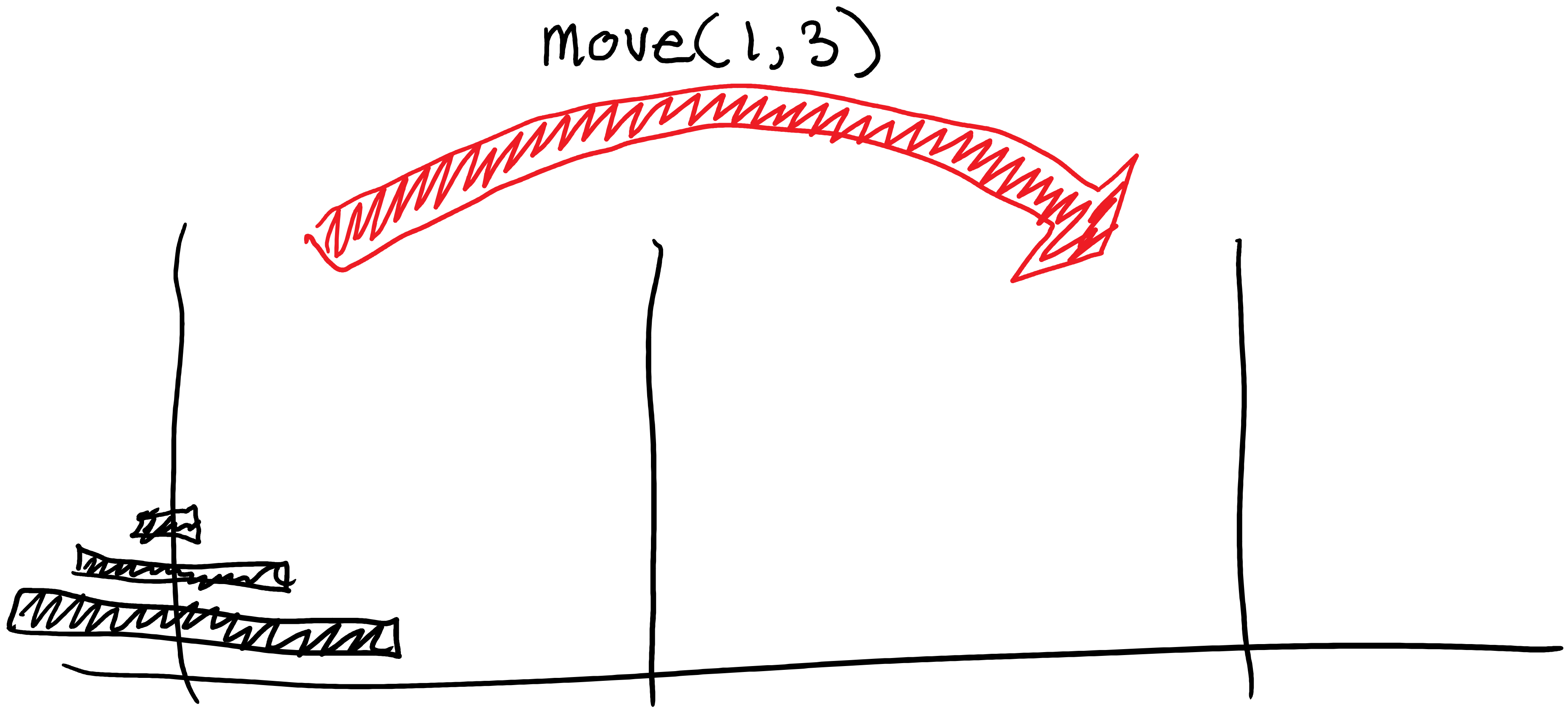
- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Base Case
  - $\sum_{n=1}^{1} 2n - 1 = 2(1) - 1 = 1 = 1^2$.
- Inductive Hypothesis
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Inductive Step
  - $\sum_{n=1}^{k+1} 2n - 1 = 2(k+1) - 1 + \sum_{n=1}^{k}(2n - 1) = 2(k+1) - 1 + k^2$
    $= k^2 + 1 + 2k$

As Equations

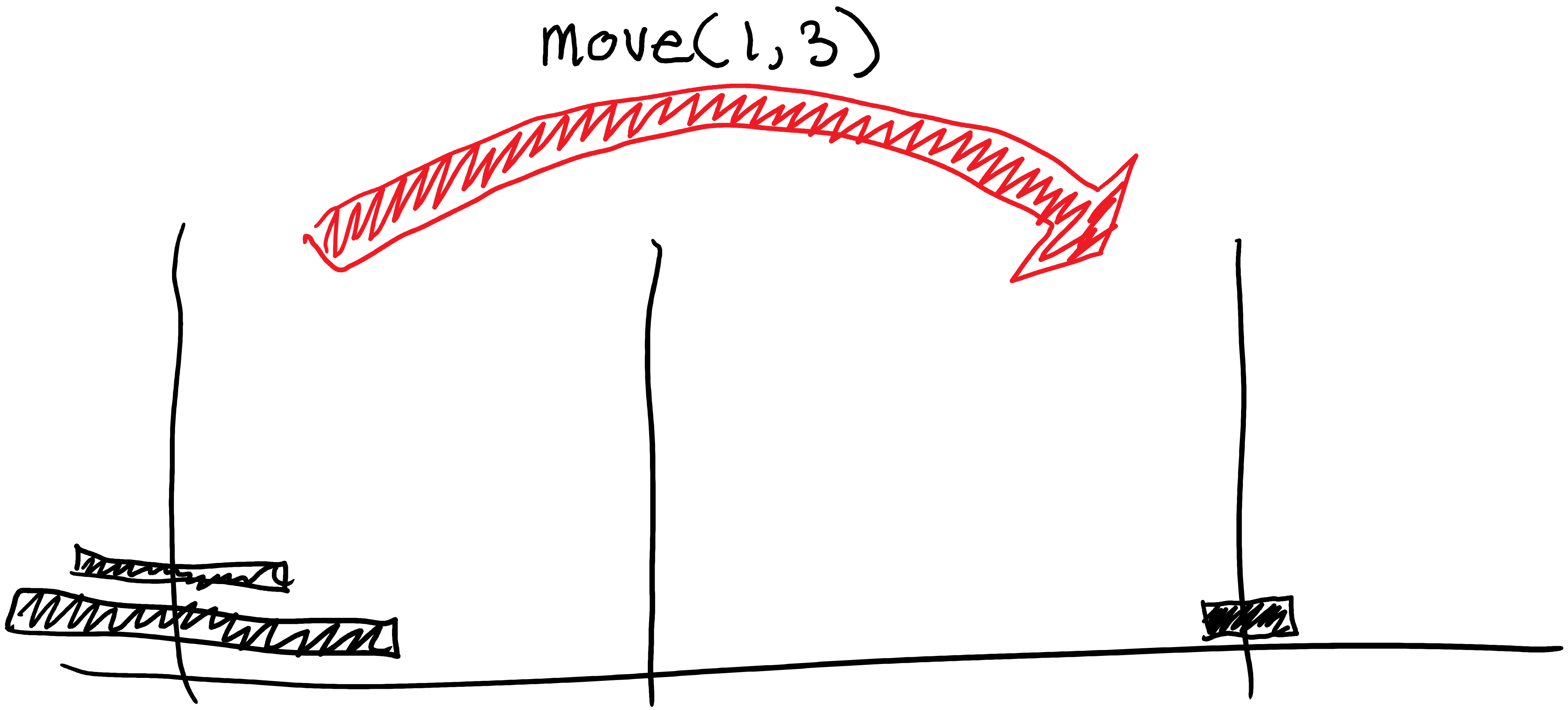
- Prove by induction on k that
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Base Case
  - $\sum_{n=1}^{1} 2n - 1 = 2(1) - 1 \ = 1 = 1^2.$
- Inductive Hypothesis
  - $\sum_{n=1}^{k} 2n - 1 = k^2$
- Inductive Step
  - $\sum_{n=1}^{k+1} 2n - 1 = 2(k+1) - 1 + \sum_{n=1}^{k}(2n-1) = 2(k+1) - 1 + k^2$
    $= k^2 + 1 + 2k = (k+1)^2.$

Towers of Hanoi

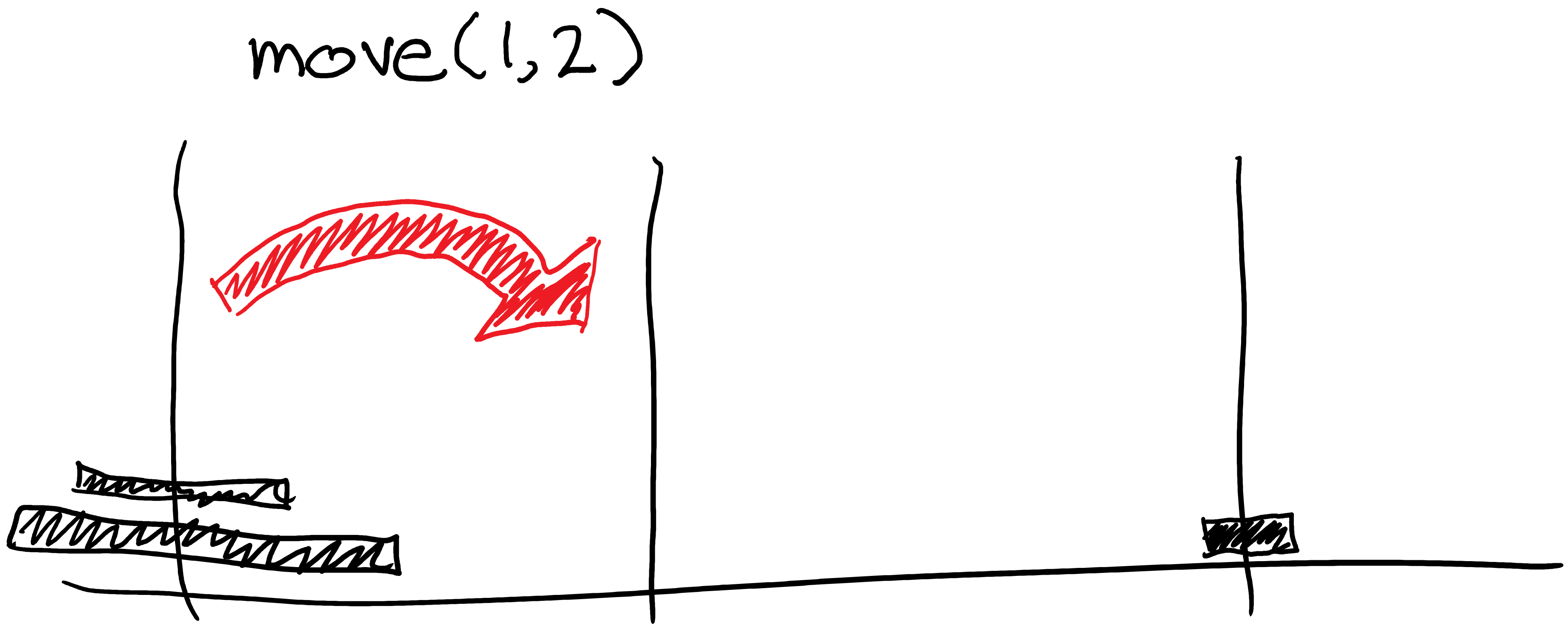
Towers of Hanoi



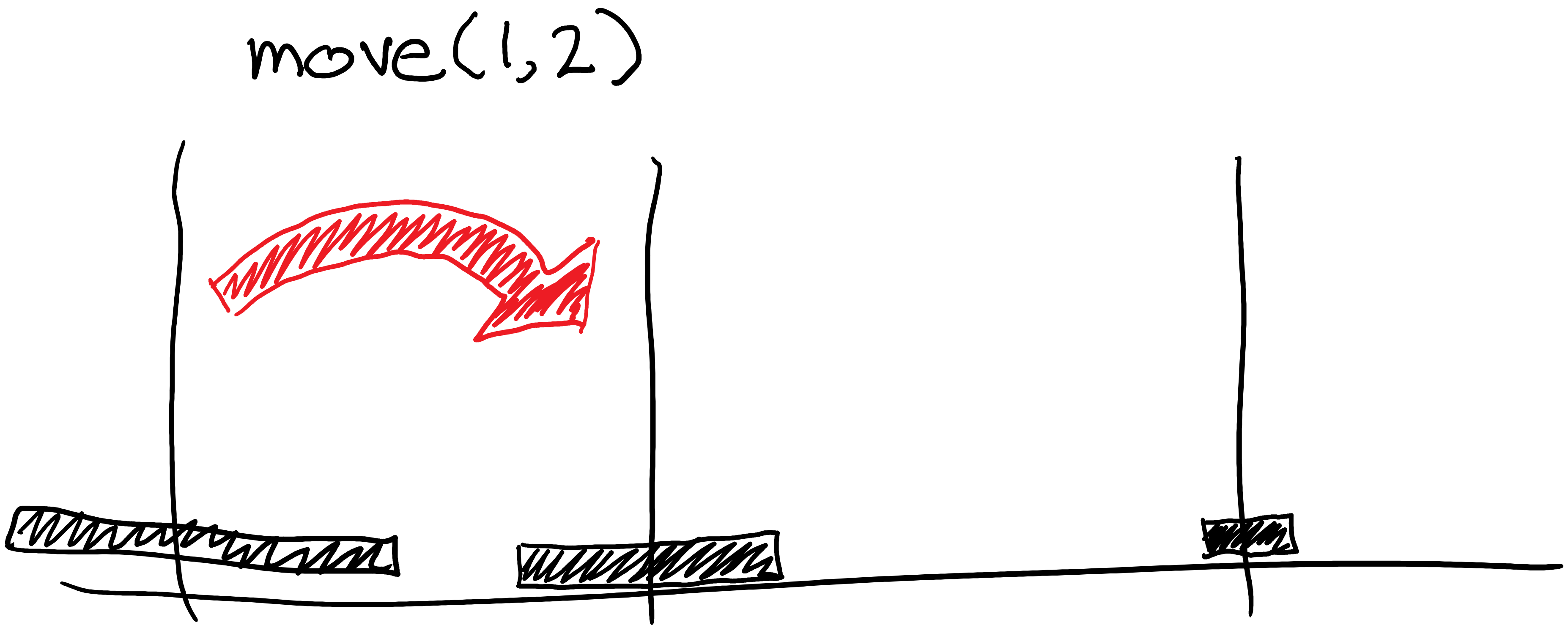
Towers of Hanoi



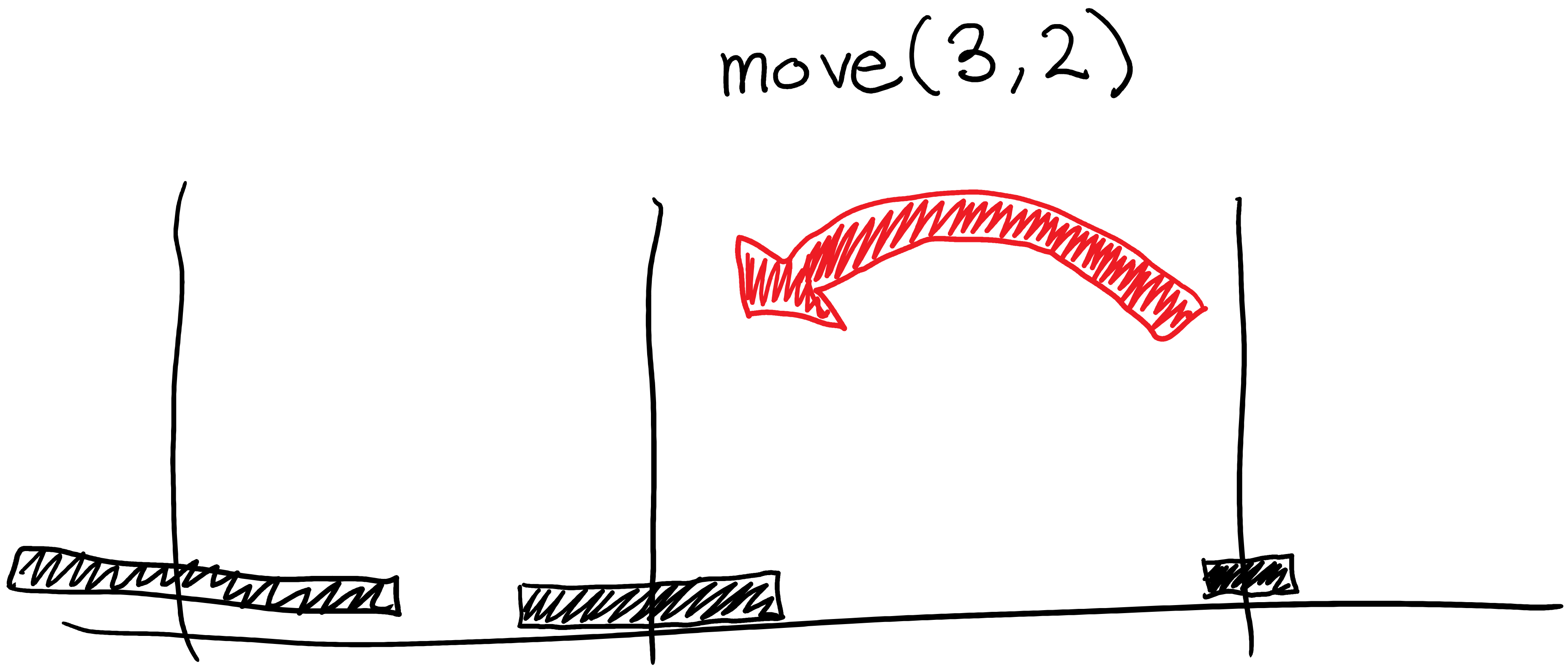
Towers of Hanoi



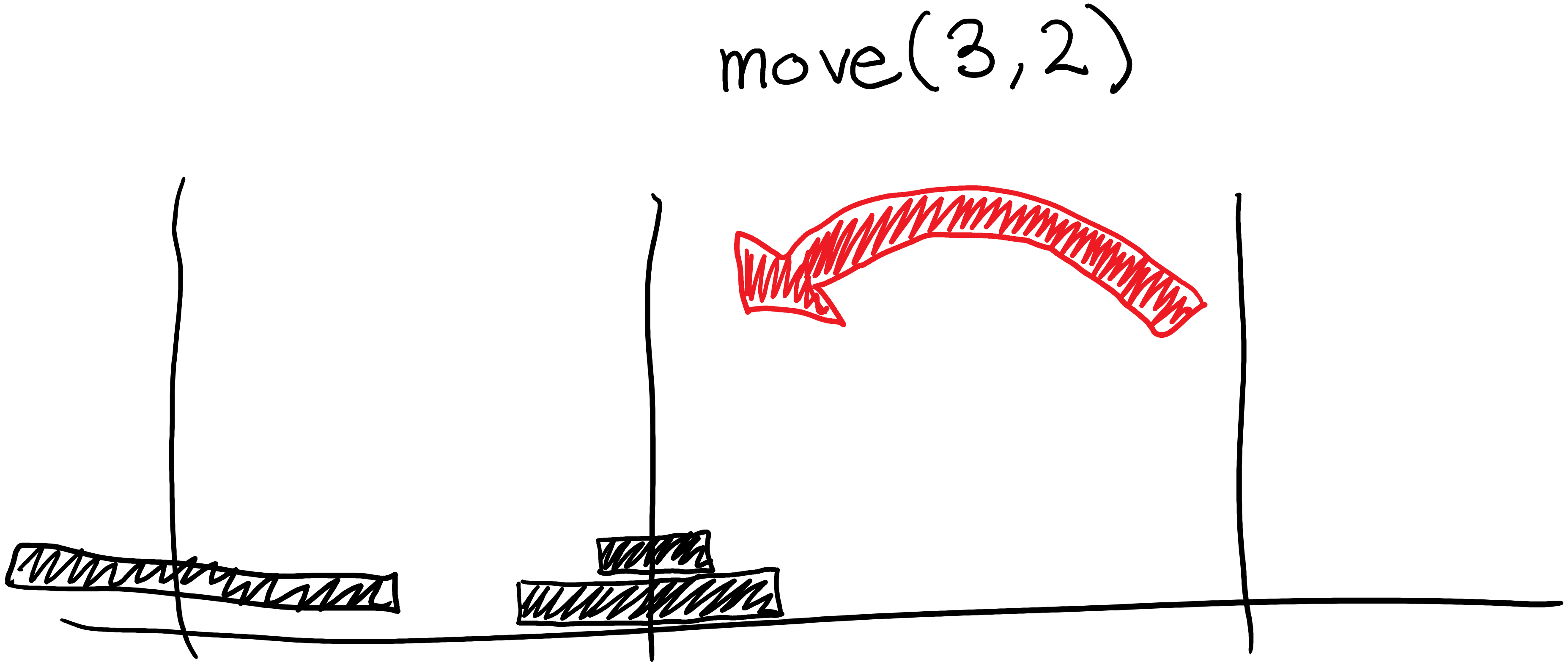
Towers of Hanoi



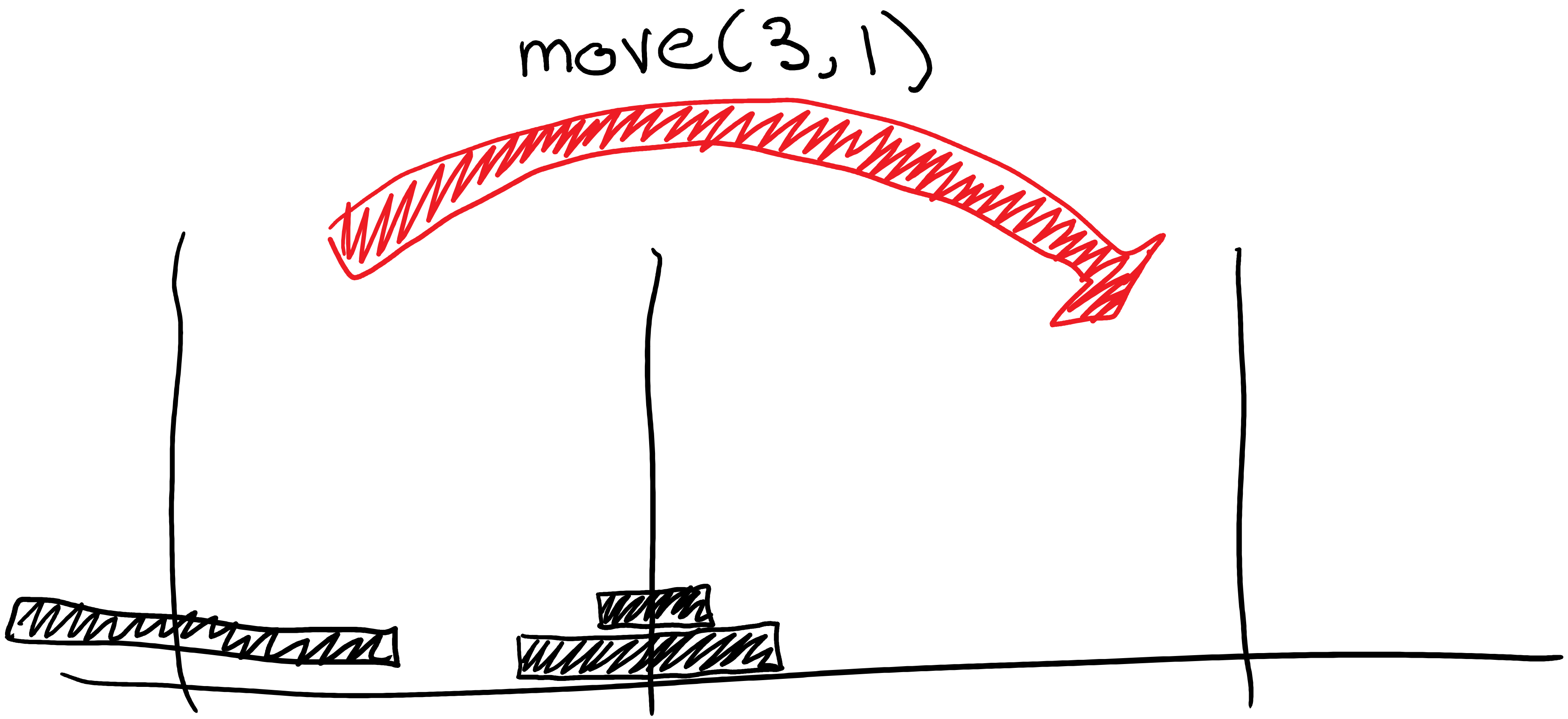
Towers of Hanoi



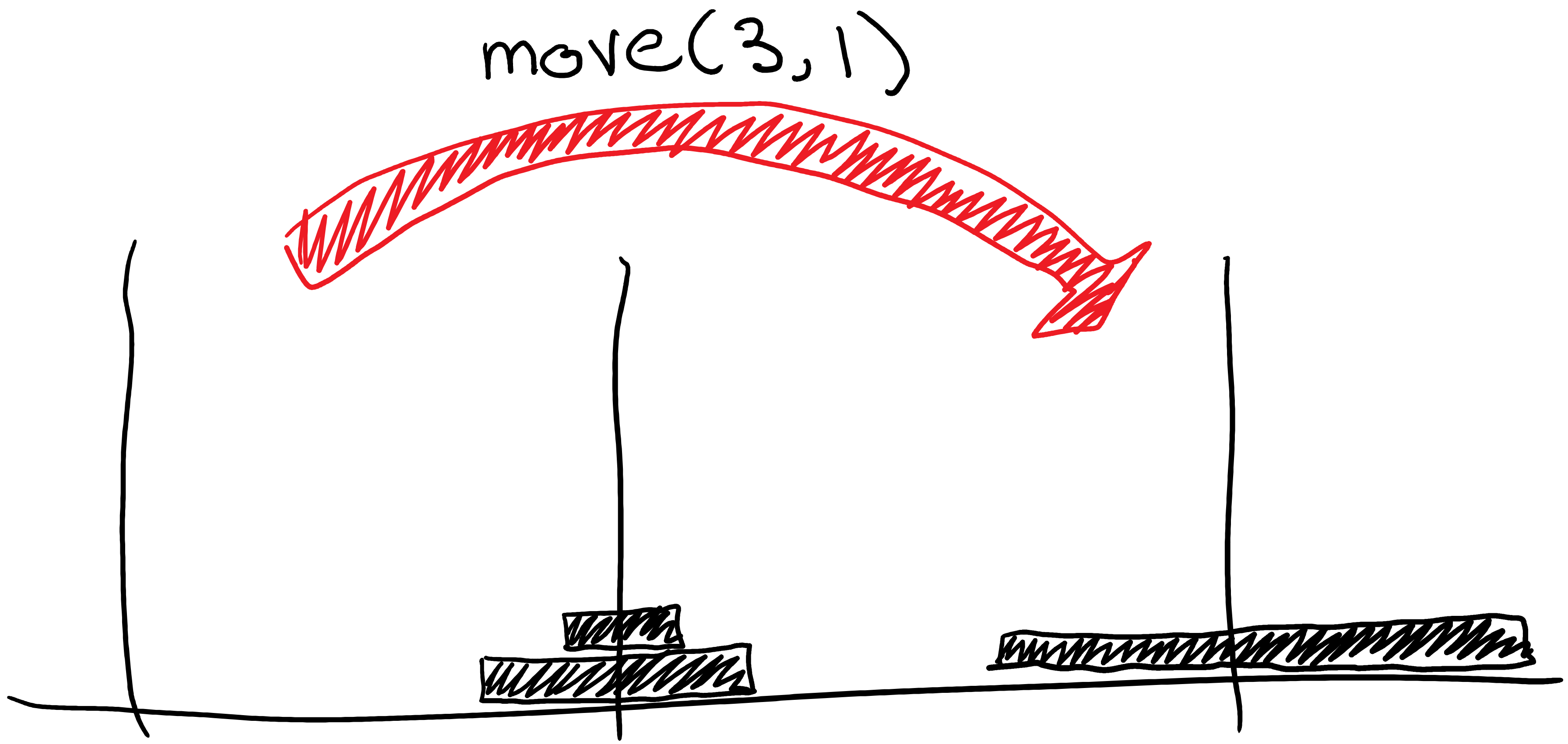
Towers of Hanoi



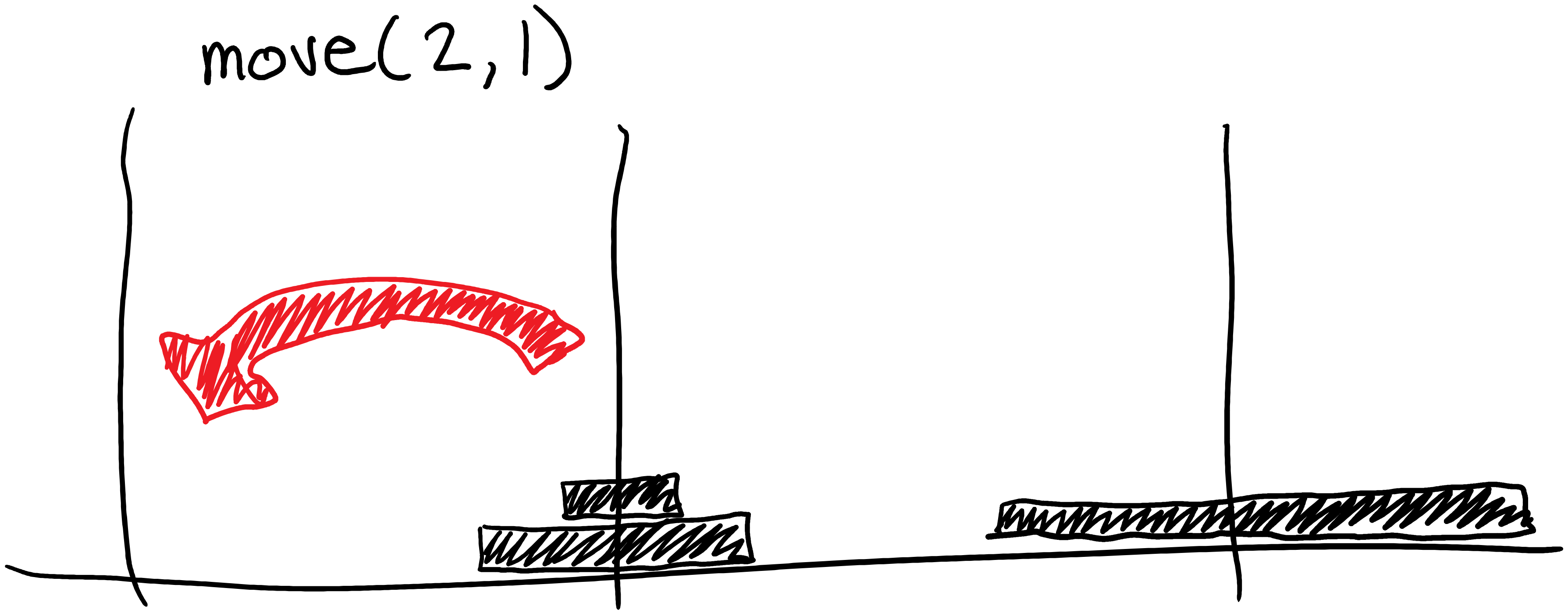
Towers of Hanoi



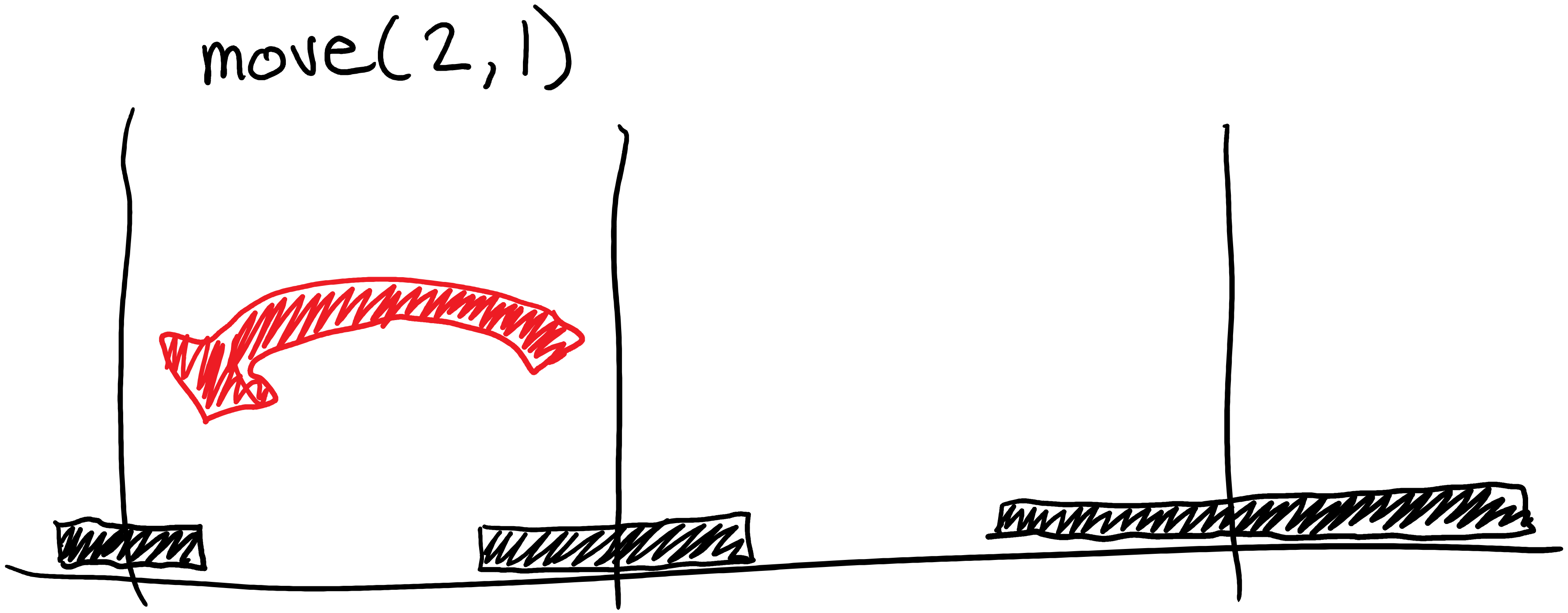
Towers of Hanoi



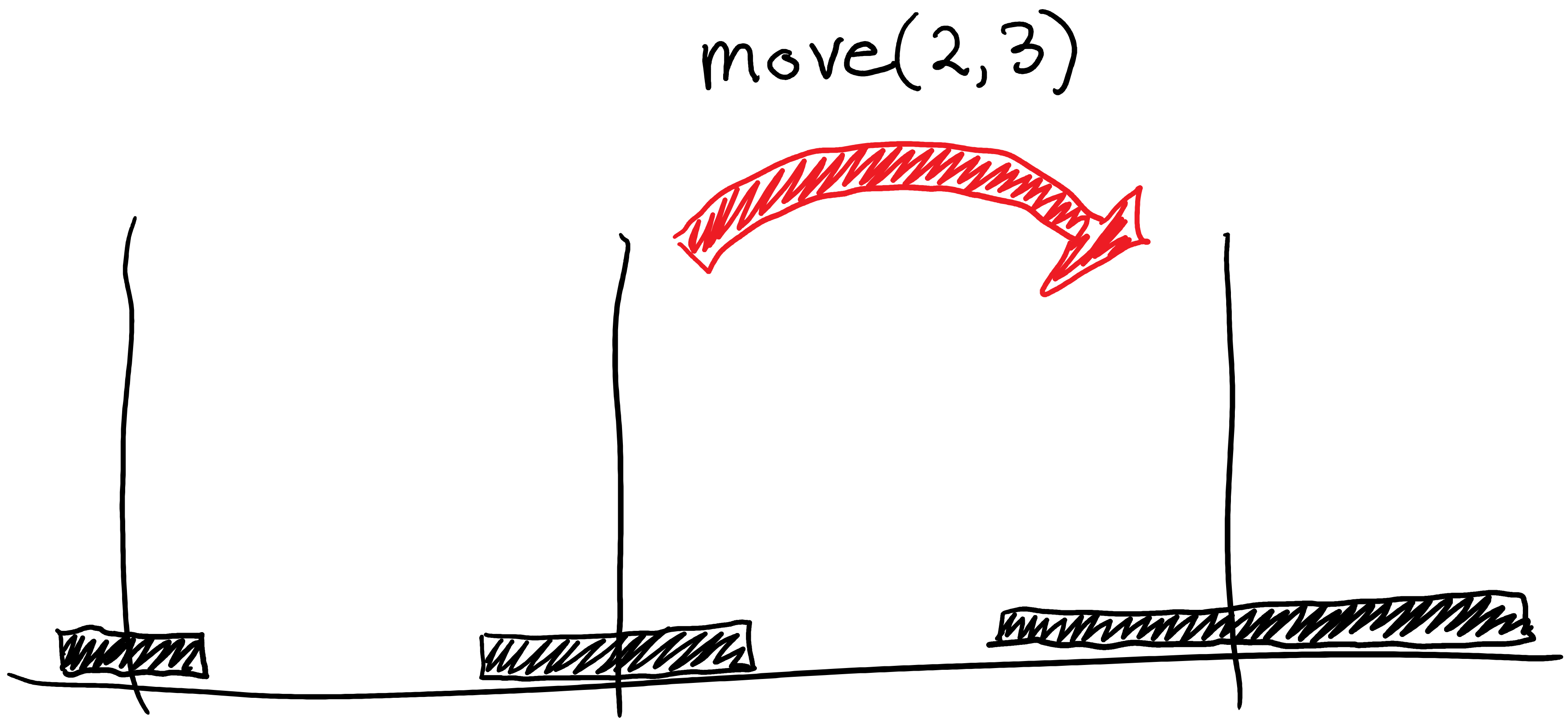
Towers of Hanoi



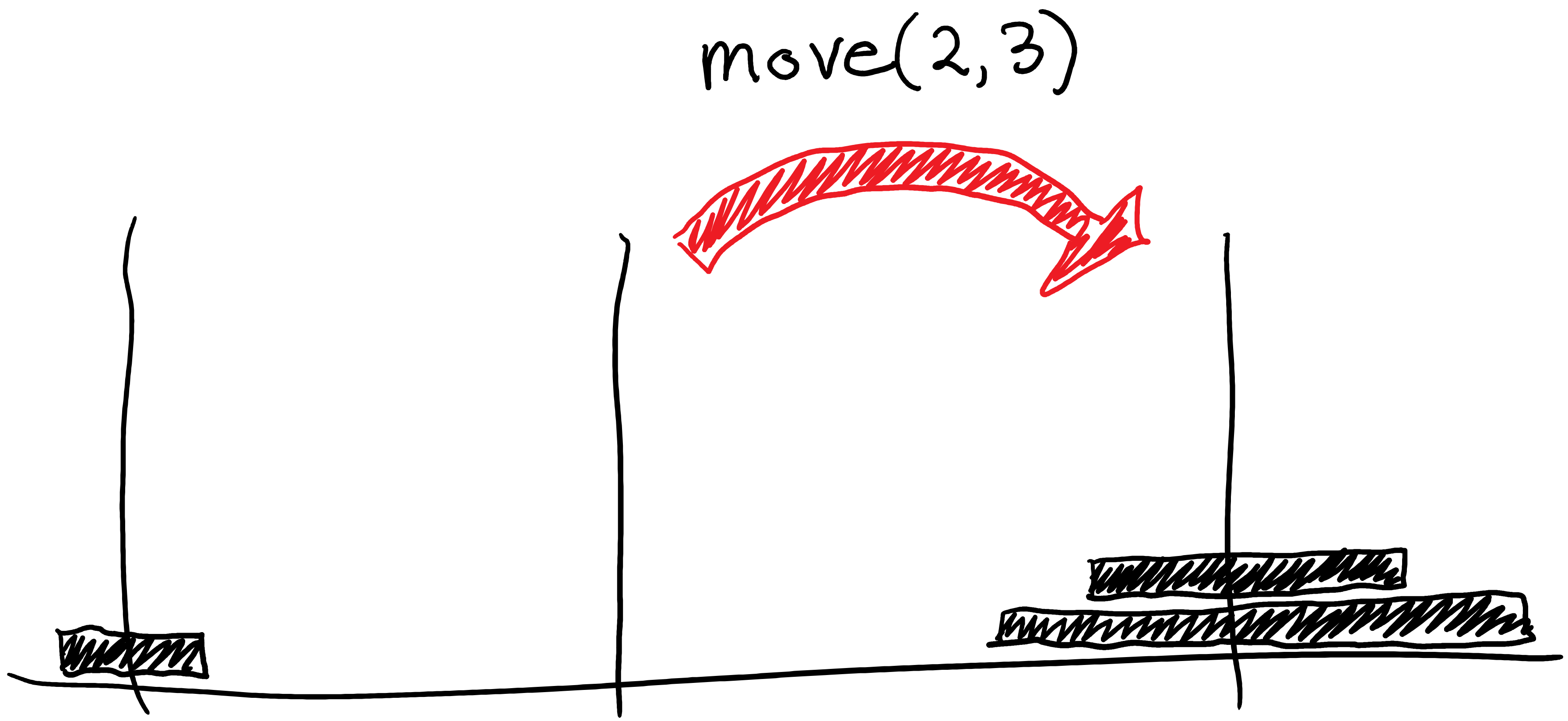
Towers of Hanoi



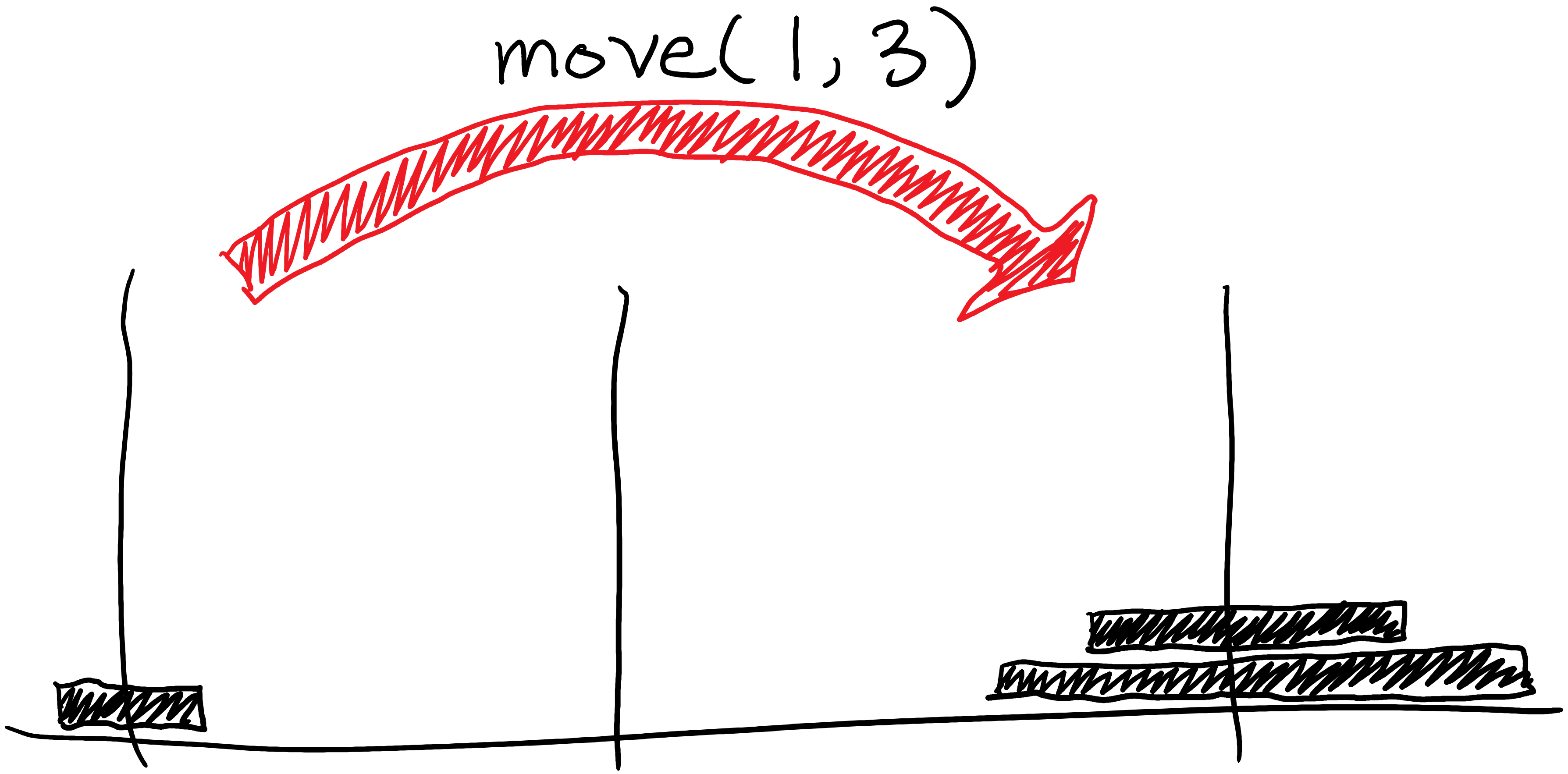
Towers of Hanoi



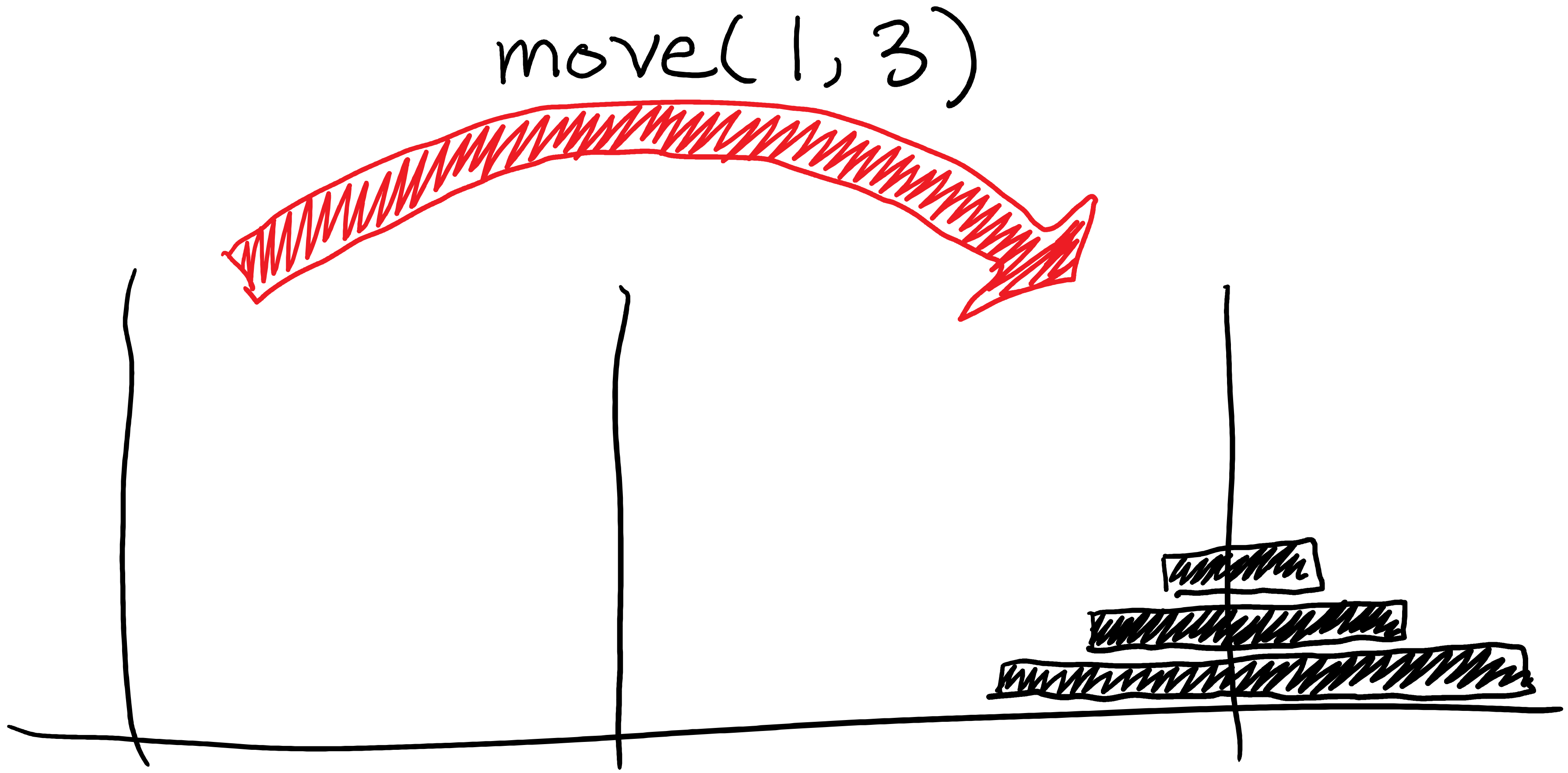
Towers of Hanoi



Towers of Hanoi



Towers of Hanoi



Towers of Hanoi

1: move(1,3)
2: move(1,2)
3: move(3,2)
4: move(1,3)
5: move(2,1)
6: move(2,3)
7: move(1,3)

Solving a tower of height 3 took 7 moves.

Questions

- How do we solve any tower?
- How do we know the solution is correct?
- How many steps does it take to solve a tower of height k?

Questions

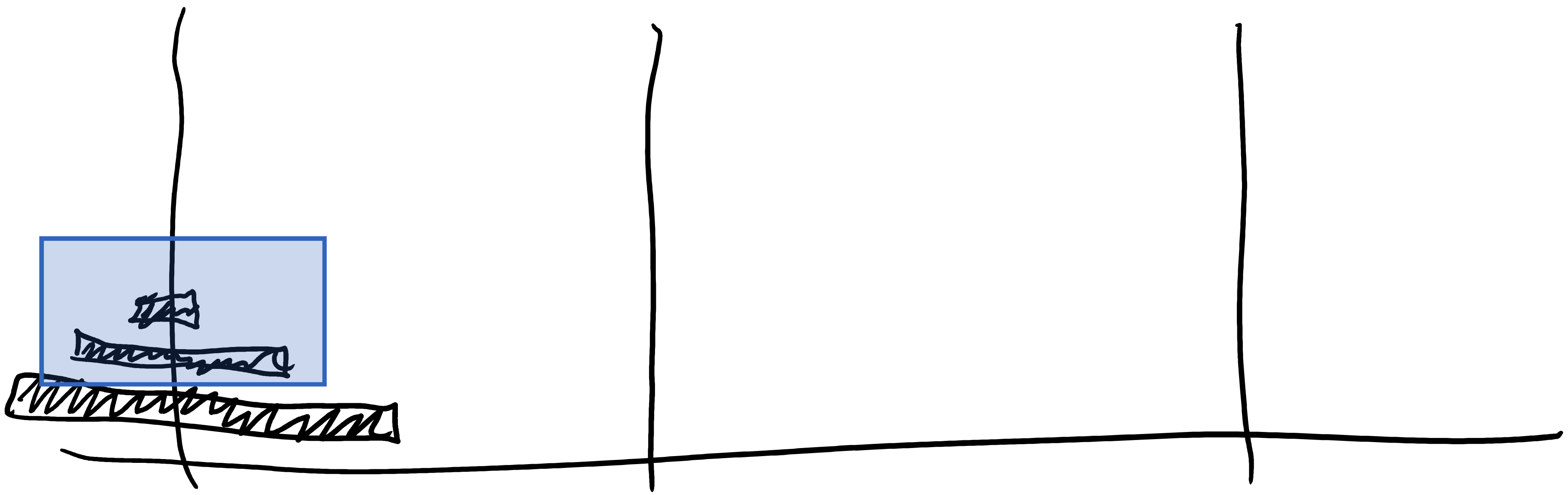
- How do we solve any tower?
  - Recursion!
- How do we know the solution is correct?
  - Induction!
- How many steps does it take to solve a tower of height k?
  - Recurrence relation! (And induction!)

Solving Hanoi

Key Insight: If we can move a tower of size k-1 to the “other” peg, we can move the whole tower.

Solving Hanoi

Key Insight: If we can move a tower of size k-1 to the “other” peg, we can move the whole tower.

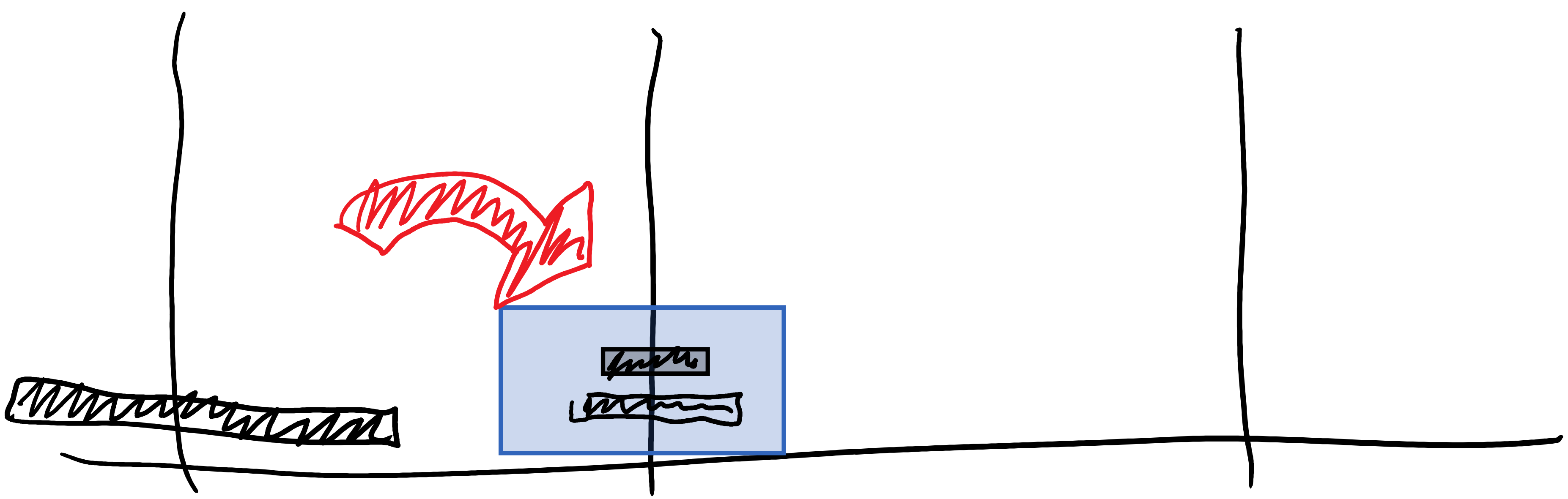


Solving Hanoi

Key Insight: If we can move a tower of size k-1 to the "other" peg, we can move the whole tower.

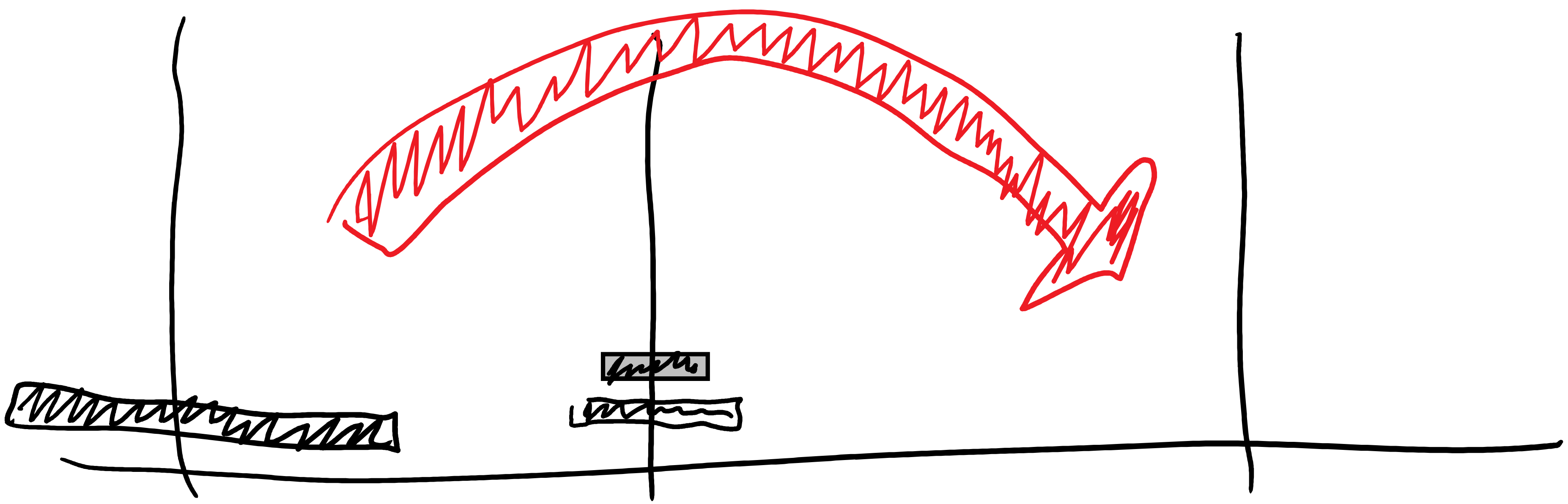
Solving Hanoi

Key Insight: If we can move a tower of size k-1 to the "other" peg, we can move the whole tower.



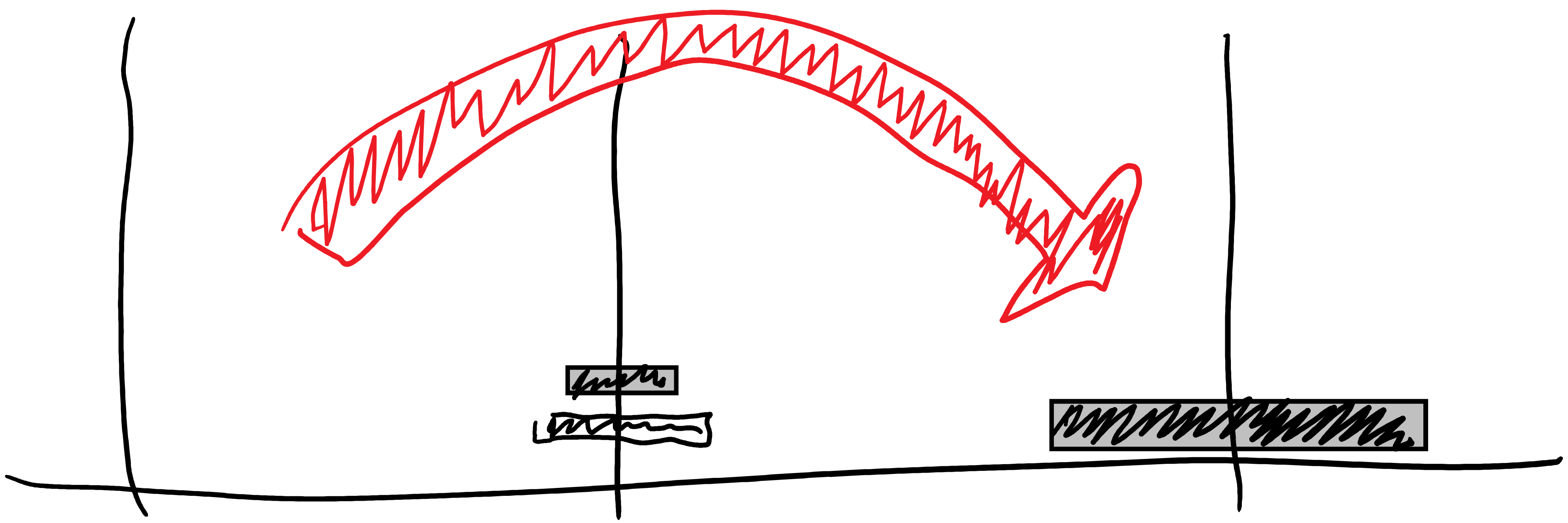
Solving Hanoi

Key Insight: If we can move a tower of size k-1 to the "other" peg, we can move the whole tower.



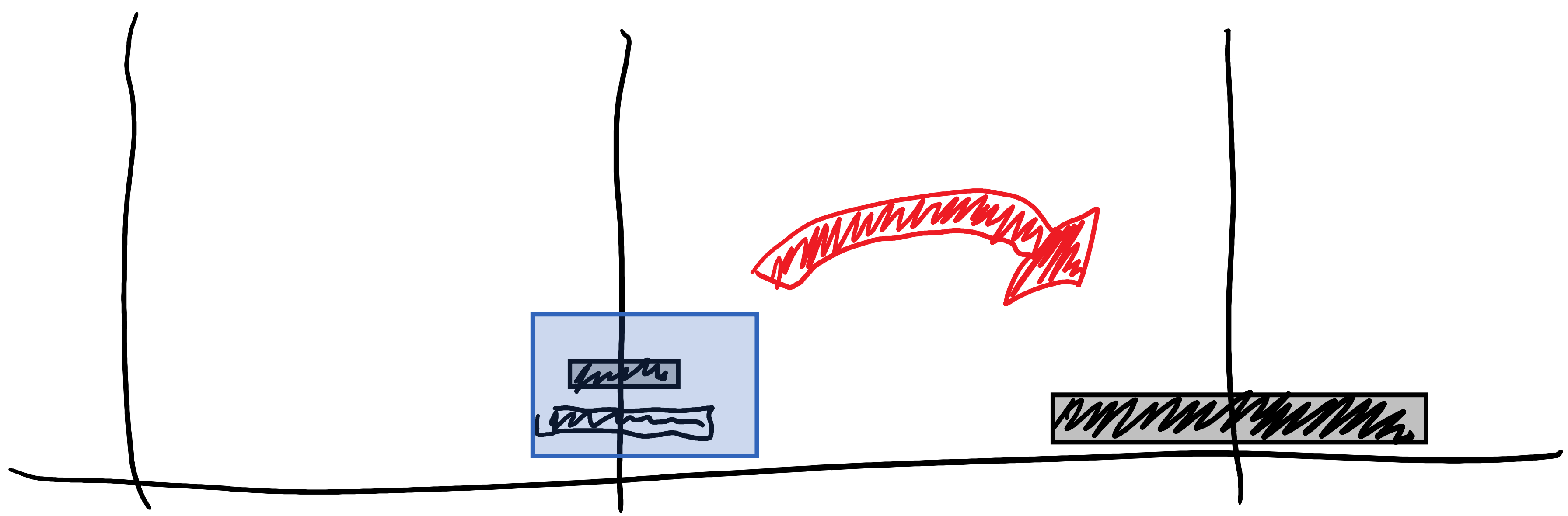
Solving Hanoi

Key Insight: If we can move a tower of size k-1 to the “other” peg, we can move the whole tower.



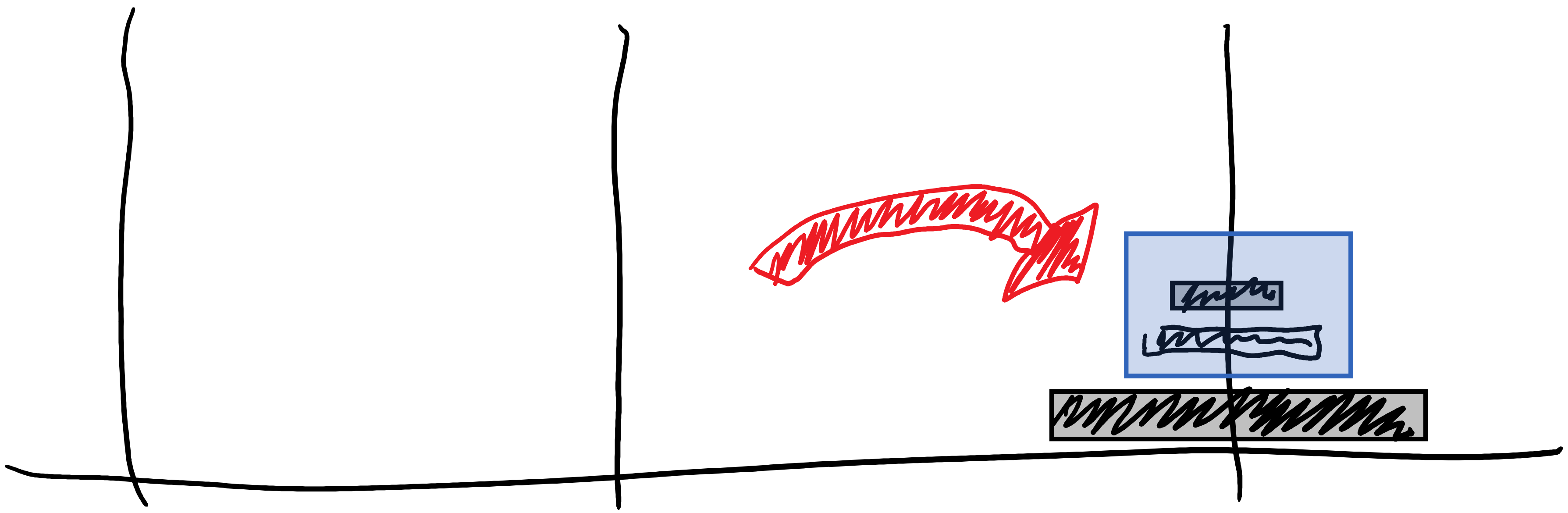
Solving Hanoi

Key Insight: If we can move a tower of size k-1 to the "other" peg, we can move the whole tower.



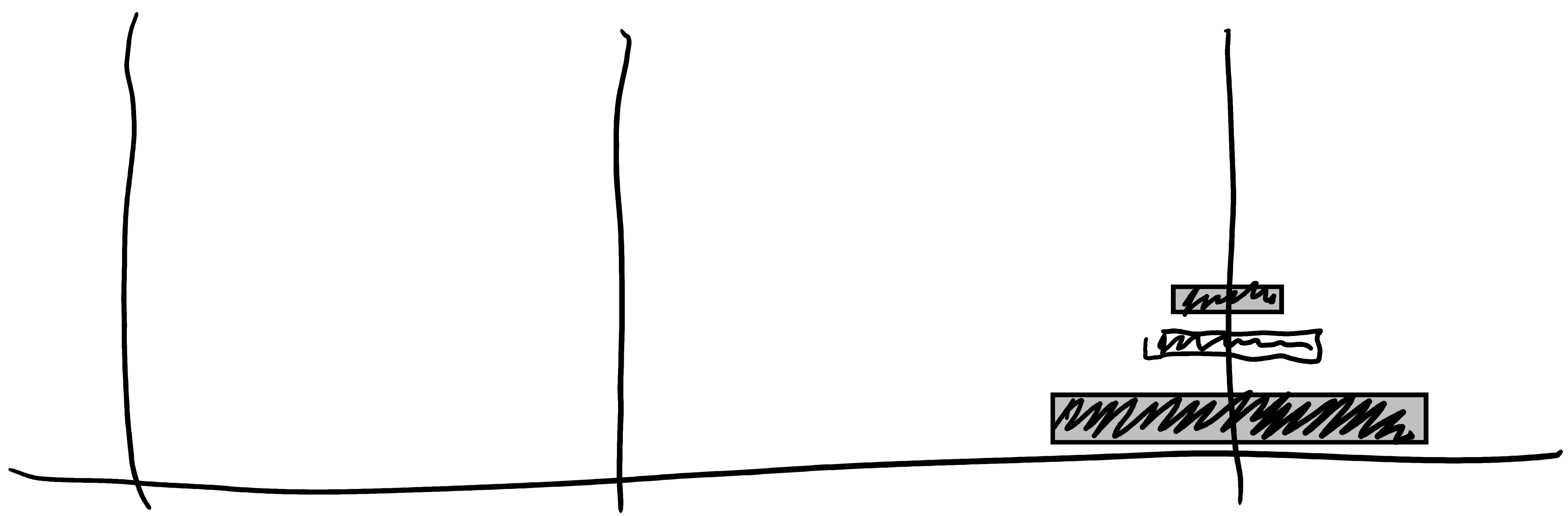
Solving Hanoi

Key Insight: If we can move a tower of size k-1 to the “other” peg, we can move the whole tower.



Solving Hanoi

Key Insight: If we can move a tower of size k-1 to the "other" peg, we can move the whole tower.



Finding the Induction

To figure out how to prove a recursion is correct, think about writing it yourself.

Recursive Subroutine ⇔ Inductive Hypothesis

Base Case ⇔ Base Case

Using the recursive subroutine ⇔ Inductive Step

Step 1: Assume it already works

hanoi_recursive(k, A, B)

Move a tower of size k from peg A to peg B.

Step 2: Build the function using itself

```
hanoi_recursive( k, A, B ) {
        if ( BASE CASE ) {
                // Base Case Solution!
        } else {
                hanoi_recursive(k-1, A, other(A,B));
                move(A, B);
                hanoi_recursive(k-1, other(A,B), B);
        }
}
```

Step 3: Fill in the Base Case

```
hanoi_recursive( k, A, B ) {
    if ( k == 1) {
        move(A, B);
    } else {
        hanoi_recursive(k-1, A, other(A,B));
        move(A, B);
        hanoi_recursive(k-1, other(A,B), B);
    }
}
```

Step 2: Fill in the Base Case

```
hanoi_recursive( k, A, B ) {
	if ( k == 0 ) {
		// Nothing to do!
	} else {
		hanoi_recursive(k-1, A, other(A,B));
		move(A, B);
		hanoi_recursive(k-1, other(A,B), B);
	}
}
```

Cleaned Up

```
hanoi_recursive( k, A, B ) {
        if ( k > 0 ) {
                hanoi_recursive(k-1, A, other(A,B));
                move(A, B);
                hanoi_recursive(k-1, other(A,B), B);
        }
}
hanoi(k) { hanoi_recursive(k, 1, 3); }
```

Inductive Proof

Inductive Hypothesis:

hanoi_recursive(k-1, A, B) moves a correct tower of height k from peg A to peg B if no disks on pegs other than A are smaller than k-1. After executing, no pegs other than B have disks smaller than k-1

Base Case:

hanoi_recursive(0, A, B) does nothing.

Inductive Step:

Suppose hanoi_recursive(k-1, A, B) works. Then step through execution of hanoi_recursive(k, A, B)

Proving the Base Case

```
1: hanoi_recursive( k, A, B ) {
2:     if ( k > 0 ) {
3:         hanoi_recursive(k-1, A, other(A,B));
4:         move(A, B);
5:         hanoi_recursive(k-1, other(A,B), B);
6:     }
7: }
```

Moving a tower of size 0 requires no work, so we wish to show that hanoi_recursive(0, A, B) does nothing.

Suppose k=0. Then
On line 2, if (k>0) will evaluate to false, so lines 3-5 will be skipped.
There are no statements after line 5, so the function does nothing.

Other (IH) restrictions trivially satisfied.

Proving the Inductive Step

```
1: hanoi_recursive( k, A, B ) {
2:       if ( k > 0 ) {
3:             hanoi_recursive(k-1, A, other(A,B));
4:             move(A, B);
5:             hanoi_recursive(k-1, other(A,B), B);
6:       }
7: }
```

Suppose (k > 0) and hanoi_recursive(k-1,A,B) works, (IH).

Line 3: By assumption, conditions of (IH) are currently satisfied, so we can call hanoi_recursive. By (IH), after line 3, there is a stack of size (k-1) at the top of other(A,B), and the ordering property is still satisfied.

Proving the Inductive Step

```
1: hanoi_recursive( k, A, B ) {
2:       if ( k > 0 ) {
3:             hanoi_recursive(k-1, A, other(A,B));
4:             move(A, B);
5:             hanoi_recursive(k-1, other(A,B), B);
6:       }
7: }
```

Suppose (k > 0) and hanoi_recursive(k-1,A,B) works, (IH).

Line 4: Since the ordering property is satisfied, there is no disk smaller than k-1 on peg B. Thus we can move the size k disk from A to B. After this, there is still no disk of size k-1 or smaller on A or B, so ordering property is satisfied.

Proving the Inductive Step

```
1: hanoi_recursive( k, A, B ) {
2:     if ( k > 0 ) {
3:         hanoi_recursive(k-1, A, other(A,B));
4:         move(A, B);
5:         hanoi_recursive(k-1, other(A,B), B);
6:     }
7: }
```

Suppose (k > 0) and hanoi_recursive(k-1,A,B) works, (IH).

Line 5: Since the ordering property is satisfied, we can call hanoi_recursive. We already had a size k disk on B, so now on top of that we have a tower of size k-1. Thus B now has on top a tower of size k. Since hanoi_recursive conserves the ordering property, IS is proven.

Proofs About Programs

- Inductive Hypothesis and Inductive steps can involve words as well as equations.
- Make assertions about the program's *state* after each instruction.
  - It is helpful to find *invariants* – things that don't change.
    - In this case, the invariant was the ordering property – only one stack ever had disks of size (k-1) or smaller after a step.
- There are sometimes more than one way to solve it – find the easiest one.
  - Sometimes it's cleaner to use k-1 as your inductive step instead of k.
- If you are stuck, think about trying to write the program from scratch and fill in the blanks.

How many steps?

```
hanoi_recursive( k, A, B ) {                                        // H(k) = ?
    if ( k == 0 ) {
        // Nothing to do!
    } else {
        hanoi_recursive(k-1, A, other(A,B));
        move(A, B);
        hanoi_recursive(k-1, other(A,B), B);
    }
}
```

How many steps?

```
hanoi_recursive( k, A, B ) {                              // H(k) = ?
        if ( k == 0 ) {
                // Nothing to do!                         // H(0) =0
        } else {
                hanoi_recursive(k-1, A, other(A,B));
                move(A, B);
                hanoi_recursive(k-1, other(A,B), B);
        }
}
```

How many steps?

```
hanoi_recursive( k, A, B ) {                                  // H(k) = ?
        if ( k == 0 ) {
                // Nothing to do!                             // H(0) =0
        } else {
                hanoi_recursive(k-1, A, other(A,B));          // H(k-1)
                move(A, B);                                   // 1
                hanoi_recursive(k-1, other(A,B), B);          // H(k-1)
        }
}
```

Recursion Relation

$$H(k) = H(k\text{-}1) + 1 + H(k\text{-}1)$$
$$= 2H(k\text{-}1) + 1$$

Solving $H(k)=2H(k-1)+1,\ H(0) = 0$

$H(k) = \quad 2\,H(k-1) + 1$

Solving $H(k)=2H(k-1)+1$, $H(0) = 0$

$H(k) = 2\,H(k-1) + 1$

$= 2\,(\,2H(k-2) + 1) + 1 = 4\,H(k-2) + 3$

.

Solving $H(k)=2H(k-1)+1$, $H(0) = 0$

$$
\begin{aligned}
H(k) &= 2\,H(k-1) + 1 \\
&= 2\,(\,2H(k-2) + 1) + 1 = 4\,H(k-2) + 3 \\
&= 4\,(\,2H(k-3) + 1) + 3 = 8\,H(k-2) + 7
\end{aligned}
$$

Solving $H(k)=2H(k-1)+1$, $H(0) = 0$

$$
\begin{aligned}
H(k) &= 2\,H(k-1) + 1 \\
&= 2\,(\,2H(k-2) + 1) + 1 = 4\,H(k-2) + 3 \\
&= 4\,(\,2H(k-3) + 1) + 3 = 8\,H(k-2) + 7 \\
&= 8\,(\,2H(k-4) + 1) + 7 = 16\,H(k-2) + 15
\end{aligned}
$$

…

Solving $H(k)=2H(k-1)+1,\ H(0) = 0$

H(k) =	2 H(k-1) + 1	n=1
= 2 (2H(k-2) + 1) + 1 =	4 H(k-2) + 3	n=2
= 4 (2H(k-3) + 1) + 3 =	8 H(k-2) + 7	n=3
= 8 (2H(k-4) + 1) + 7 =	16 H(k-2) + 15	n=4

…

$$H(k,n) = 2^n H(k-n) + (2^n - 1)$$

Solving $H(k)=2H(k-1)+1,\ H(0) = 0$

H(k) = 2 H(k-1) + 1 n=1

= 2 (2H(k-2) + 1) + 1 = 4 H(k-2) + 3 n=2

= 4 (2H(k-3) + 1) + 3 = 8 H(k-2) + 7 n=3

= 8 (2H(k-4) + 1) + 7 = 16 H(k-2) + 15 n=4

…

$$H(k,n) = 2^n \underbrace{H(k-n)}_{H(0)\,?} + (2^n - 1)$$

Solving $H(k)=2H(k-1)+1$, $H(0) = 0$

H(k) = 2 H(k-1) + 1 n=1

= 2 (2H(k-2) + 1) + 1 = 4 H(k-2) + 3 n=2

= 4 (2H(k-3) + 1) + 3 = 8 H(k-2) + 7 n=3

= 8 (2H(k-4) + 1) + 7 = 16 H(k-2) + 15 n=4

...

$$H(k,n) = 2^n H(k-n) + (2^n - 1)$$

Base Case Substitution: H(k-n) = H(0) => n=k

Solving H(k)=2H(k-1)+1, H(0) = 0

H(k) = 2 H(k-1) + 1 n=1

= 2 (2H(k-2) + 1) + 1 = 4 H(k-2) + 3 n=2

= 4 (2H(k-3) + 1) + 3 = 8 H(k-2) + 7 n=3

= 8 (2H(k-4) + 1) + 7 = 16 H(k-2) + 15 n=4

...

$$H(k,n) = 2^n H(k-n) + (2^n - 1)$$

Base Case Substitution: H(k-n) = H(0) => n=k:

$$H(k) = H(k,k) = 2^k H(0) + \left(2^k - 1\right)$$

Solving H(k)=2H(k-1)+1, H(0) = 0

H(k) =	2 H(k-1) + 1	n=1
= 2 (2H(k-2) + 1) + 1 =	4 H(k-2) + 3	n=2
= 4 (2H(k-3) + 1) + 3 =	8 H(k-2) + 7	n=3
= 8 (2H(k-4) + 1) + 7 =	16 H(k-2) + 15	n=4

…

$$H(k,n) = 2^n H(k-n) + (2^n - 1)$$

Base Case Substitution: H(k-n) = H(0) => n=k:

$$H(k) = H(k,k) = 2^k H(0) + \left(2^k - 1\right) = 2^k \cdot 0 + 2^k - 1 = 2^k - 1.$$

Complexity of Hanoi?

Complexity of Hanoi

$$O(2^k)$$

Exponential

Does this really work?

“Look at the pattern” is a bit hand wave-y. Can we prove it?

Yes – use induction (on n)!

Prove Recurrence Solution Using Induction

Inductive Hypothesis:

$H(k,n) = 2^n H(k-n) + (2^n - 1)$ for all k <= K

Base Case:

By definition, H(k,1) = H(k).

$H(k,1) = H(k) = 2H(k-1) + 1 = 2H(k\ -1{,}1) = 2^1 H(k-1) + (2^1 - 1).$

Inductive Step:

$H(k,n+1) = 2^n\big(H(k-(n+1)+1) + (2^n + 1)\big)$
$= 2^{n+1}H\big(k-(n+1)\big) + 2^n + 2^n + 1 = 2^{n+1}H\big(k-(n+1)\big) + 2 \cdot 2^n + 1$
$= 2^{n+1}H\big(k-(n+1)\big) + (2^{n+1} + 1).$

What about the other base case?

```
hanoi_recursive( k, A, B ) {                                    // H(k) = 2H(k-1)+1
        if ( k == 1) {
                move(A, B);                                     //H(1) = 1
        } else {
                hanoi_recursive(k-1, A, other(A,B));  // H(k-1)
                move(A, B);                                     // O(1)
                hanoi_recursive(k-1, other(A,B), B);  // H(k-1)
        }
}
```

What about the other base case?

H(K) = 2H(k-1) + 1, H(1) = 1

Recurrence has same form, so generalized version is the same.

$$H(k,n) = 2^n H(k-n) + (2^n - 1)$$

Base Case Substitution is more complicated:

Need k-n = 1, so n = k-1

$$H(k) = H(k,k-1) = 2^{k-1} H\big(k-(k-1)\big) + \big(2^{k-1} - 1\big)$$
$$= 2^{k-1} H(1) + 2^{k-1} - 1 = 2^{k-1} + 2^{k-1} - 1 = 2\big(2^{k-1}\big) - 1 = 2^k - 1$$

The inductive step of the recurrence relation has a similar complication.

More Induction (if we have time)

- Hockey Stick Identity
- Euler's Formula: E-V+F = 2