Recursion

Alexander S. Kulikov

Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences
and
University of California, San Diego

Outline

Recursion

Coin Problem

Hanoi Towers

Line to Louvre

By Edal Anton Lefterov, `https://commons.wikimedia.org/w/index.php?curid=11806863`

Computing Queue Length

- Alice gets in line

Computing Queue Length

- Alice gets in line
- She wants to know the number of people before her in the line

Computing Queue Length

- Alice gets in line
- She wants to know the number of people before her in the line
- She asks a person (call him Bob) standing before her in the line: Could you please tell me the number of people before you?

Computing Queue Length

- Alice gets in line
- She wants to know the number of people before her in the line
- She asks a person (call him Bob) standing before her in the line: Could you please tell me the number of people before you?
- Now, Bob faces exactly the same problem

Computing Queue Length

- Alice gets in line
- She wants to know the number of people before her in the line
- She asks a person (call him Bob) standing before her in the line: Could you please tell me the number of people before you?
- Now, Bob faces exactly the same problem
- Bob somehow computes that the number of people before him is 239

Computing Queue Length

- Alice gets in line
- She wants to know the number of people before her in the line
- She asks a person (call him Bob) standing before her in the line: Could you please tell me the number of people before you?
- Now, Bob faces exactly the same problem
- Bob somehow computes that the number of people before him is 239
- Now, Alice knows that the number of people before her is 240

Recursive Program

```
numberOfPeopleBefore(A):
if there is nobody before A:
  return 0
B ← person right before A
return numberOfPeopleBefore(B) + 1
```

Factorial Function

Definition

For a positive integer n, the factorial of n is the product of integers from 1 to n.

Factorial Function

Definition

For a positive integer n, the factorial of n is the product of integers from 1 to n.

$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120 = 4! \times 5$$

Factorial Function

Definition

For a positive integer n, the factorial of n is the product of integers from 1 to n.

$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120 = 4! \times 5$$

Recursive definition

$$n! = \begin{cases} 1 & \text{if } n = 1 \\ n \times (n-1)! & \text{if } n > 1 \end{cases}$$

Iterative Program

```python
def factorial(n):
    assert(n > 0)
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

Recursive Program

```
def factorial(n):
    assert(n > 0)
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

Termination

- One must ensure that a recursive program (or definition) terminates after a finite number of steps

Termination

- One must ensure that a recursive program (or definition) terminates after a finite number of steps
- This is usually achieved by decreasing some parameter until it reaches a base case

Termination

- One must ensure that a recursive program (or definition) terminates after a finite number of steps
- This is usually achieved by decreasing some parameter until it reaches a base case
    - Line length: the length of the line decreases by 1 with each recursive call, until it becomes 1

Termination

- One must ensure that a recursive program (or definition) terminates after a finite number of steps
- This is usually achieved by decreasing some parameter until it reaches a base case
    - Line length: the length of the line decreases by 1 with each recursive call, until it becomes 1
    - Factorial: n decreases to 1

Example of Infinite Recursion

```python
def infinite(n):
    if n == 1:
        return 0
    return n * infinite(n + 1)
```

Example of Infinite Recursion

```
def infinite(n):
    if n == 1:
        return 0
    return n * infinite(n + 1)
```

- In theory, will never stop: the parameter increases to infinity

Example of Infinite Recursion

```
def infinite(n):
    if n == 1:
        return 0
    return n * infinite(n + 1)
```

- In theory, will never stop: the parameter increases to infinity
- In practice, will cause an error message like `stack overflow` or `maximum recursion depth exceeded`

More Examples of Infinite Recursion

- No base case:

```
def factorial(n):
    return n * factorial(n - 1)
```

More Examples of Infinite Recursion

- No base case:

```
def factorial(n):
    return n * factorial(n - 1)
```

- Parameter does not change:

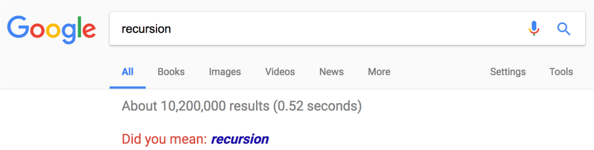
```
def infinite(n):
    if n == 1:
        return 0
    return 1 + infinite(n)
```

More (Jokey) Examples

- To understand recursion, you must first understand recursion

More (Jokey) Examples

- To understand recursion, you must first understand recursion
- By Google:



Outline

Recursion

Coin Problem

Hanoi Towers

Coin Problem

Problem

Prove that any monetary amount starting from 8 can be paid using coins of denominations 3 and 5.

Coin Problem

Problem

Prove that any monetary amount starting from 8 can be paid using coins of denominations 3 and 5.

- Looks plausibly: $8 = 3 + 5$, $9 = 3 + 3 + 3$, $10 = 5 + 5$, $11 = 3 + 3 + 5$

Coin Problem

Problem

Prove that any monetary amount starting from 8 can be paid using coins of denominations 3 and 5.

- Looks plausibly: $8 = 3 + 5$, $9 = 3 + 3 + 3$, $10 = 5 + 5$, $11 = 3 + 3 + 5$
- But how can we be sure that it is always possible?

Speculating

- OK, we can definitely pay 8

Speculating

- OK, we can definitely pay 8
- But then we can also pay $11 = 8 + 3$ (adding one 3-coin)

Speculating

- OK, we can definitely pay 8
- But then we can also pay 11 $= 8 + 3$ (adding one 3-coin)
- Then, 14, 17, 20, etc

Speculating

- OK, we can definitely pay 8
- But then we can also pay $11 = 8 + 3$ (adding one 3-coin)
- Then, 14, 17, 20, etc
- Similarly, 9 gives 12, 15, 18, 21, etc

Speculating

- OK, we can definitely pay 8
- But then we can also pay $11 = 8 + 3$ (adding one 3-coin)
- Then, 14, 17, 20, etc
- Similarly, 9 gives 12, 15, 18, 21, etc
- While 10 gives 13, 16, 19, 22, etc

Recursive Program

```
def change(amount):
    assert(amount >= 8)
    if amount == 8:
        return [3, 5]
    if amount == 9:
        return [3, 3, 3]
    if amount == 10:
        return [5, 5]

    coins = change(amount - 3)
    coins.append(3)
    return coins
```

Outline

Recursion

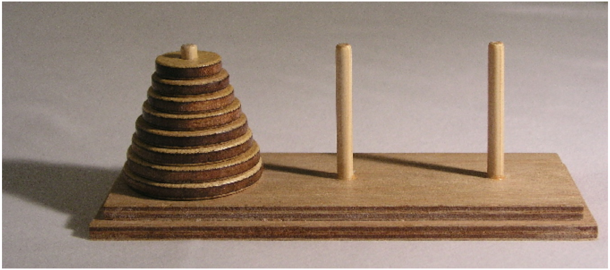
Coin Problem

Hanoi Towers

Hanoi Towers

Puzzle

There are three sticks with n discs sorted by size on one of the sticks. The goal is to move all n discs to another stick subject to two constraints: move one disc at a time and don't place a larger disc on a smaller one.



https://commons.wikimedia.org/w/index.php?curid=228623

Is It Even Possible?

- For what n this is possible?

Is It Even Possible?

- For what *n* this is possible?
- For all!

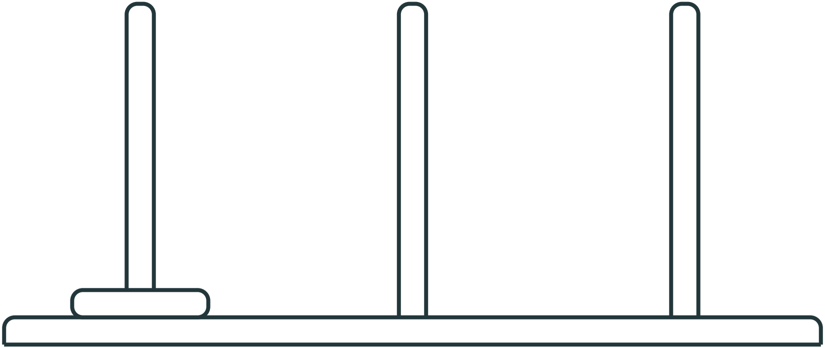
Is It Even Possible?

- For what n this is possible?
- For all!
- But how can we be sure?

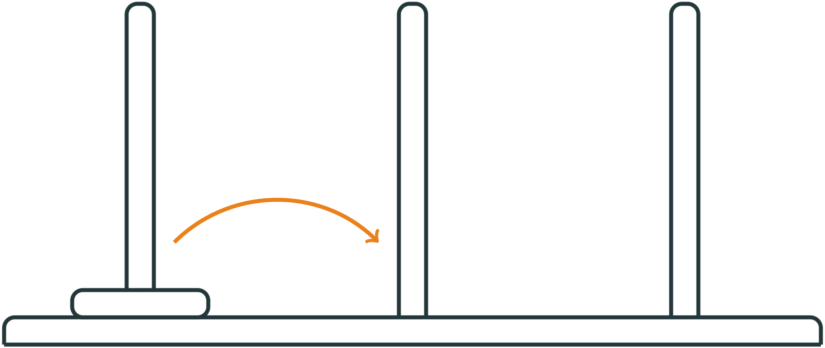
Is It Even Possible?

- For what n this is possible?
- For all!
- But how can we be sure?
- We will design a recursive program that will solve the puzzle for every n

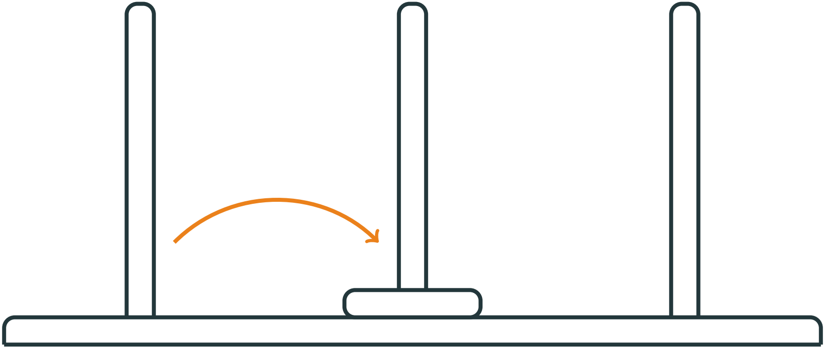
Simplest Possible Scenario



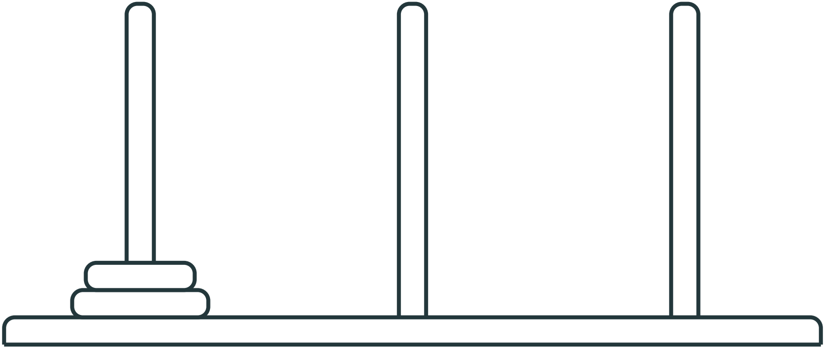
Simplest Possible Scenario



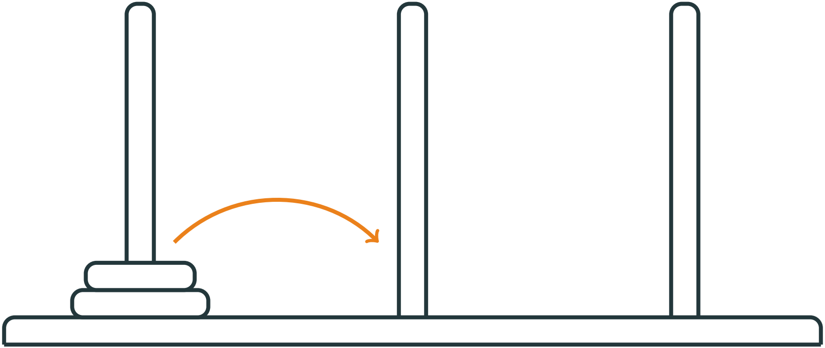
Simplest Possible Scenario



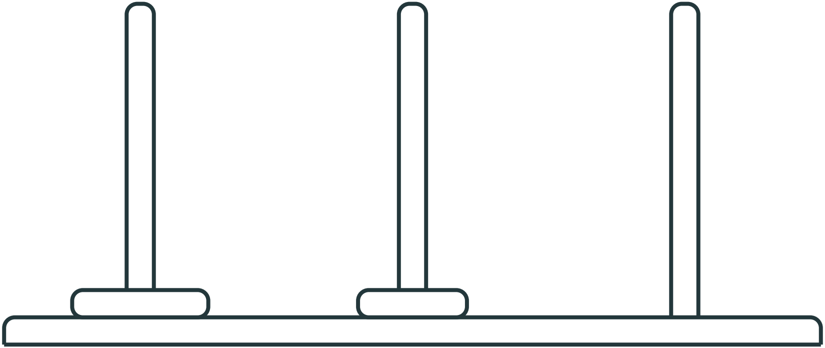
Two Discs



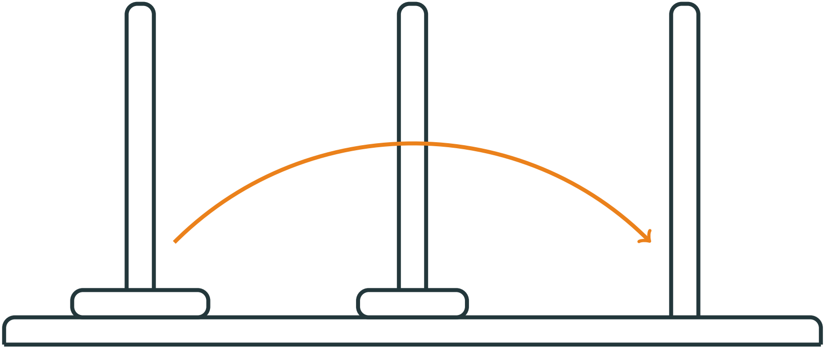
Two Discs



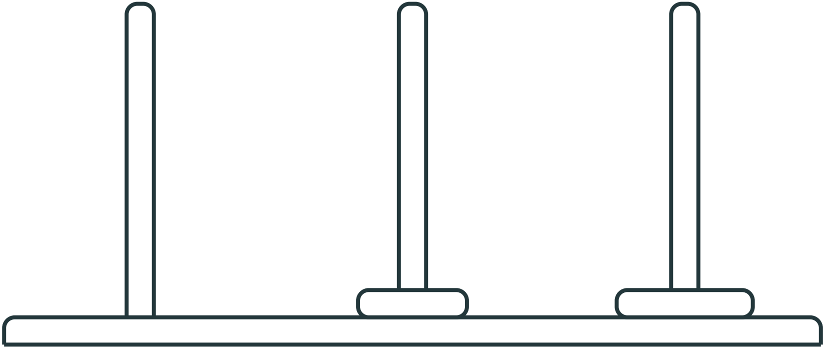
Two Discs



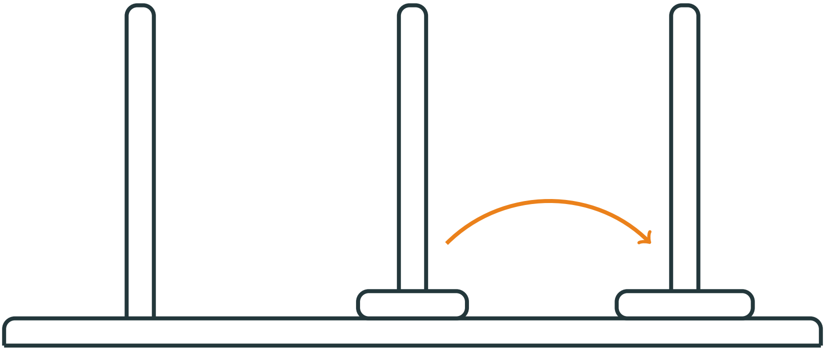
Two Discs



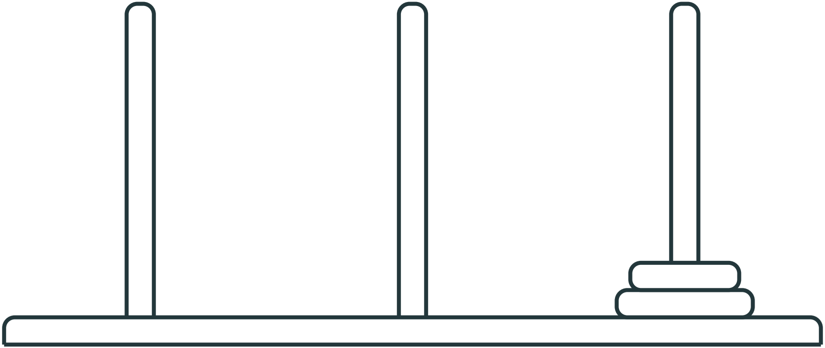
Two Discs



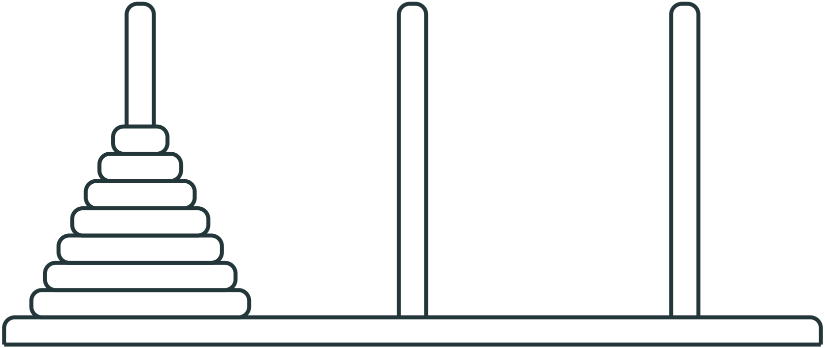
Two Discs



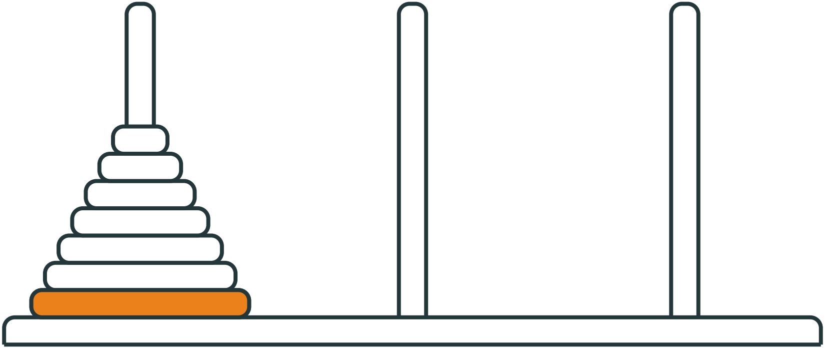
Two Discs



Speculating

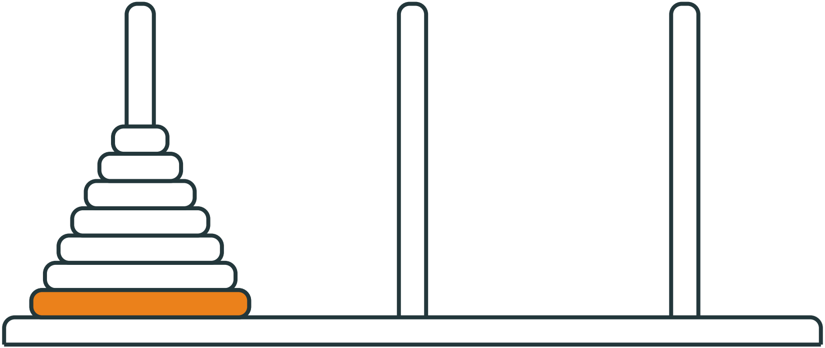


Speculating



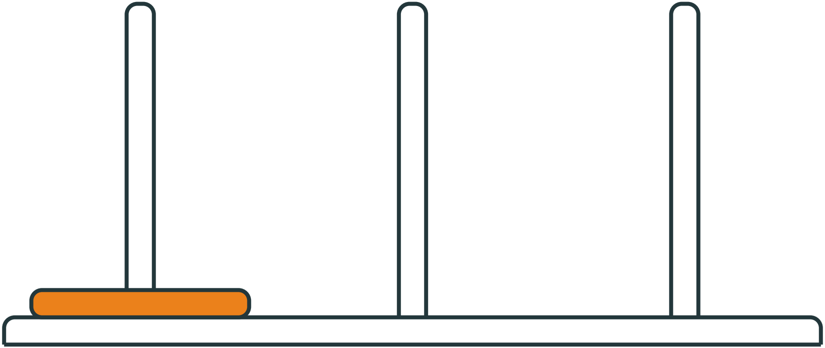
consider the largest disc

Speculating



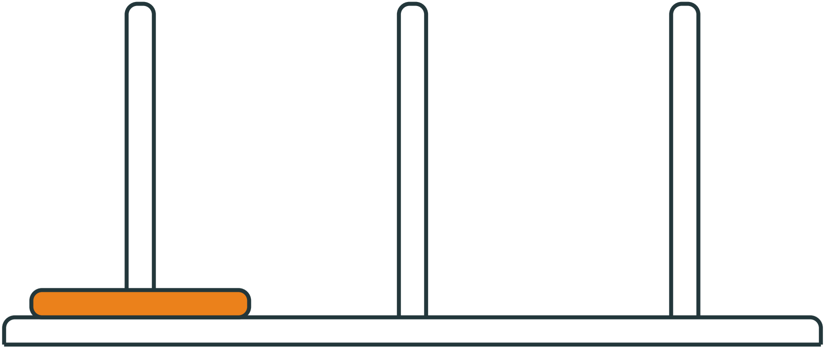
when can we move it?

Speculating



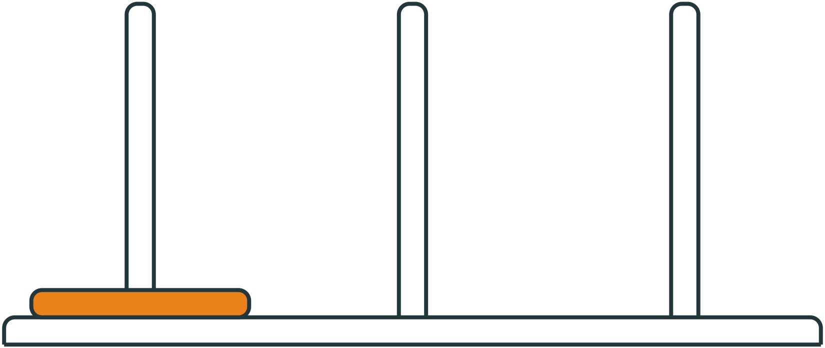
when there are no other discs on this stick!

Speculating



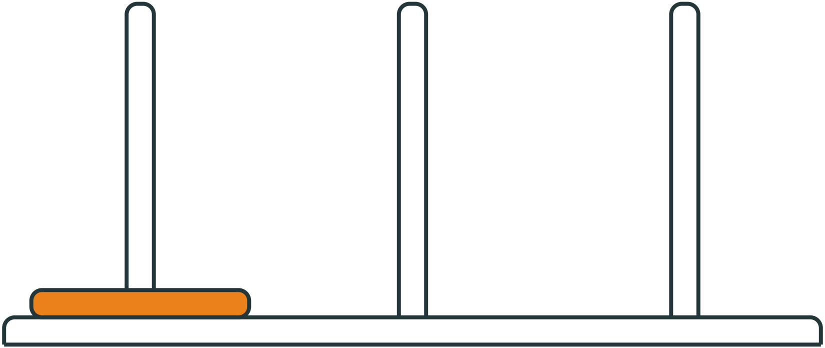
where can we move it?

Speculating



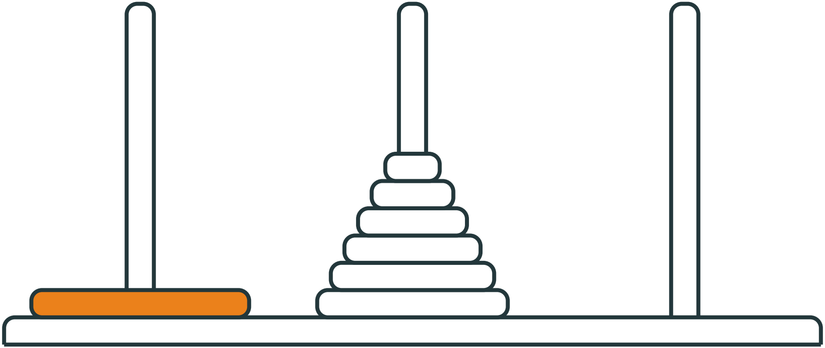
only to an empty stick!

Speculating



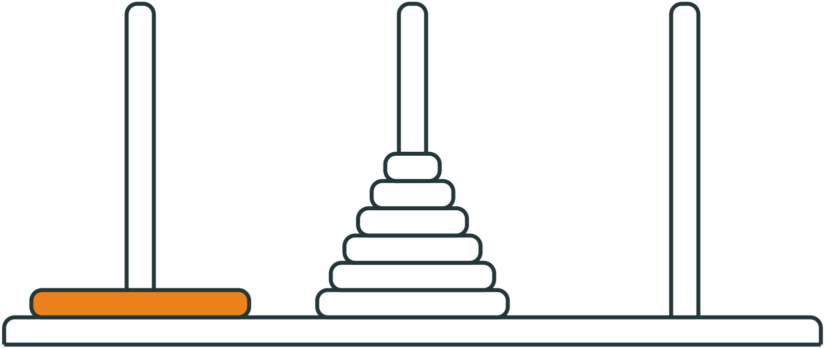
all other discs should be at another stick

Speculating



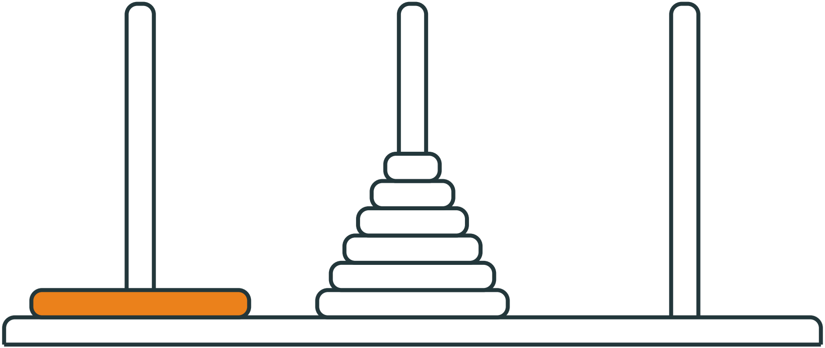
all other discs should be at another stick

Speculating



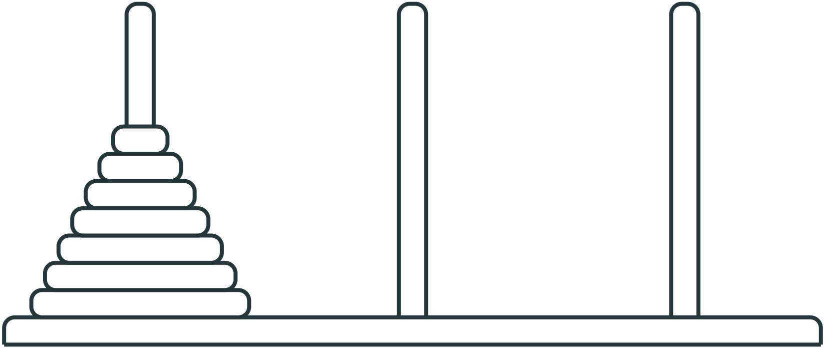
need to move $n-1$ discs

Speculating

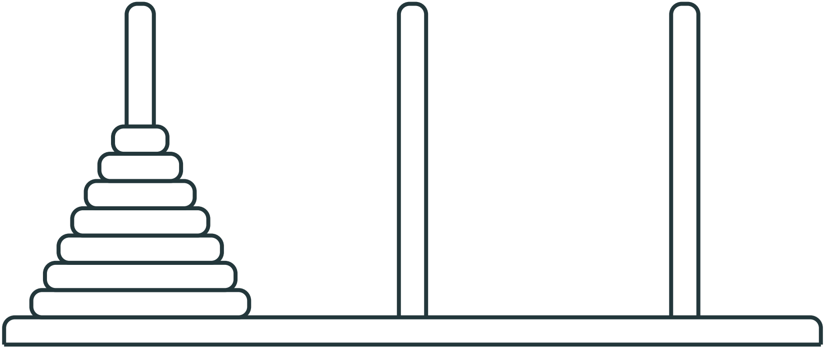


this is the same problem!

Recursive Solution

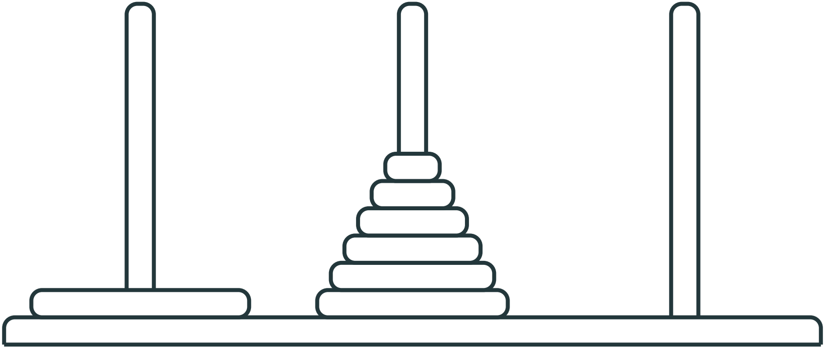


Recursive Solution



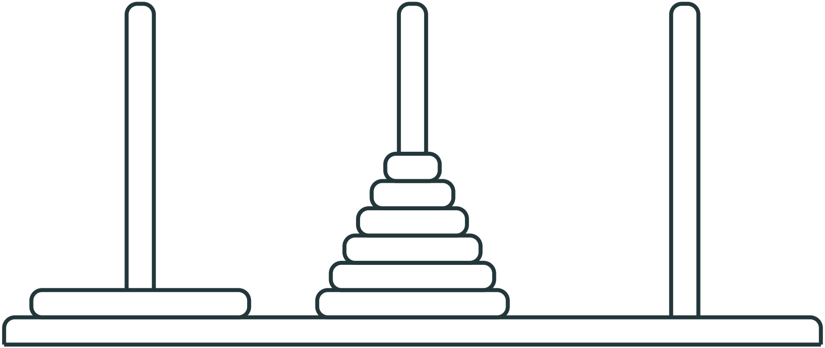
move $n - 1$ discs recursively

Recursive Solution



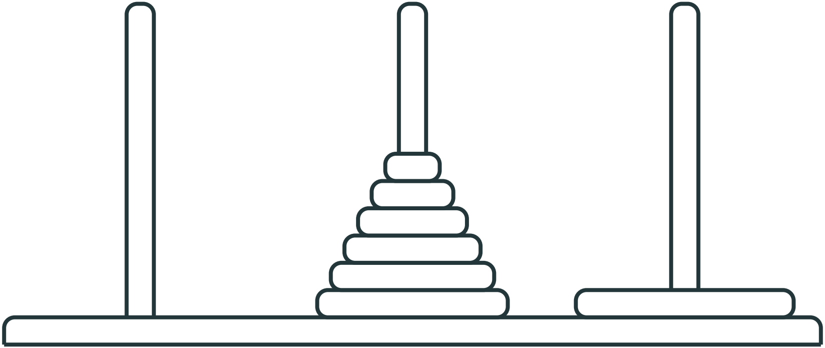
move $n - 1$ discs recursively

Recursive Solution



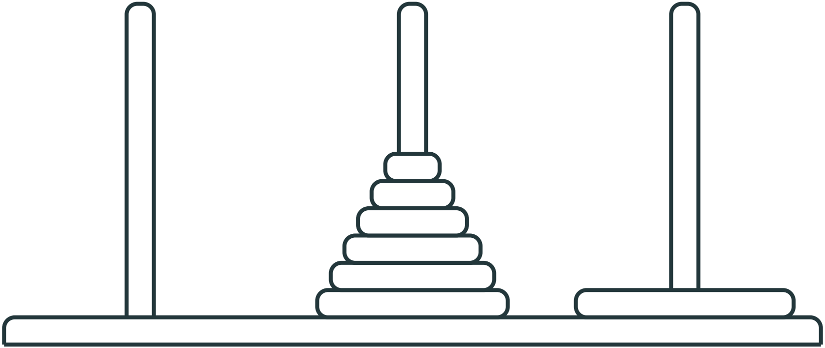
move the largest disk

Recursive Solution



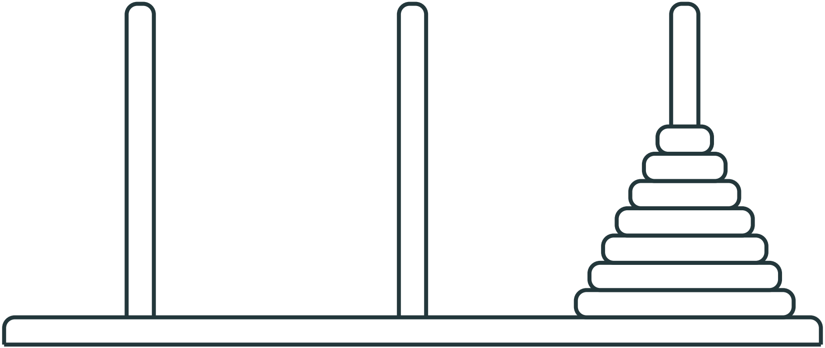
move the largest disk

Recursive Solution



move $n-1$ discs recursively

Recursive Solution



move $n-1$ discs recursively

Summary

- We've constructed a solution for all n:
  - Base case: it is possible for $n = 1$
  - Hence, it is possible for $n = 2$
  - Hence, it is possible for $n = 3$
  - $\ldots$
- We'll later learn a related notion of induction