Recursion 2

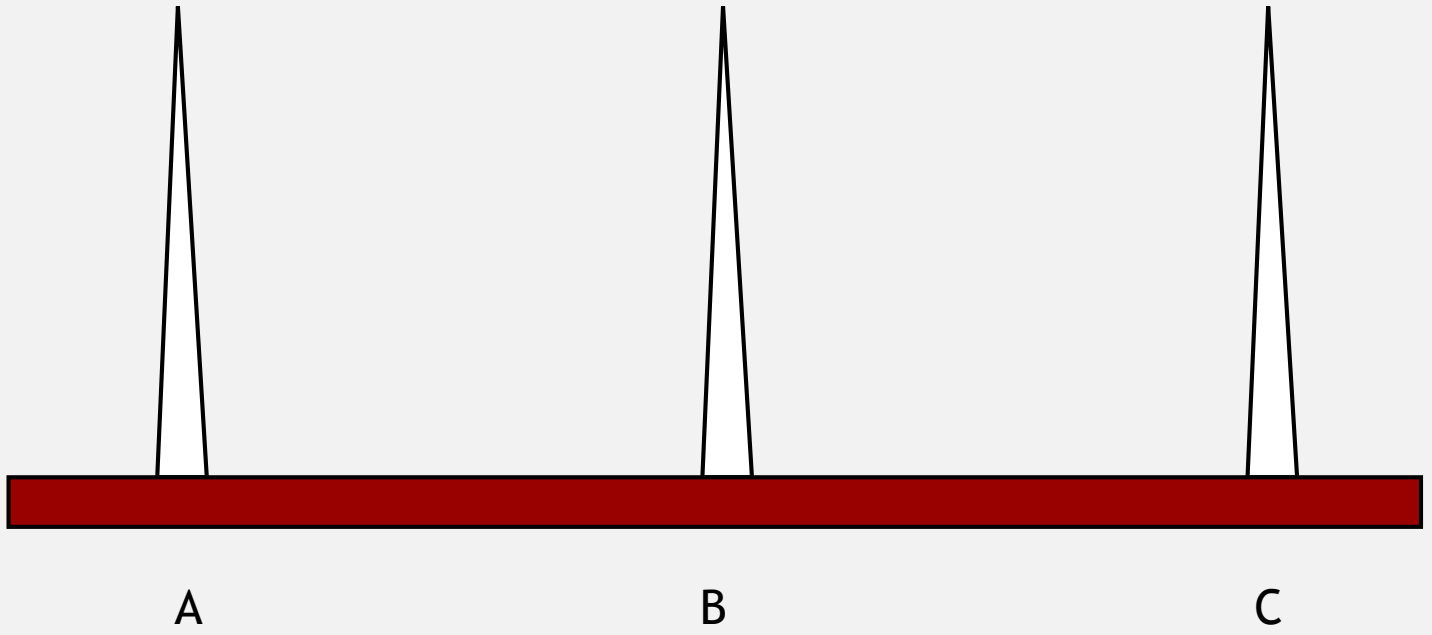
Mark Matthews PhD

Recursion vs Iteration

- Any recursive algorithm can be re-implemented as a loop instead
  - This is an “iterative” expression of the algorithm
- Any loop can be implemented as recursion instead
- Sometimes recursion is clearer and simpler
  - Mostly for data structures with a recursive structure
- Sometimes iteration is clearer and simpler

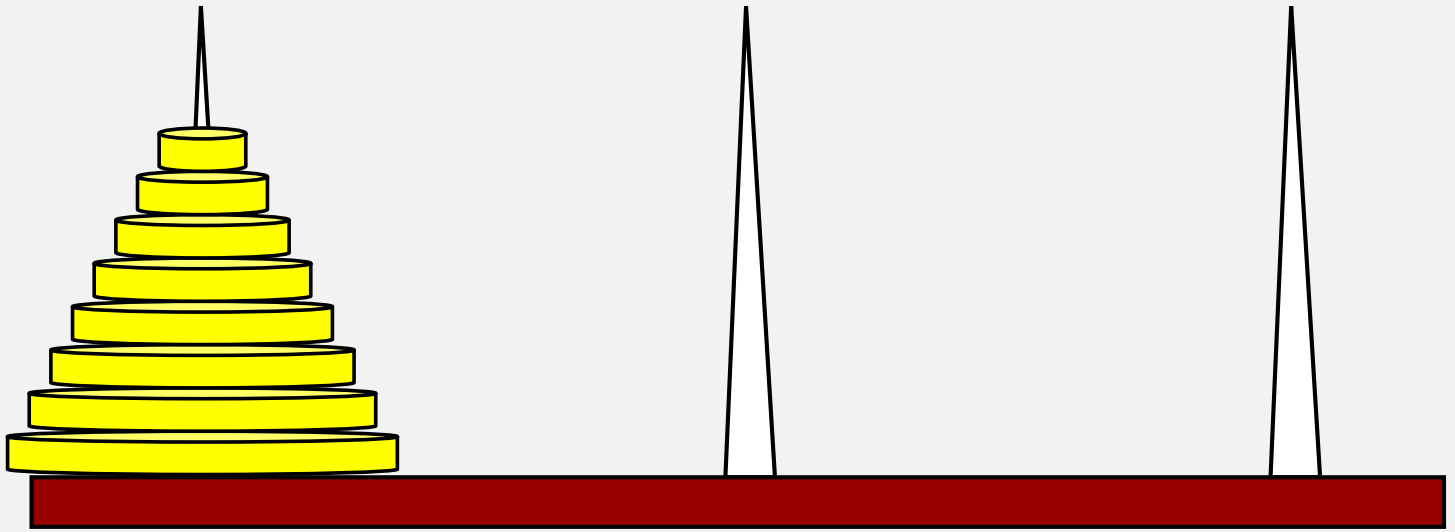
Towers of Hanoi Legend

According to legend, there ARE three diamond needles in the floor of the temple of Brahma in Hanoi.



Towers of Hanoi Legend

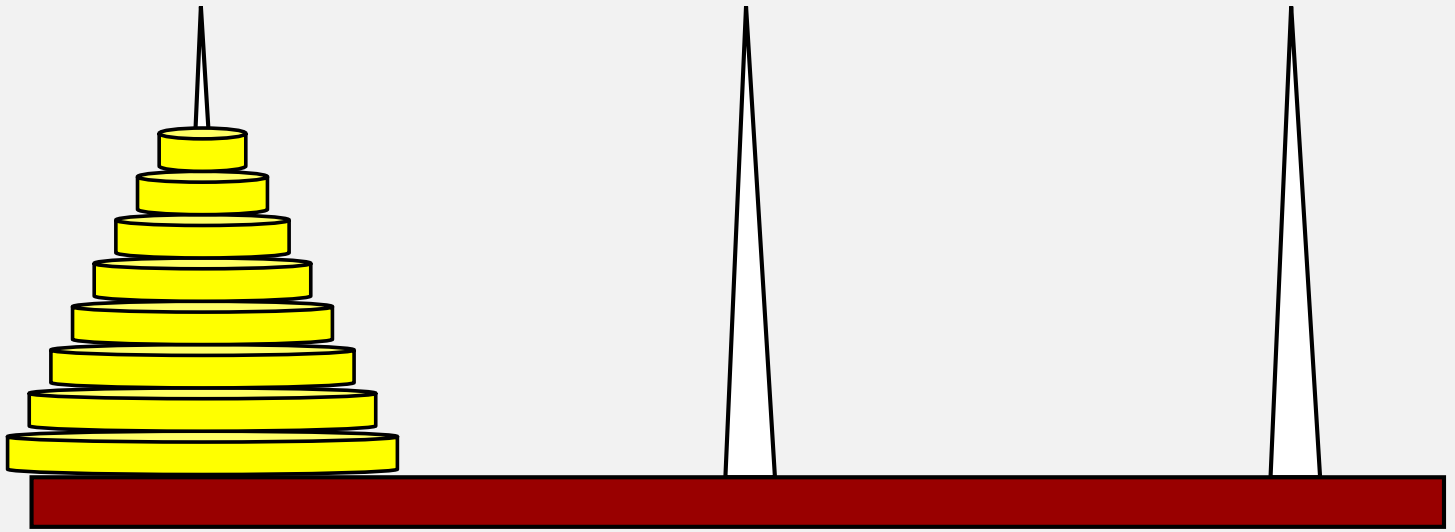
According to legend, there ARE three diamond needles in the floor of the temple of Brahma in Hanoi.



On the left needle, there are 64 golden disks in decreasing size.

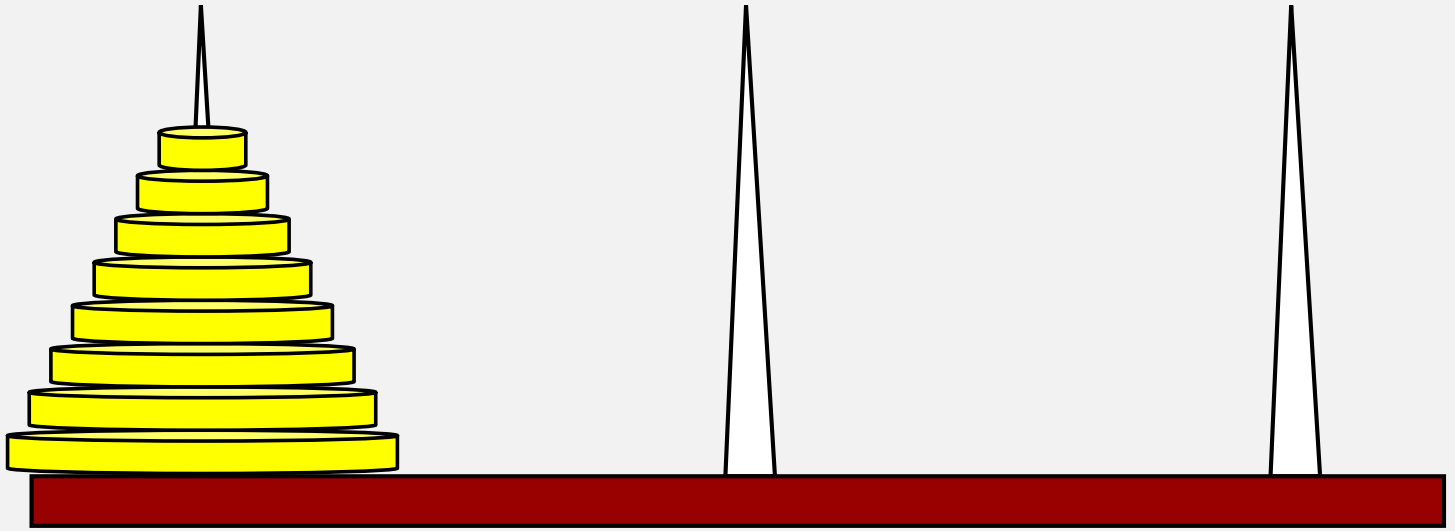
Towers of Hanoi Legend

The monks have to transfer ALL the disks from the first needle to the last.
They can use the 2nd needle to help.



Towers of Hanoi Rules

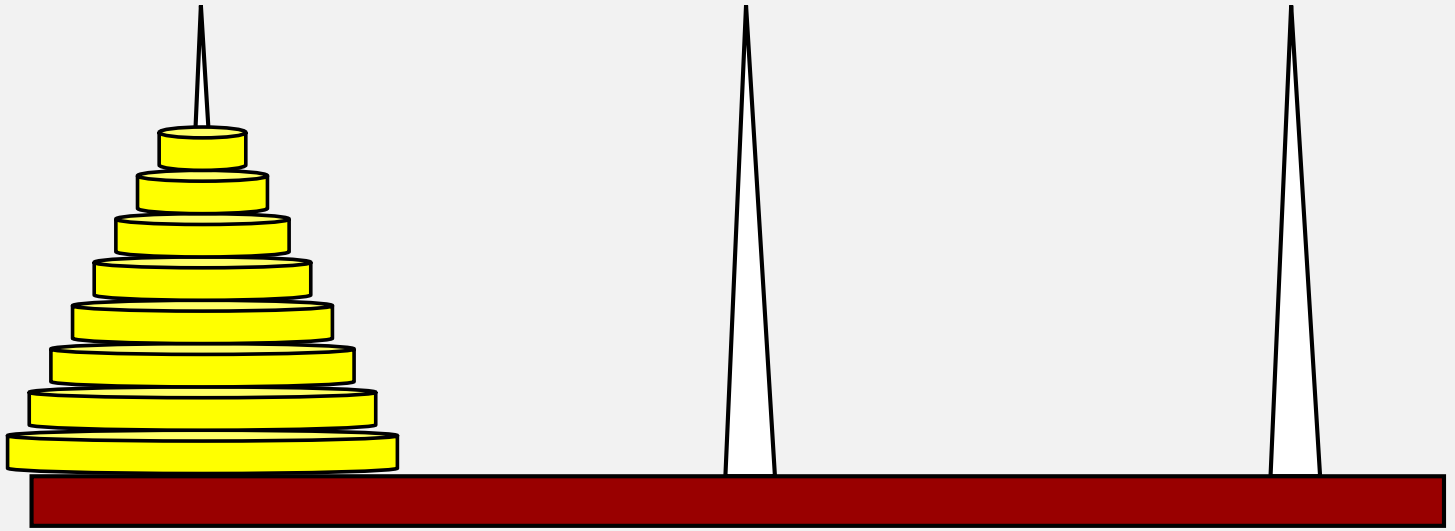
The monks have to transfer ALL the disks from the first needle to the last.
They can use the 2nd needle to help.



1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
3. No larger disk may be placed on top of a smaller disk.

Towers of Hanoi Rules

The monks have to transfer ALL the disks from the first needle to the last.
They can use the 2nd needle to help.

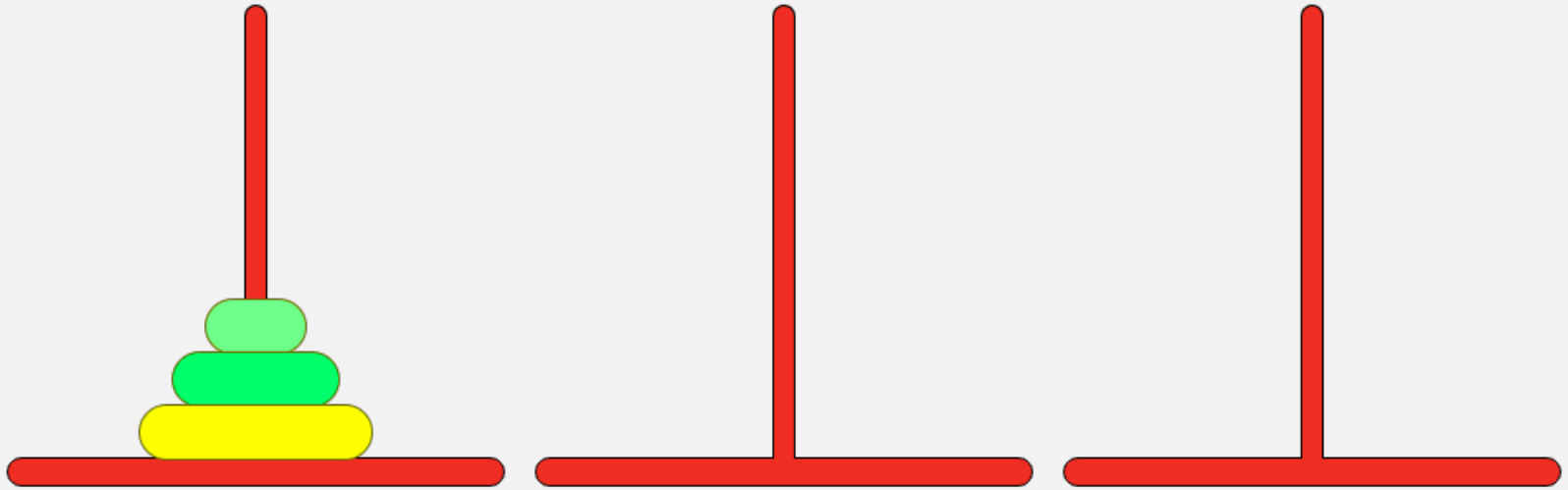


1. Only one disk can be moved at a time.
2. A disc can be placed either on empty peg or on top of a larger disc.
3. No larger disk may be placed on top of a smaller disk.

Will the world end?

Q. Will computer algorithms help?
Find out today

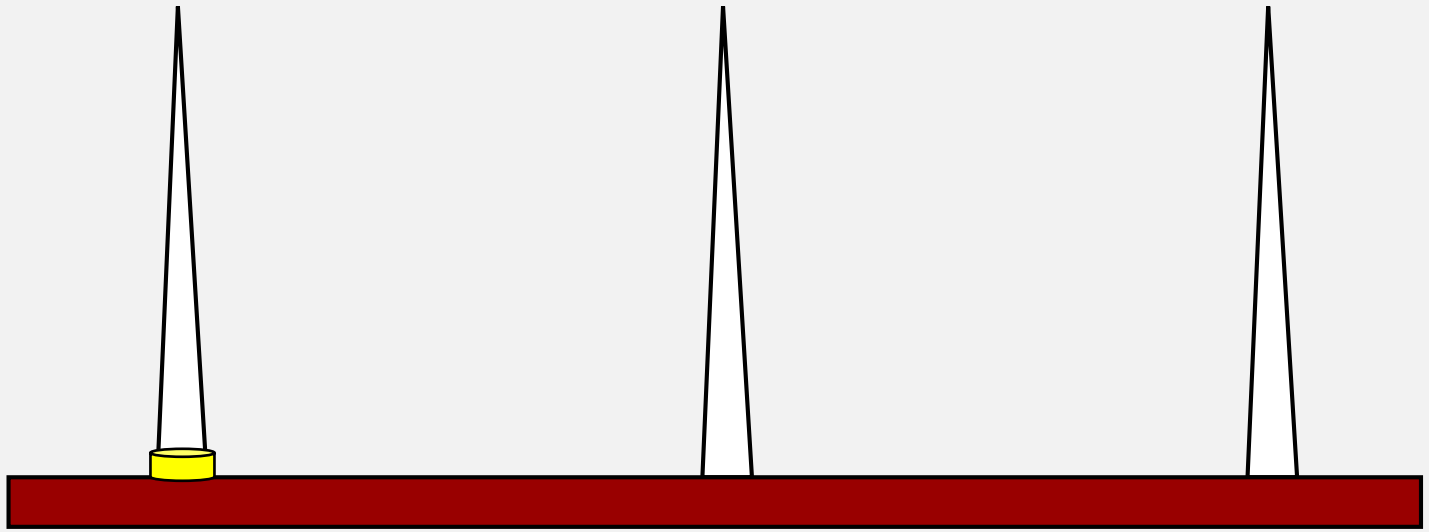
Take a few minutes to play the game



https://www.mathsisfun.com/games/towerofhanoi.html

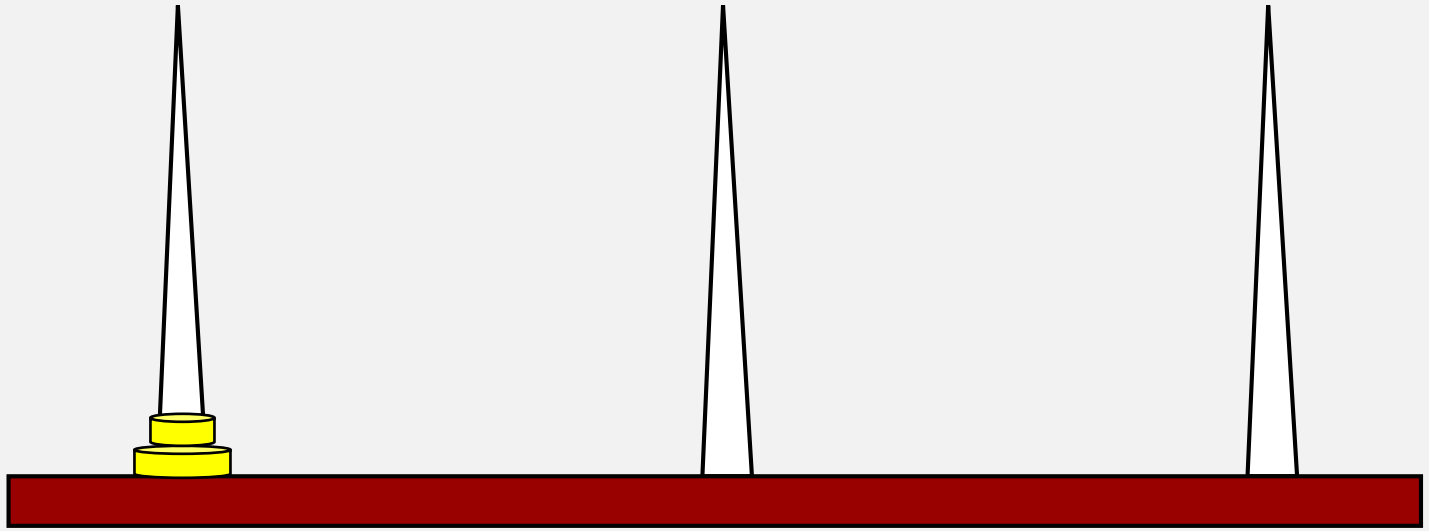
Towers of Hanoi Legend

With 1 disk the problem is trivial.



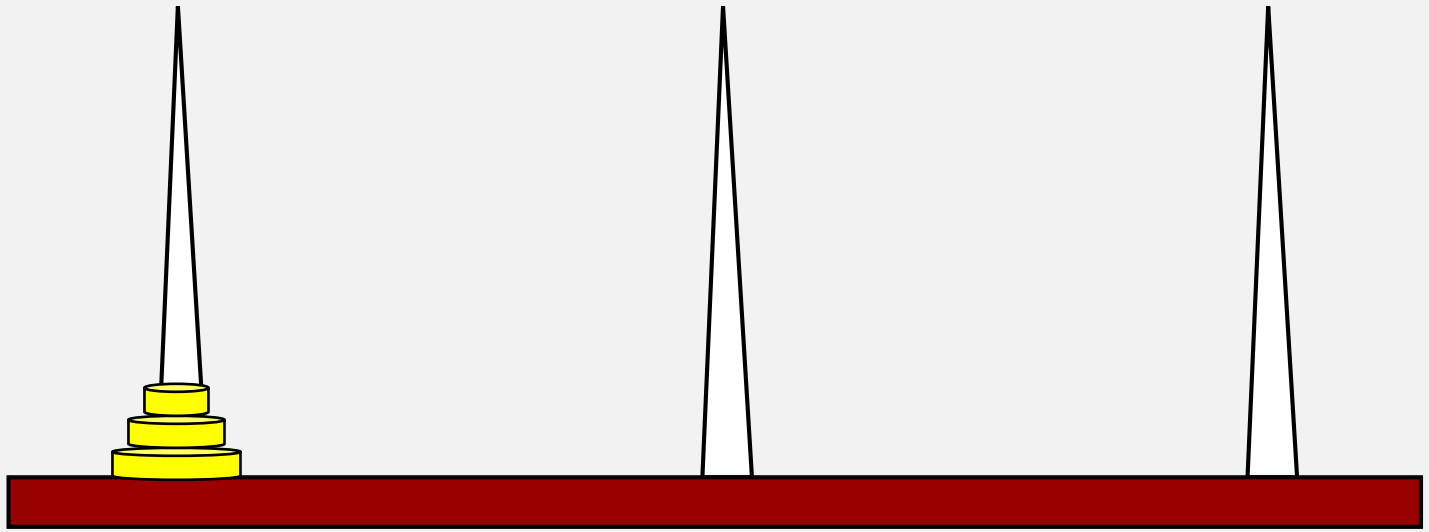
Towers of Hanoi Legend

A little more complicated with 2 but still easy.

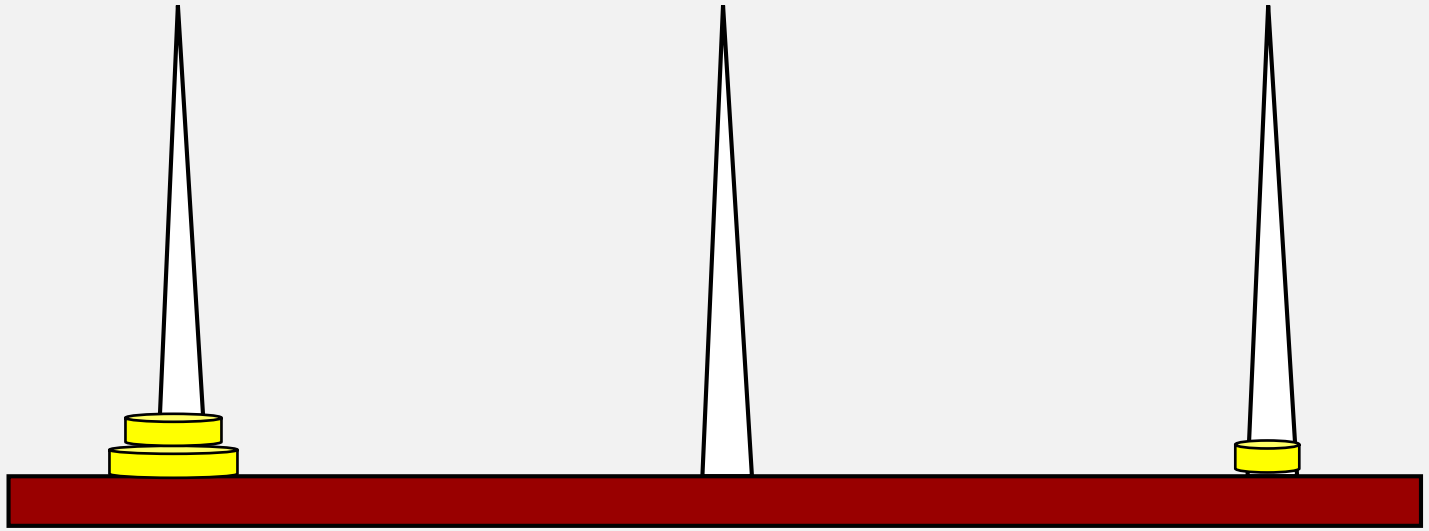


Towers of Hanoi Legend

Let's look at the problem with 3 disks.

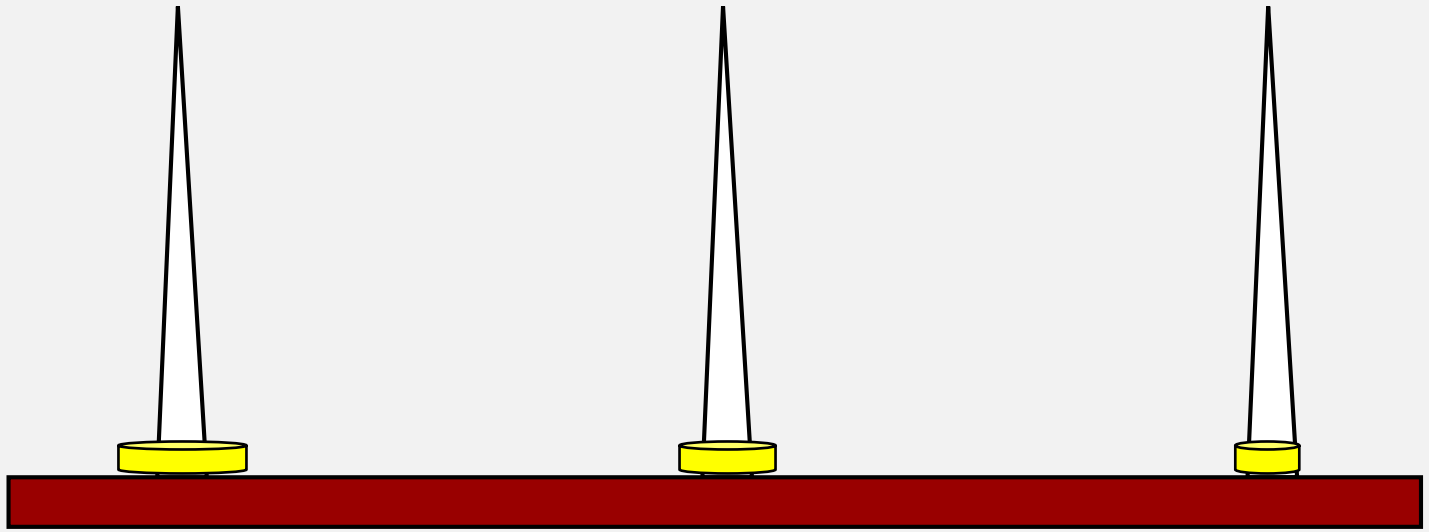


Towers of Hanoi Legend



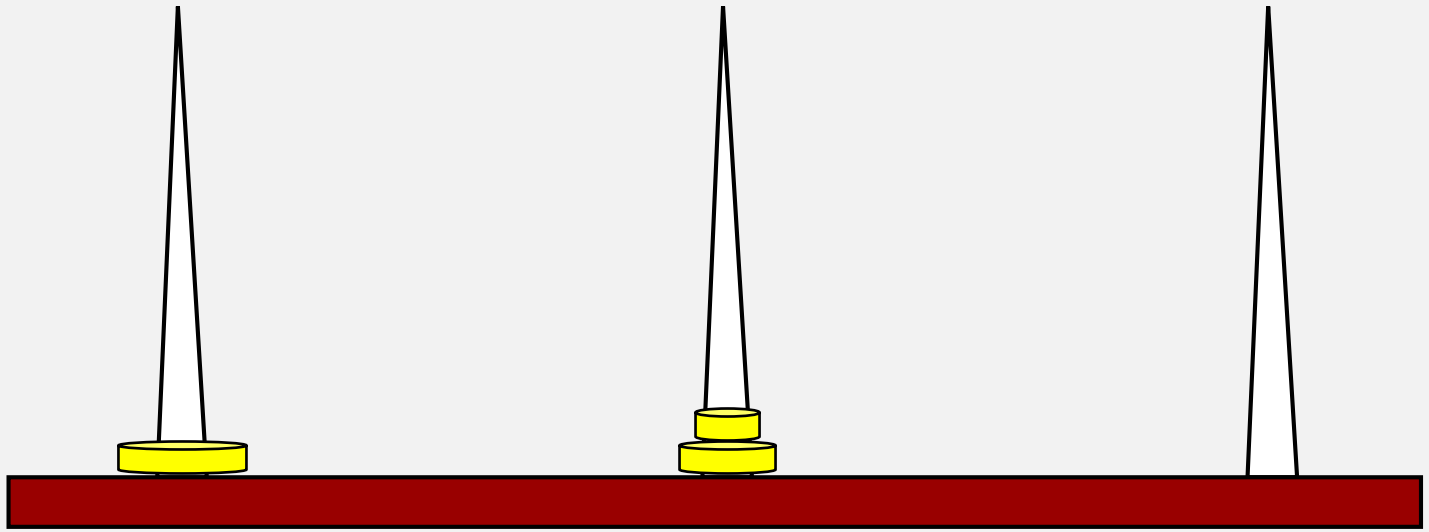
Towers of Hanoi Legend

Let's look at the problem with 3 disks.



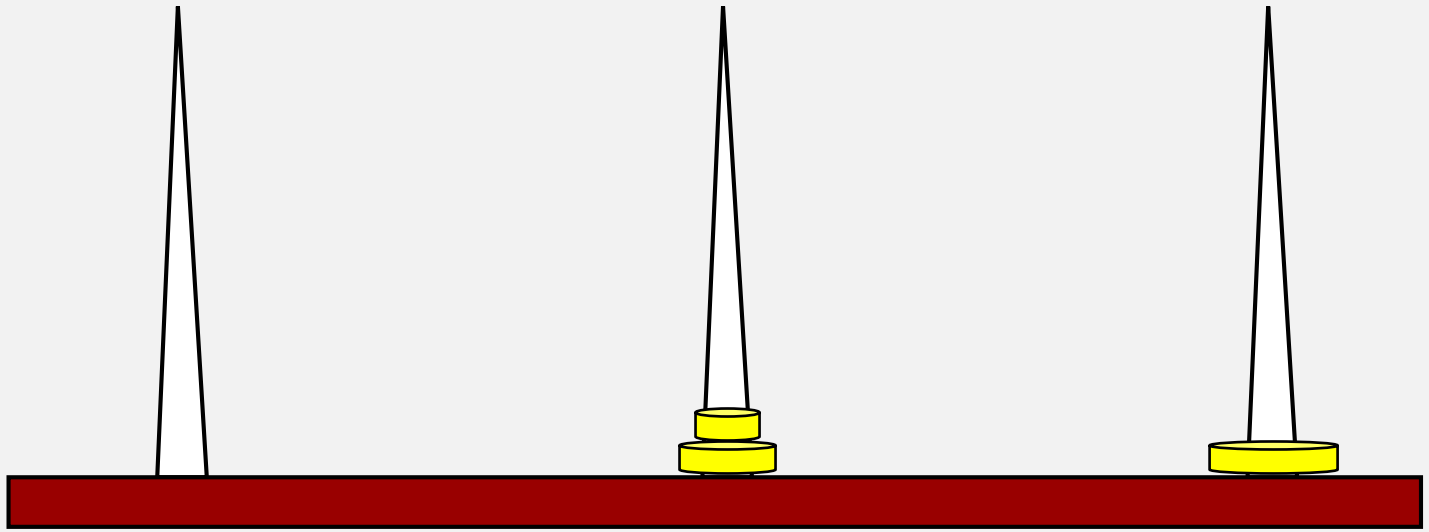
Towers of Hanoi Legend

Let's look at the problem with 3 disks.



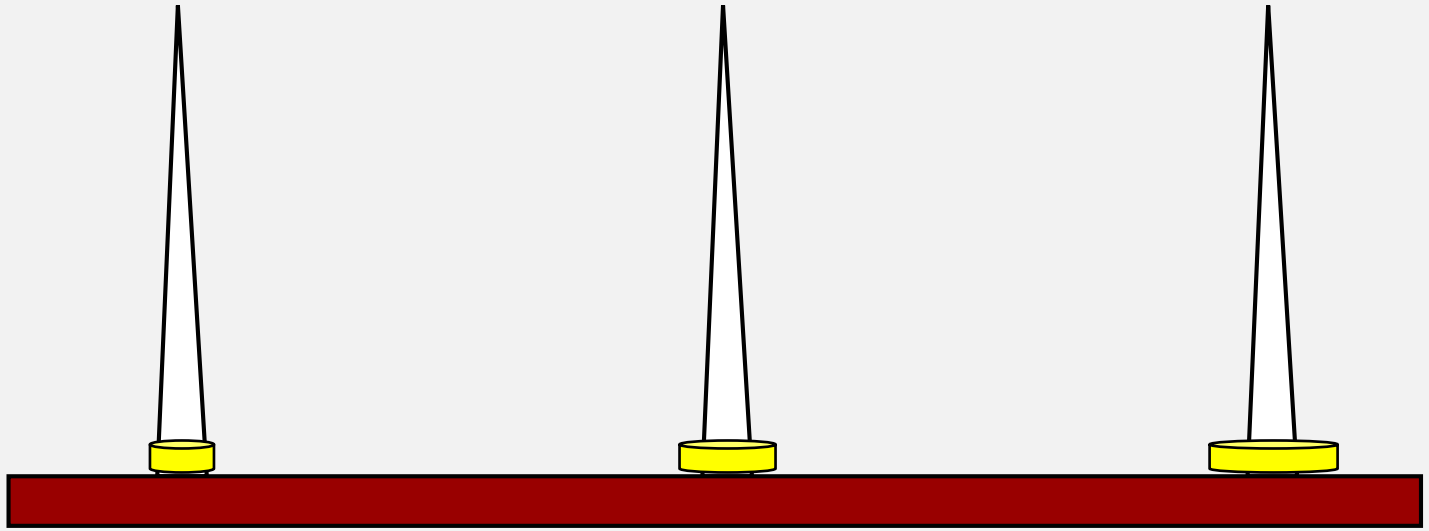
Towers of Hanoi Legend

Let's look at the problem with 3 disks.



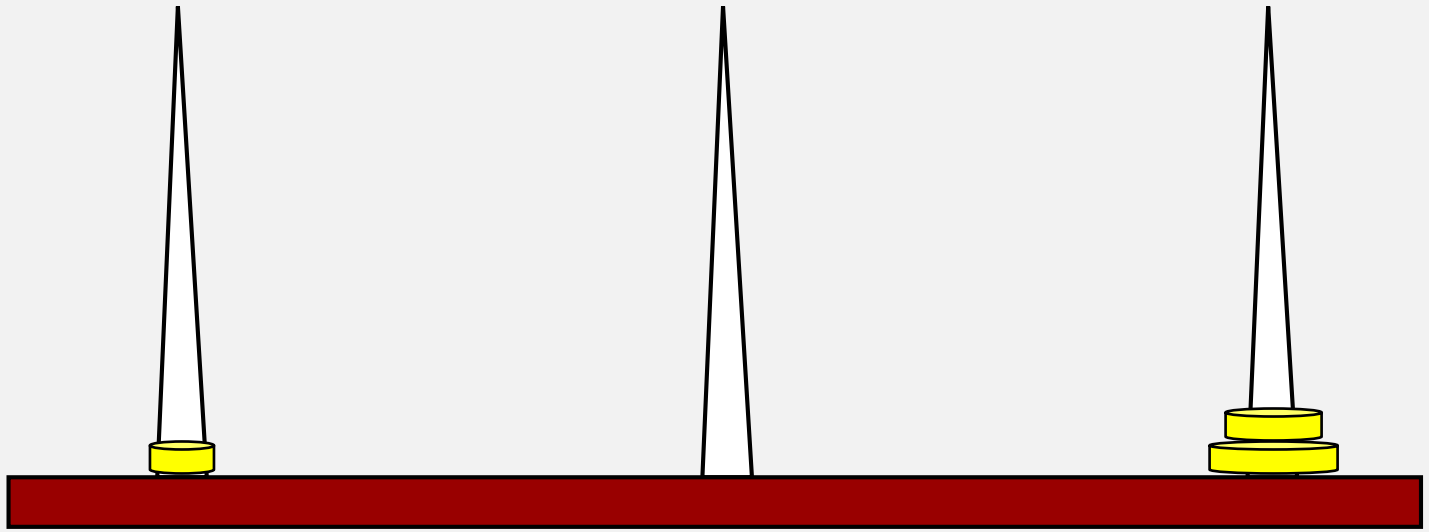
Towers of Hanoi Legend

Let's look at the problem with 3 disks.

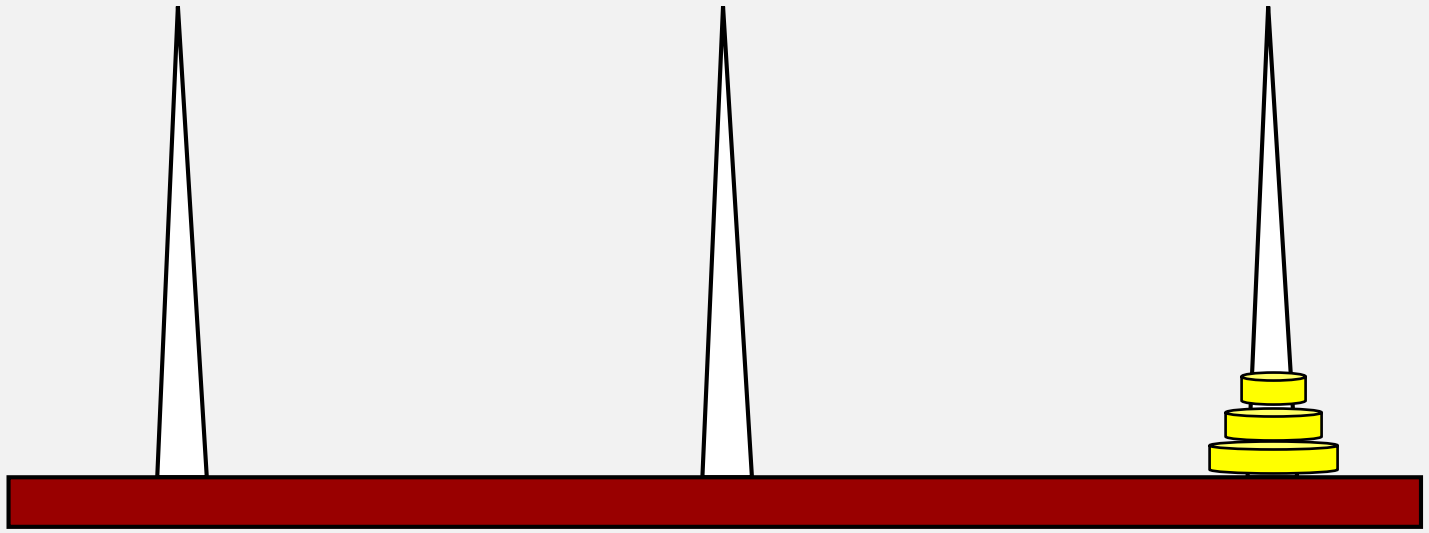


Towers of Hanoi Legend

Let's look at the problem with 3 disks.



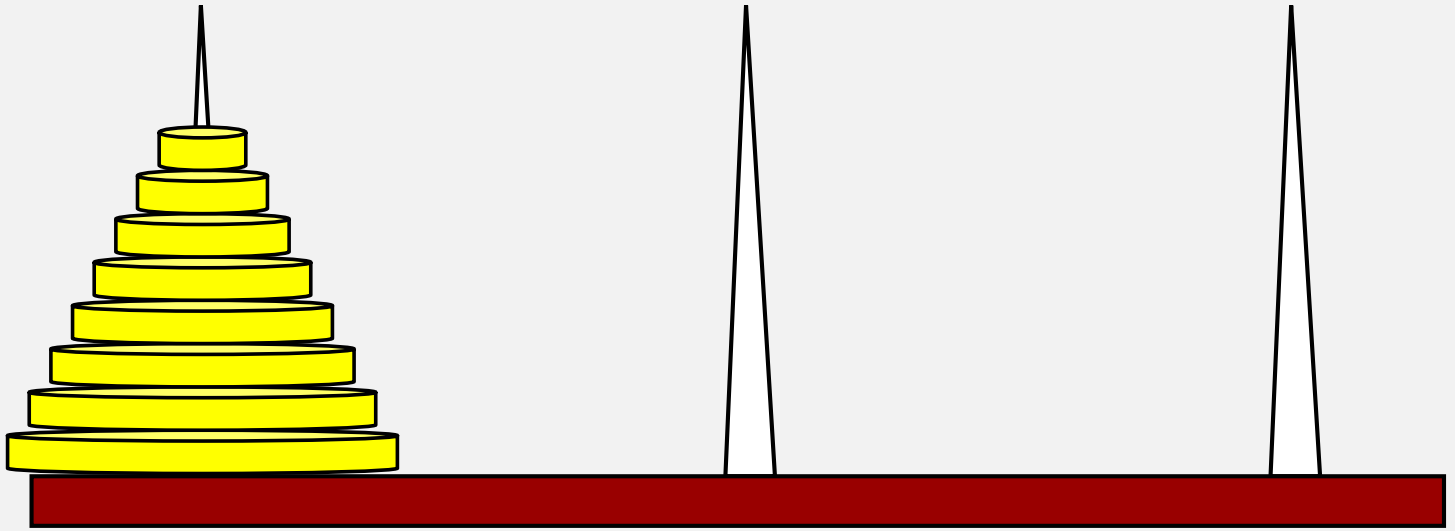
Towers of Hanoi Legend



Job done.

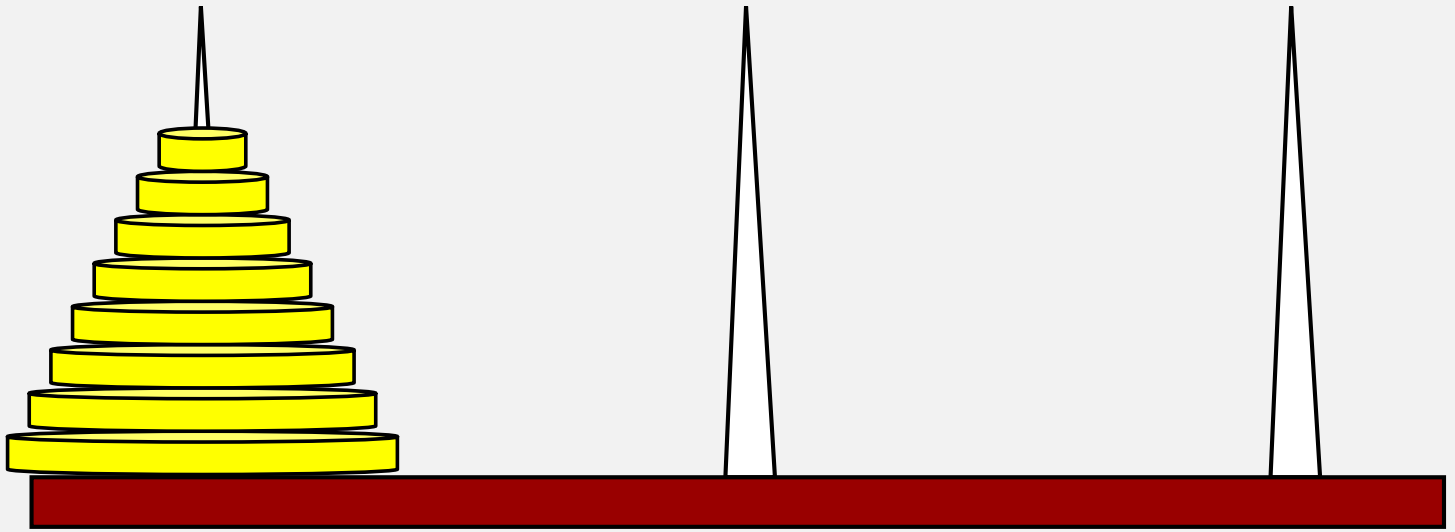
Towers of Hanoi Legend

Our challenge is to write an algorithm that generates instructions for the monks to move the disks.



Challenging to write an iterative solution but there is a very simple, elegant **recursive** solution.

Towers of Hanoi Legend

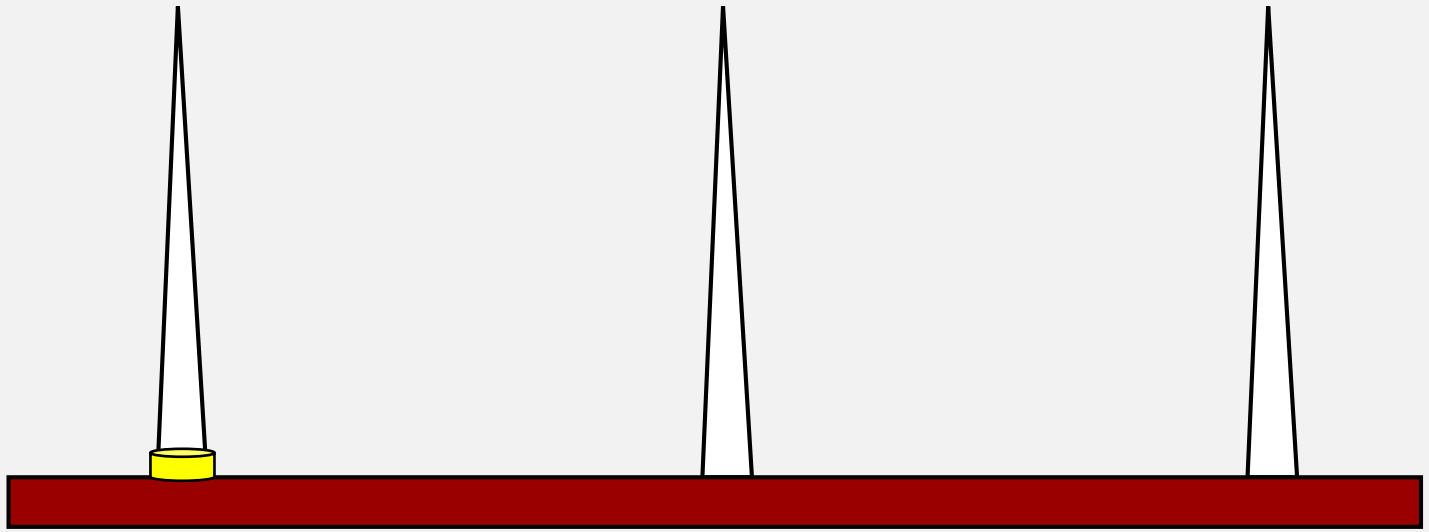


Source Auxiliary Destination

Problem Analysis: base case

Identify an instance of the problem that is trivial

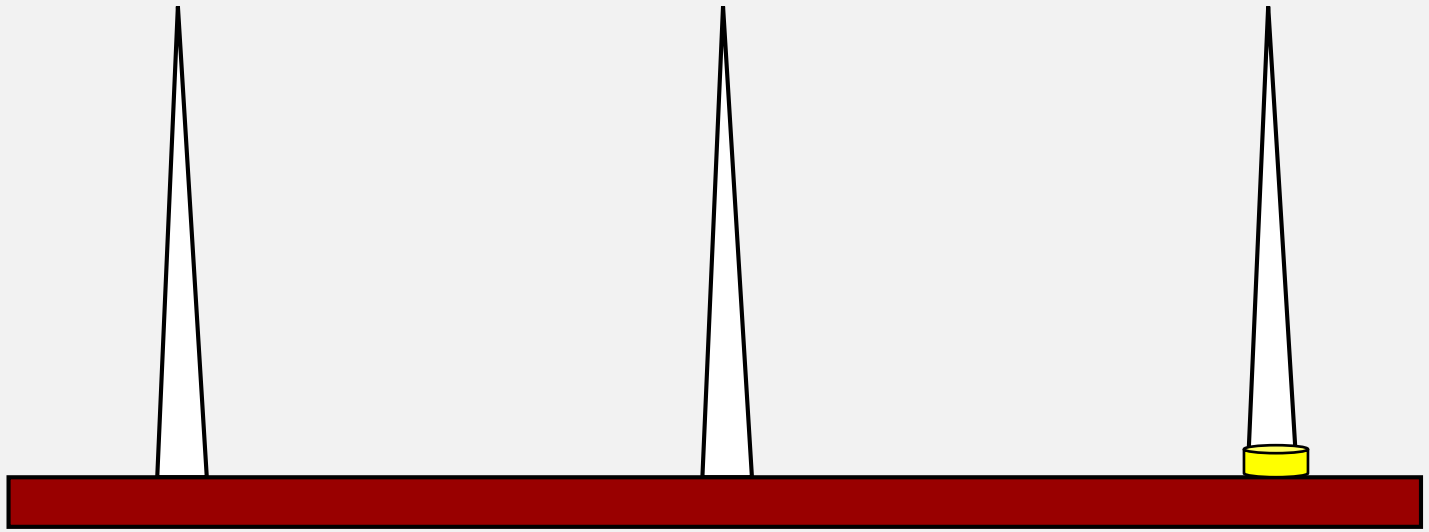
N=1



When this is true, the instruction is to move the top disk from the source to the target destination

Problem Analysis: base case

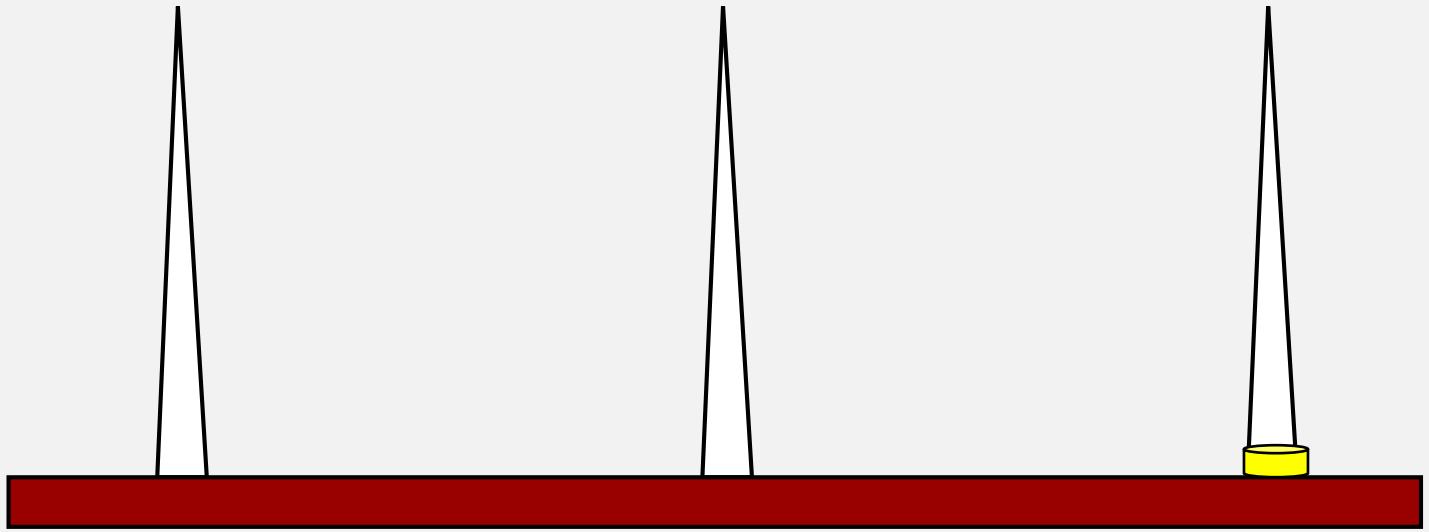
Move N from source to destination



When this is true, the instruction is to move the top disk from the source to the target destination

Problem Analysis: base case

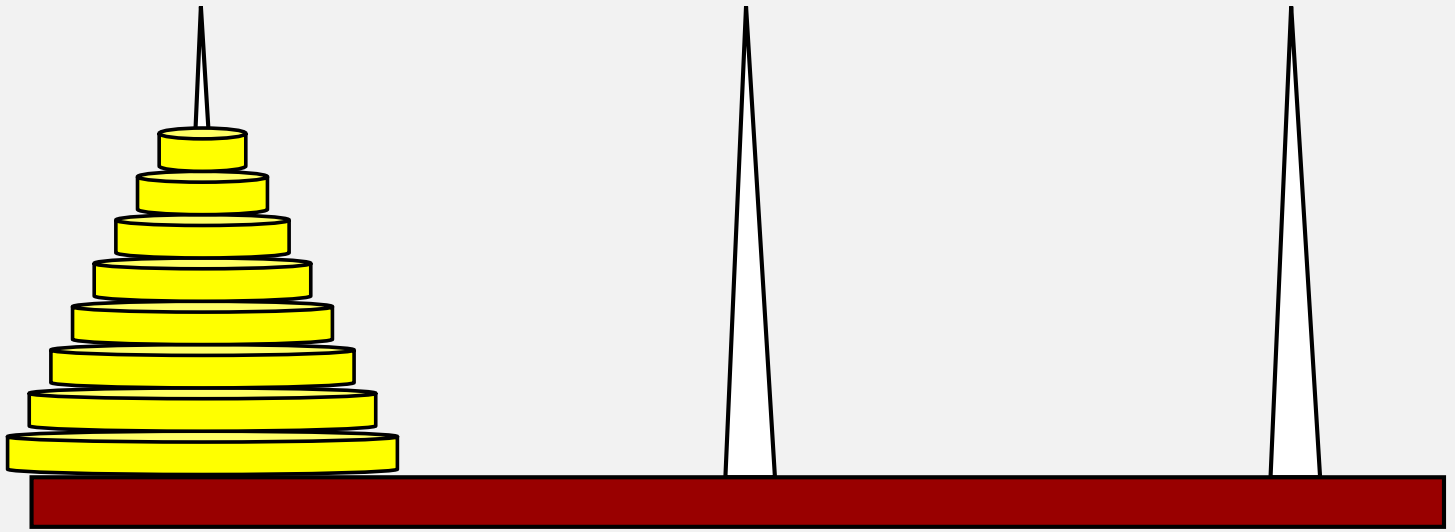
Move N from source to destination



When this is true, the instruction is to move the top disk from the source to the target destination

Problem Analysis: N > 1

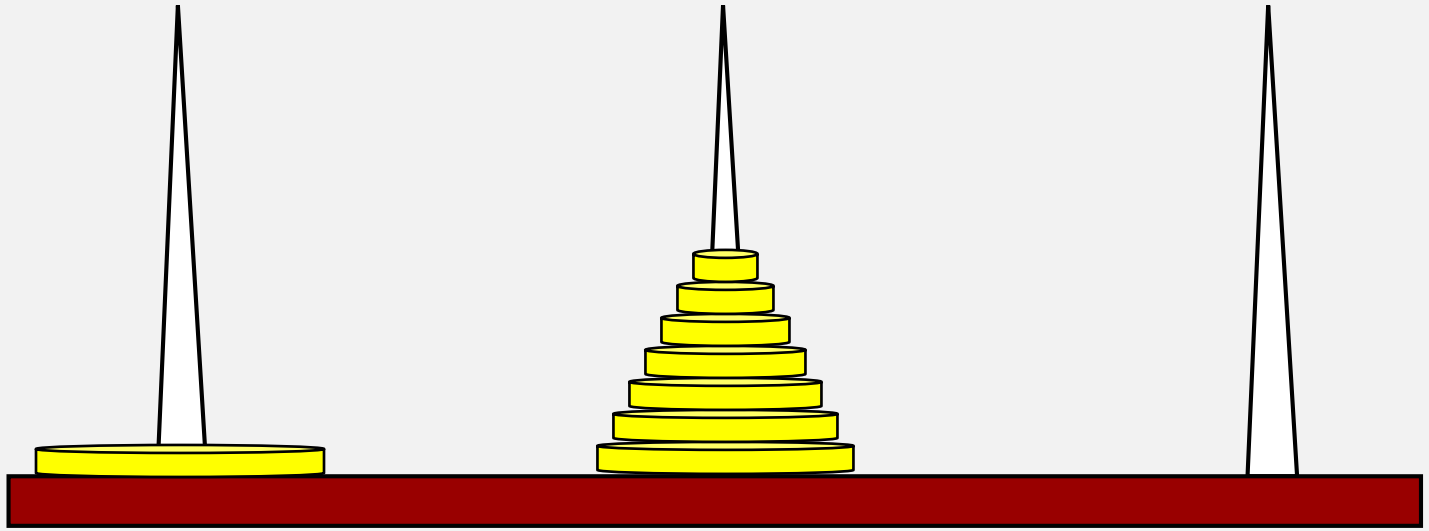
How can we use recursion?



Recursively move N-1 disks from our source to the auxiliary

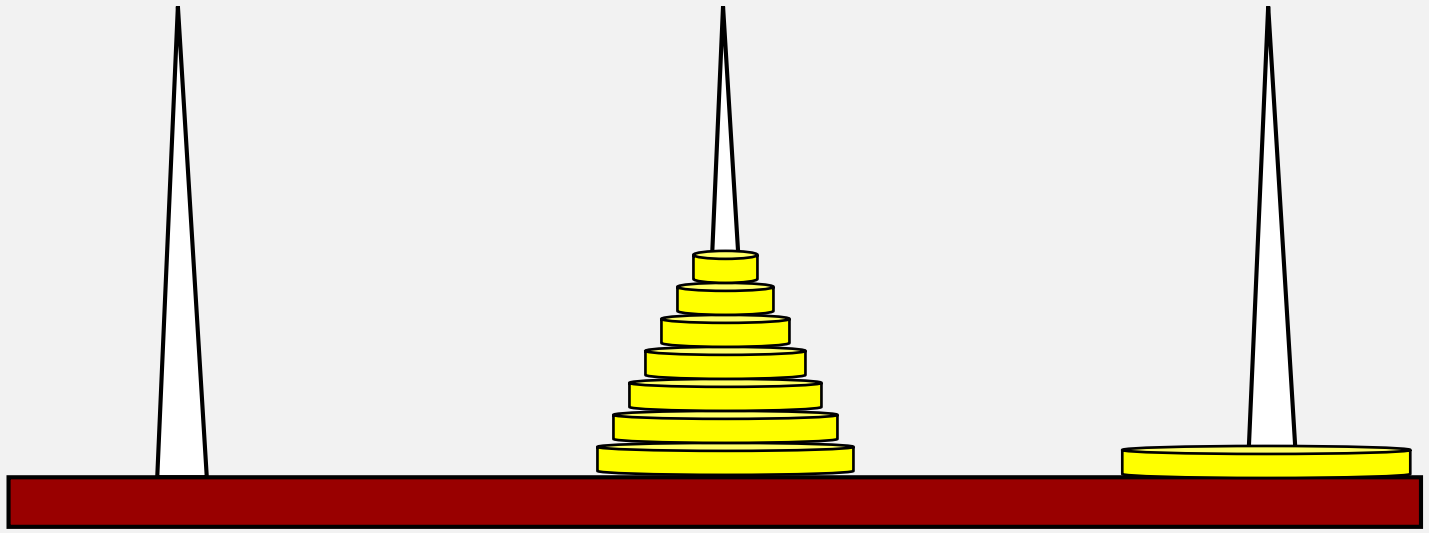
Problem Analysis: N > 1

How can we use recursion?



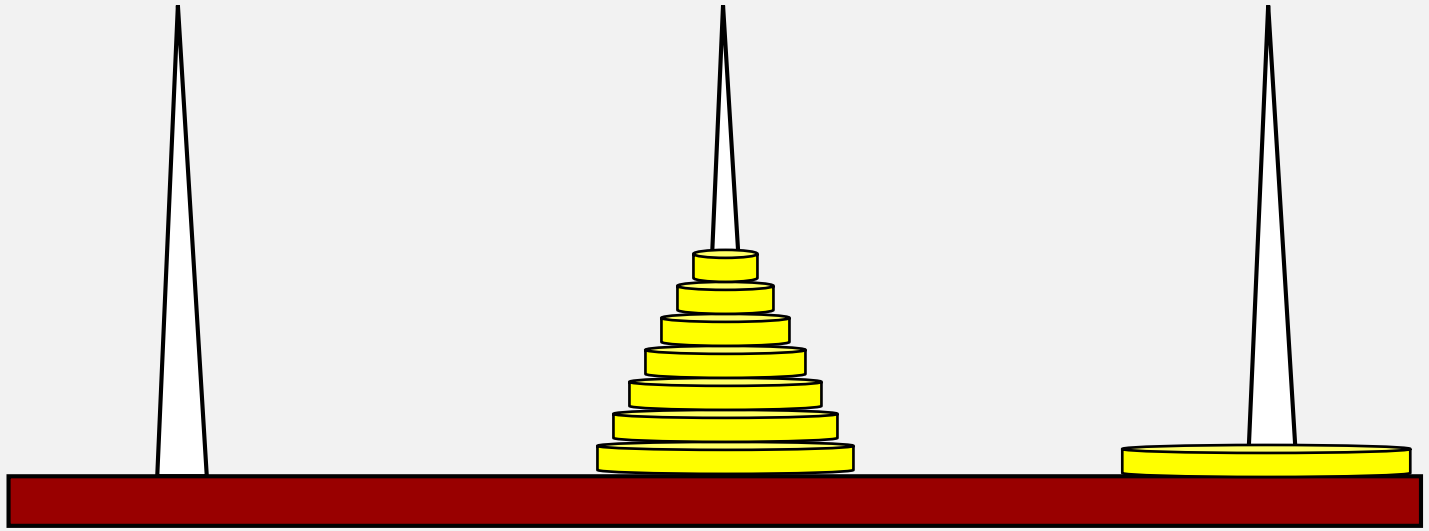
Recursively move N-1 disks from our source to the auxiliary

Problem Analysis: N > 1



Move the one remaining disk from src to dest

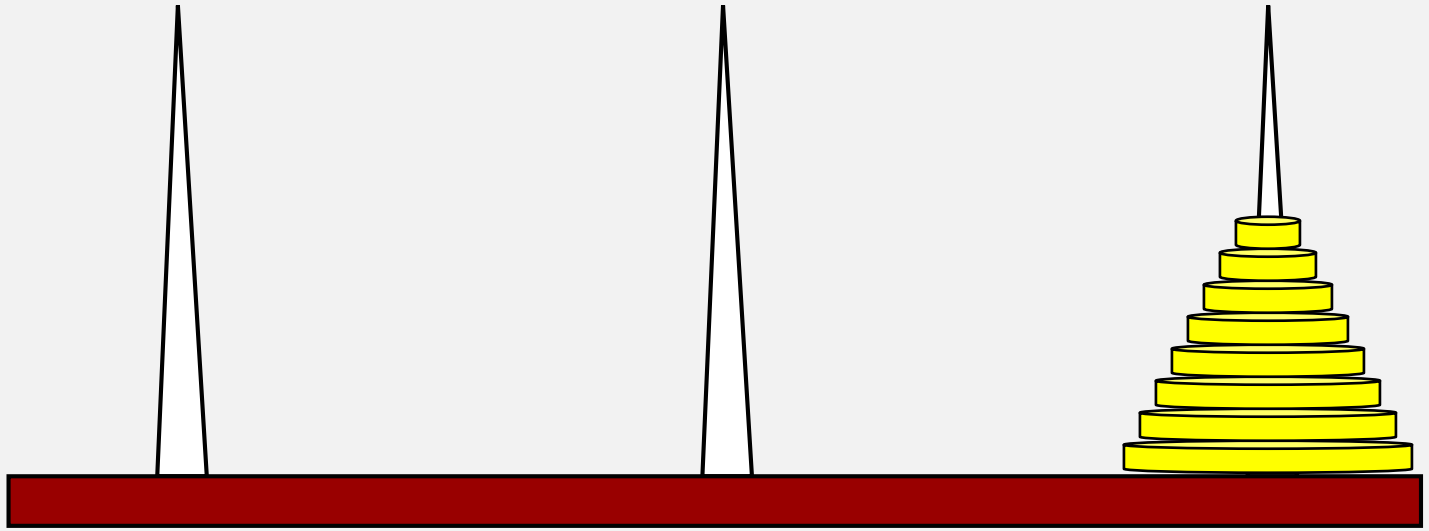
Problem Analysis: N > 1



Recursively move N-1 disks from auxiliary to destination

Problem Analysis: N > 1

We're done!

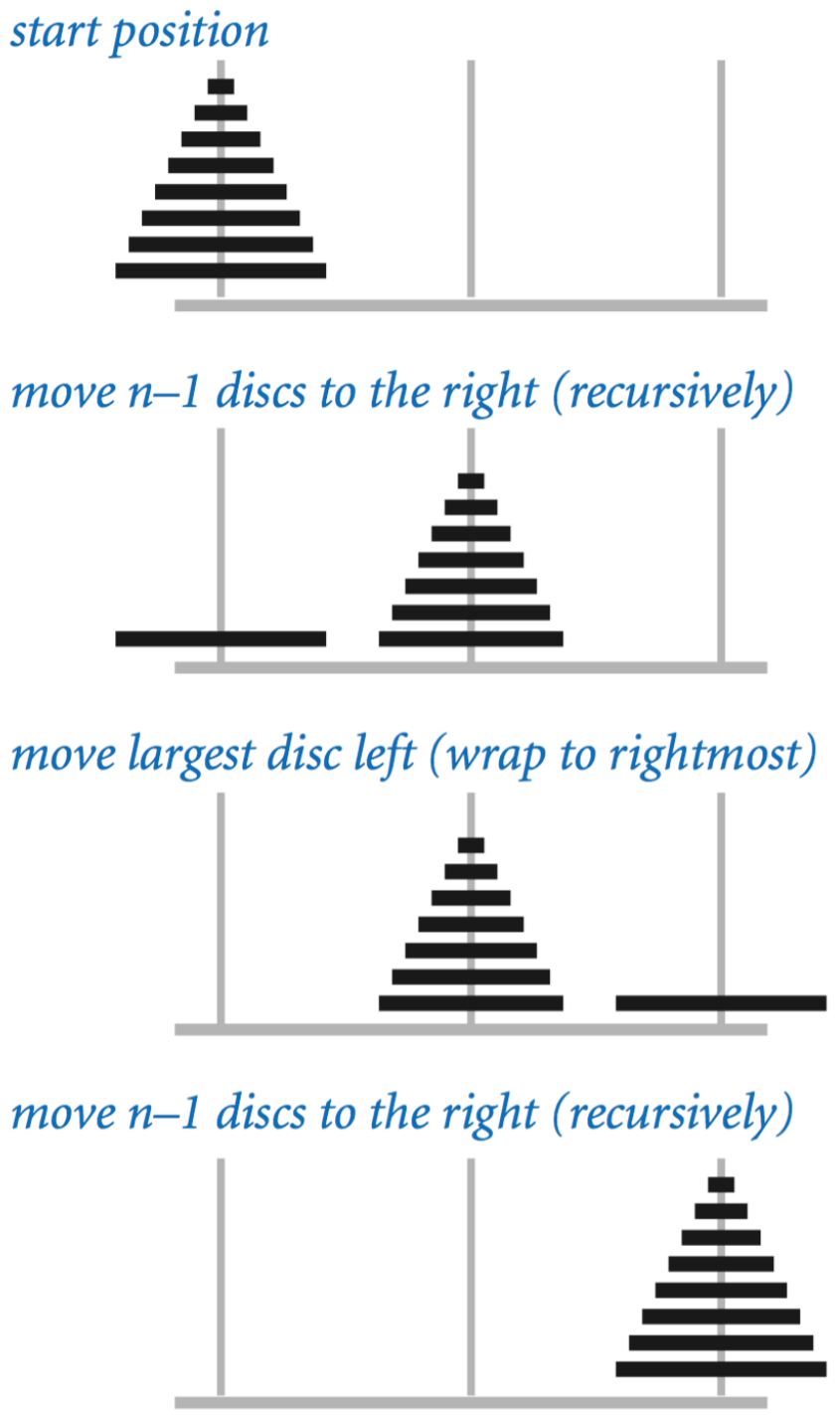


Recursively move N-1 disks from auxiliary to destination

Will the world end?

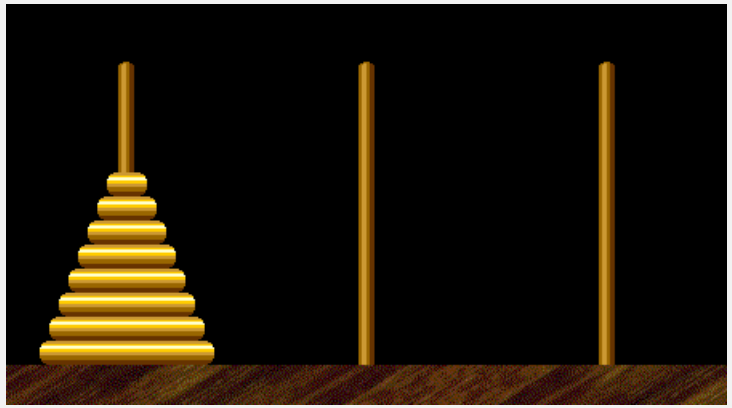
To solve a problem recursively means that you have to first redefine the problem in terms of a smaller subproblem of the same type as the original problem.

We first notice a pattern.

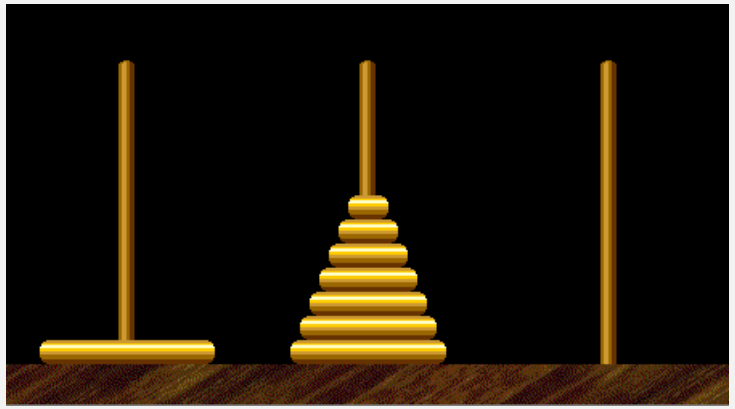


Will the world end?

We first notice a pattern.

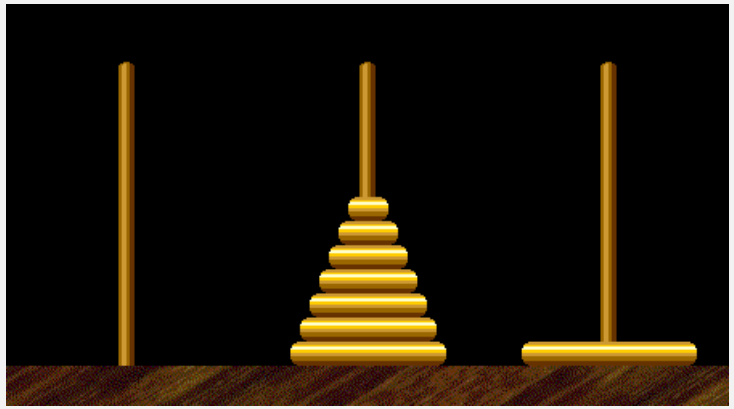


Move n-1 smallest discs right.

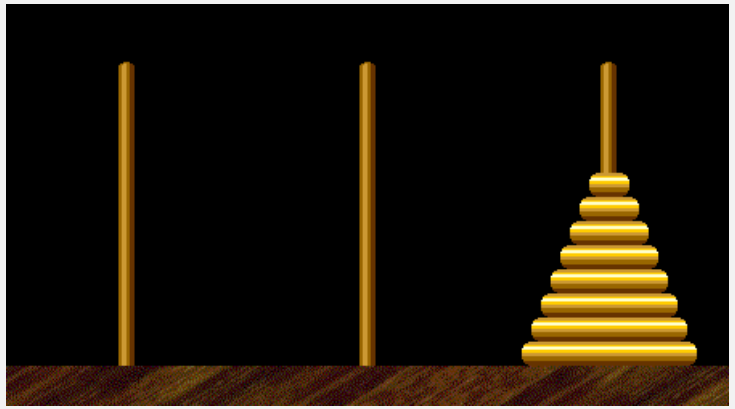


Move largest disc left.

cyclic wrap-around



Move n-1 smallest discs right.

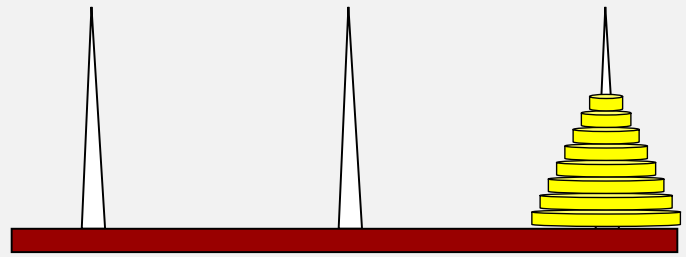


Our TowersOfHanoi Algorithm

We combine each step to create our algorithm for all N.

INPUT: N, Source, Destination, Auxiliary
Output: List of step-by-step moves to complete the challenge

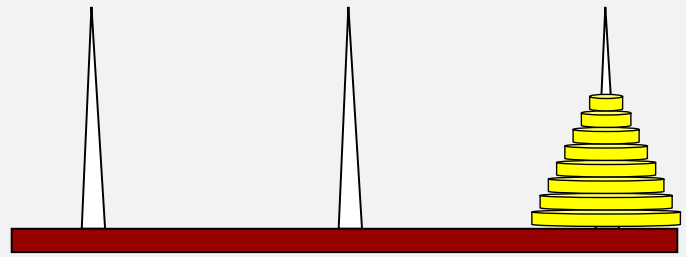
towersOfHanoi algorithm
    if n == 1:
    ##base case
        Move N from SRC to DEST
ELSE
        Move N-1 disks from Src to Aux
        Move N-1 disks from Aux to Dest



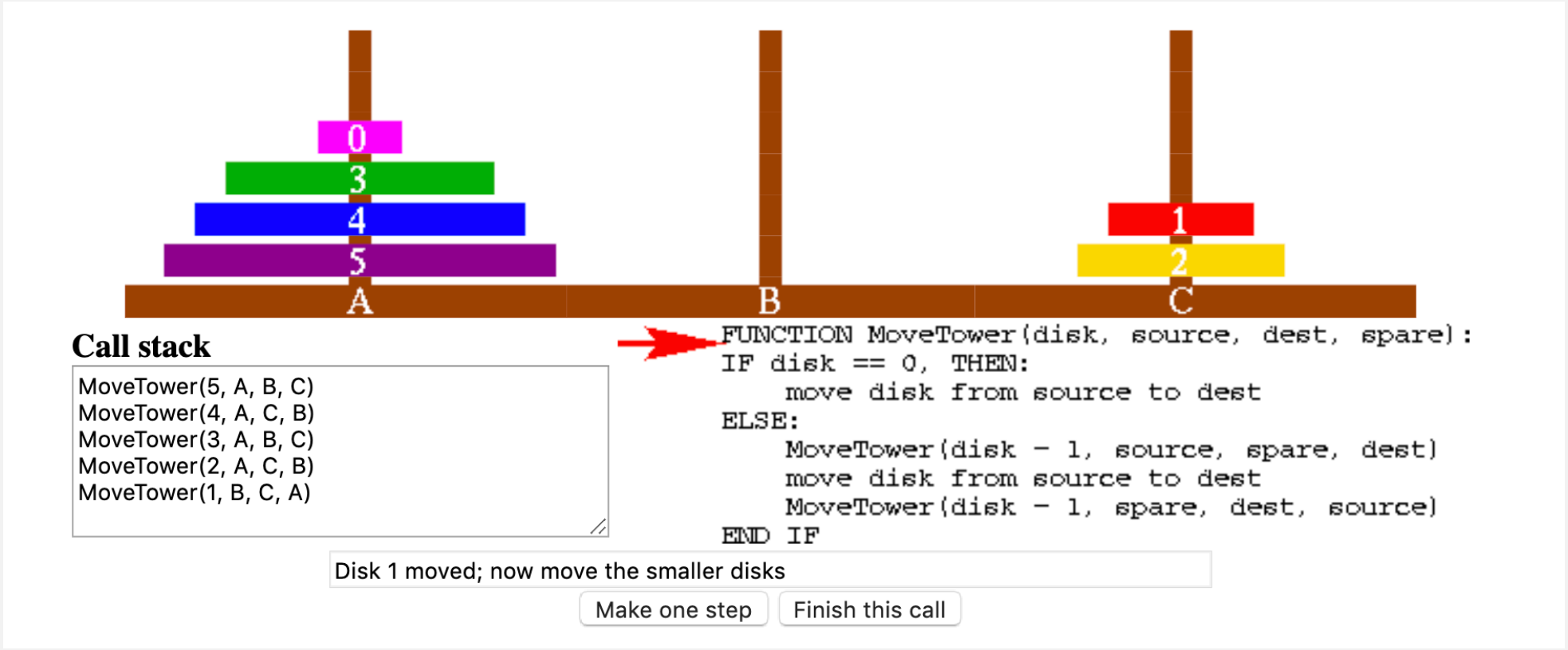
Our TowersOfHanoi Algorithm: Java

Let's code it up

```
public class Hanoi {
static void towersOfHanoi( int n, String source, String
                           destination, String auxiliary) {
if (n ==1) {
return;
}
towersOfHanoi( n-1, source, auxiliary, destination);
towersOfHanoi(n-1, auxiliary, destination, source);
}
}
```

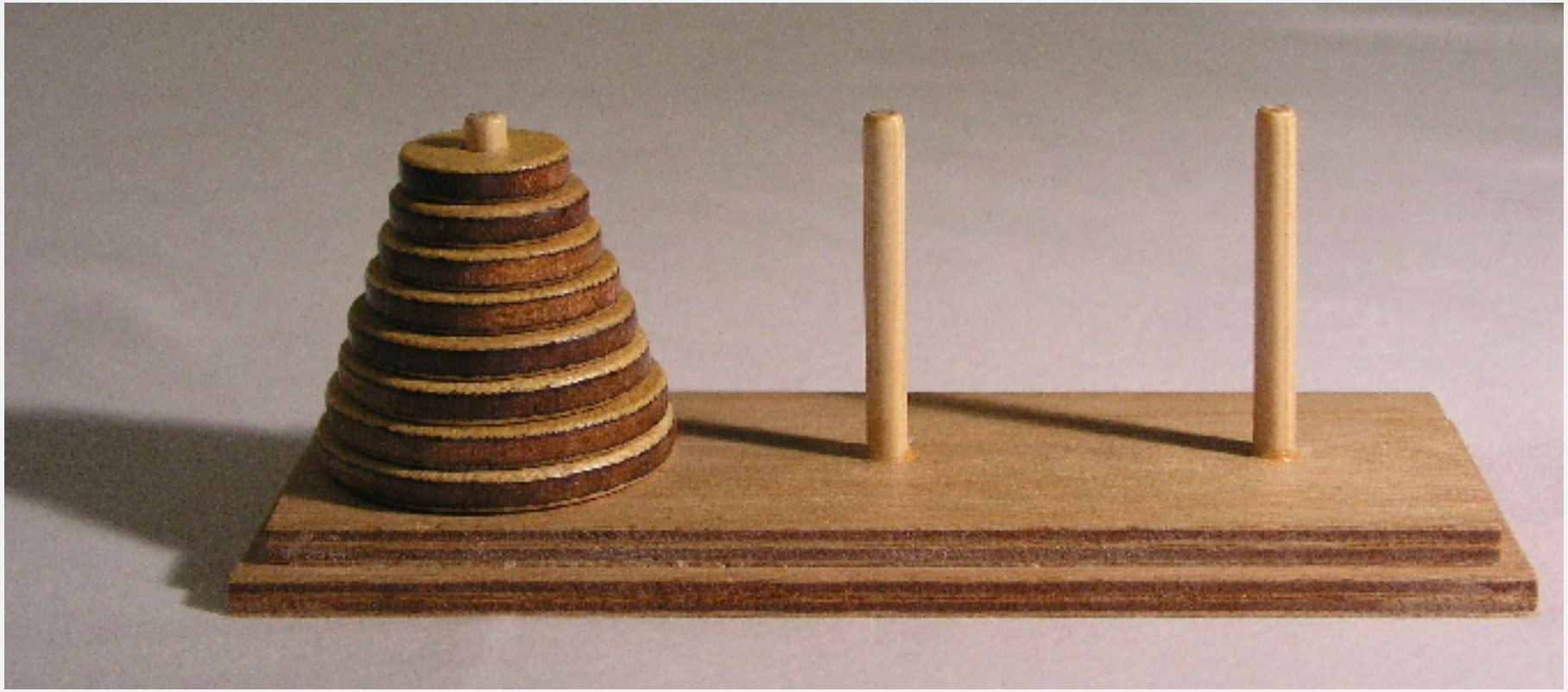


Play with this simulation to understand what's happening



https://www.cs.cmu.edu/~cburch/survey/recurse/hanoiex.html

Our TowersOfHanoi Algorithm: Java



Let's see if it works

Will the world end?

How many moves does it take to solve the problem as a function of the input, N.

Disks	Move
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023

Time Complexity: Exponential time = exponential growth!

Let T(n) be the number of move directives issued by our algorithm to move n discs from one peg to another.

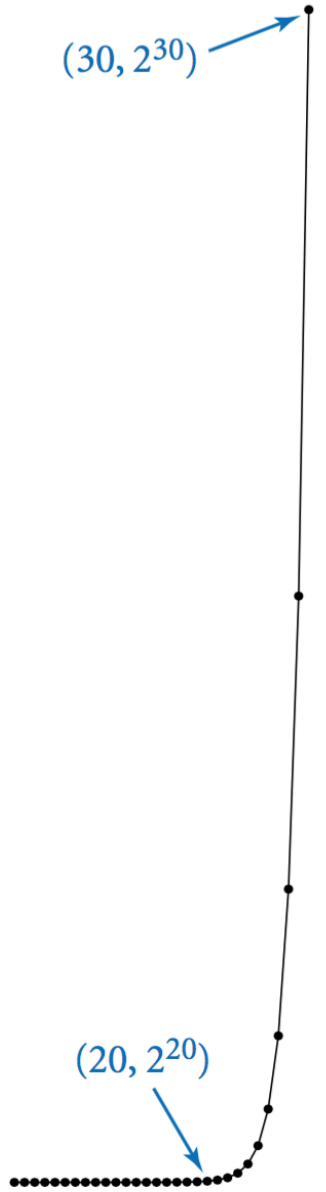
```
T(n)
```

We have two recursive calls (N-1) plus one constant operation (C)

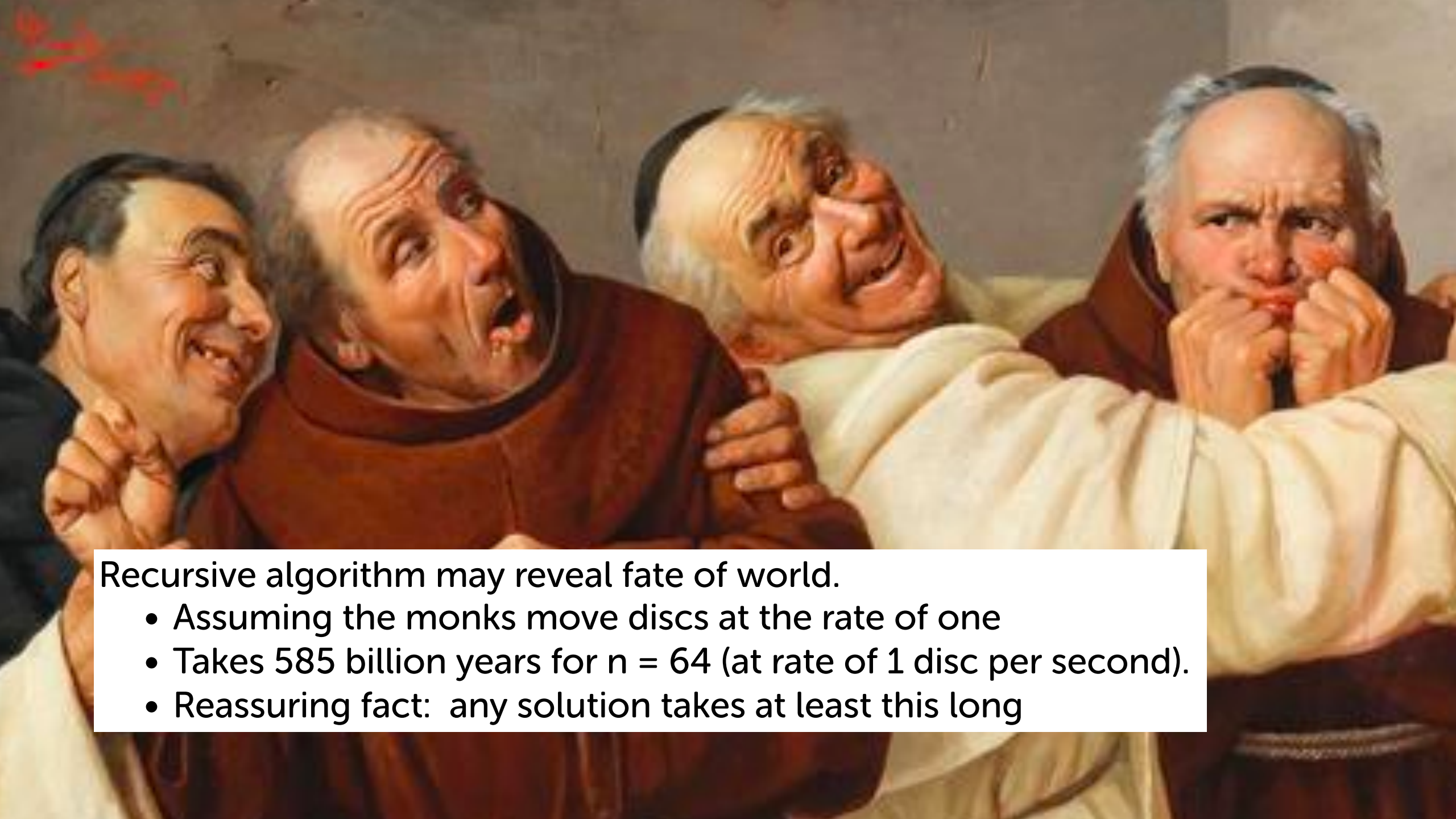
```
T(n) = 2T(n-1) + C
```

```
T(N) = O2^N - 1 or simply O^N
```

For a single increase in input the time needed is doubled!



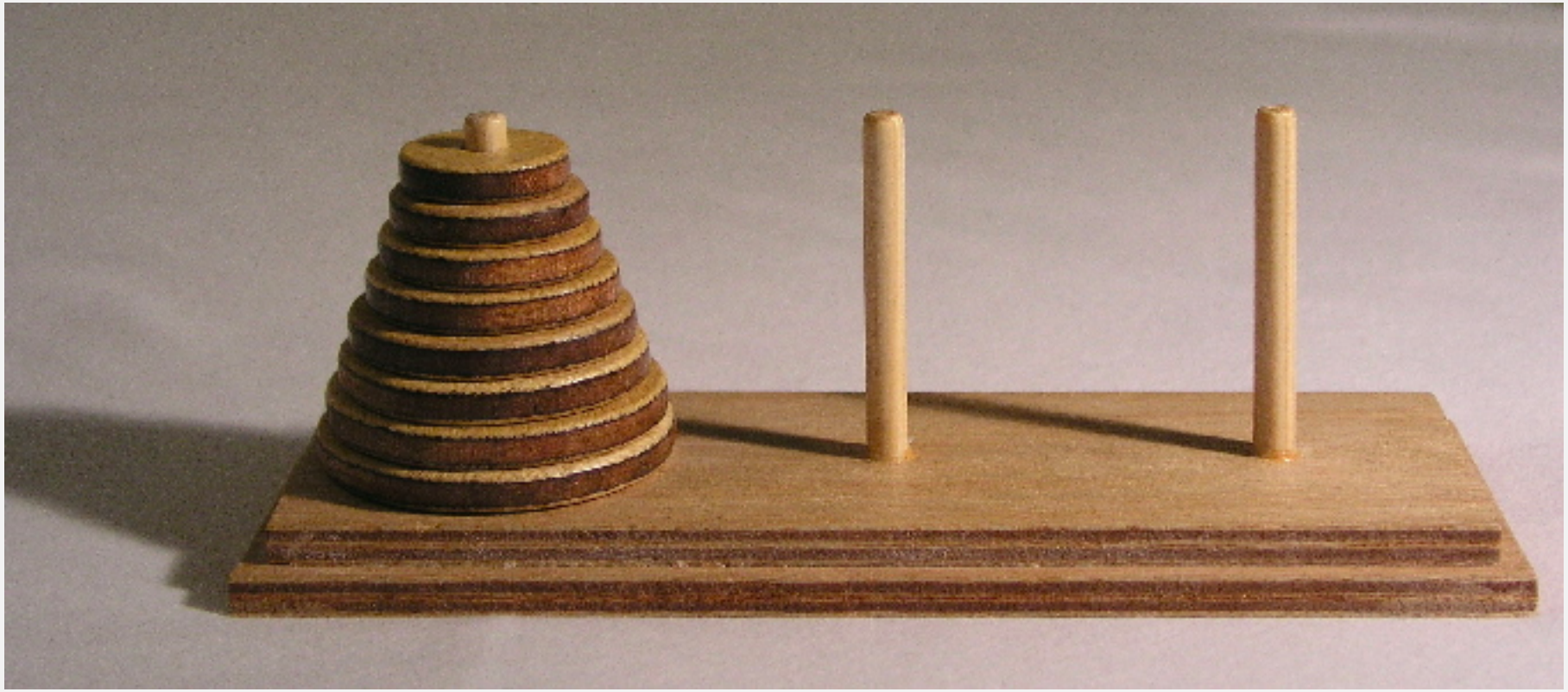
Exponential growth



Recursive algorithm may reveal fate of world.

- Assuming the monks move discs at the rate of one
- Takes 585 billion years for n = 64 (at rate of 1 disc per second).
- Reassuring fact: any solution takes at least this long

Our TowersOfHanoi Algorithm: Java



Challenge

History of Hanoi

The puzzle was invented by
the French mathematician Édouard Lucas in 1883.

Numerous myths regarding the ancient and mystical nature of the puzzle popped up almost immediately.

These myths are recounted in the monograph *The Tower of Hanoi—Myths and Maths*

Application of Hanoi?

http://en.wikipedia.org/wiki/Tower_of_Hanoi#Applications

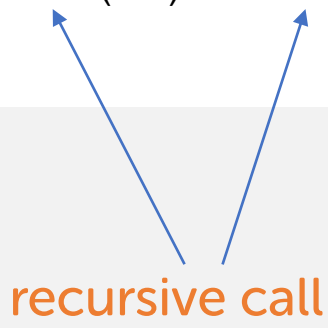
- Backing up computer data
- Psychological research for problem solving assessment
- Teaching recursion to undergraduates!

Types of Recursion

Multiple Recursion

- when calling the algorithm can cause more than one recursive call to the very same algorithm
- Examples:
  - Recursive method for computing the n-th Fibonacci number

```
public static long fibonacci(long n) {
  if (n < 0) return -1;  // F(n) is not defined when n is negative
  if (n == 0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return fibonacci(n-1) + fibonacci(n-2);
}
```



Types of Recursion

Multiple Recursion

- when calling the algorithm can cause more than one recursive call to the very same algorithm
- Risk of exponential growth
- Examples:
  - Recursive method for computing the n-th Fibonacci number
  - Tower of Hanoi

recursive call

```
public class Hanoi {
static void towersOfHanoi( int n, String source, String
                           destination, String auxiliary) {
if (n ==1) {
return;
}
towersOfHanoi( n-1, source, auxiliary, destination);
towersOfHanoi(n-1, auxiliary, destination, source);
}
}
```

Types of Recursion: Other flavours?

Tail Recursion

- When the recursive step comes last in your algorithm
- Check for the base case first
- Most common form

```
FUNCTION countDown(number){
     IF( number == 0 ):
          RETURN 0
     END IF

     RETURN countDown(number – 1)
END FUNCTION
```

Head Recursion

- When recursive call comes first
- Before any other processing
- Base case resolves afterward

```
FUNCTION countDown(number):
     IF( n > 0 ):
          RETURN countDown( number – 1 )
     END IF

     RETURN 0
END FUNCTION
```

Divide & Conquer

> Divide et impera. Veni, vidi, vici. - Julius Caesar

Divide-and-conquer paradigm.

- Break up problem into smaller subproblems of same structure.
- Solve subproblems recursively using same method.
- Combine results to produce solution to original problem.

Many important problems succumb to divide-and-conquer.

- FFT for signal processing.
- Parsers for programming languages.
- Multigrid methods for solving PDEs.
- Quicksort and mergesort for sorting.
- Hilbert curve for domain decomposition.
- Quad-tree for efficient N-body simulation.
- Midpoint displacement method for fractional Brownian motion.

Recursion Recap

How to write simple recursive programs?

- Base case, reduction step.
- Trace the execution of a recursive program.
- Break problem down into smaller sub-problems
- More than one recursive call sometimes needed

Why learn recursion?

- New mode of thinking
- Powerful programming tool

Divide-and-conquer. Elegant solution for many important problems.

Some problems while solvable are inherently not feasible to compute.