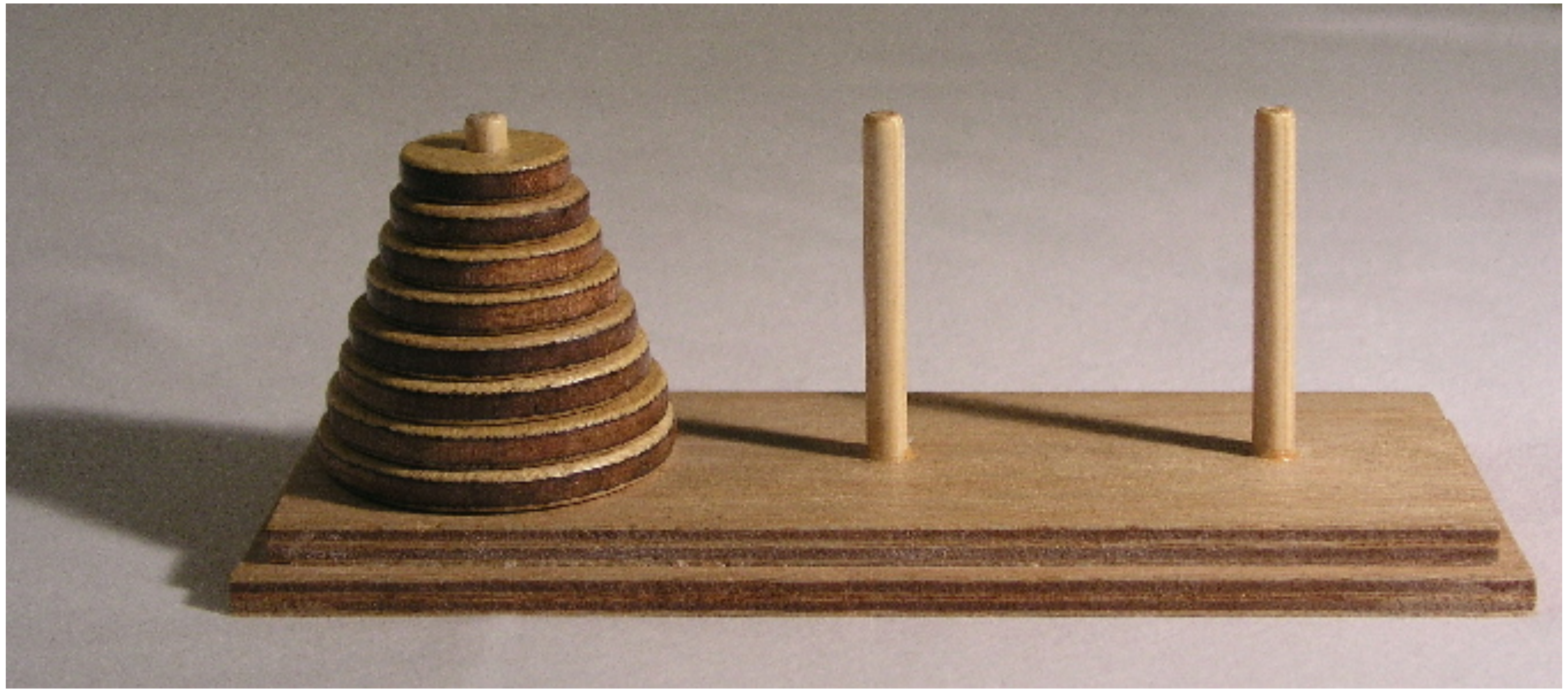
COL100: Introduction to Computer Science

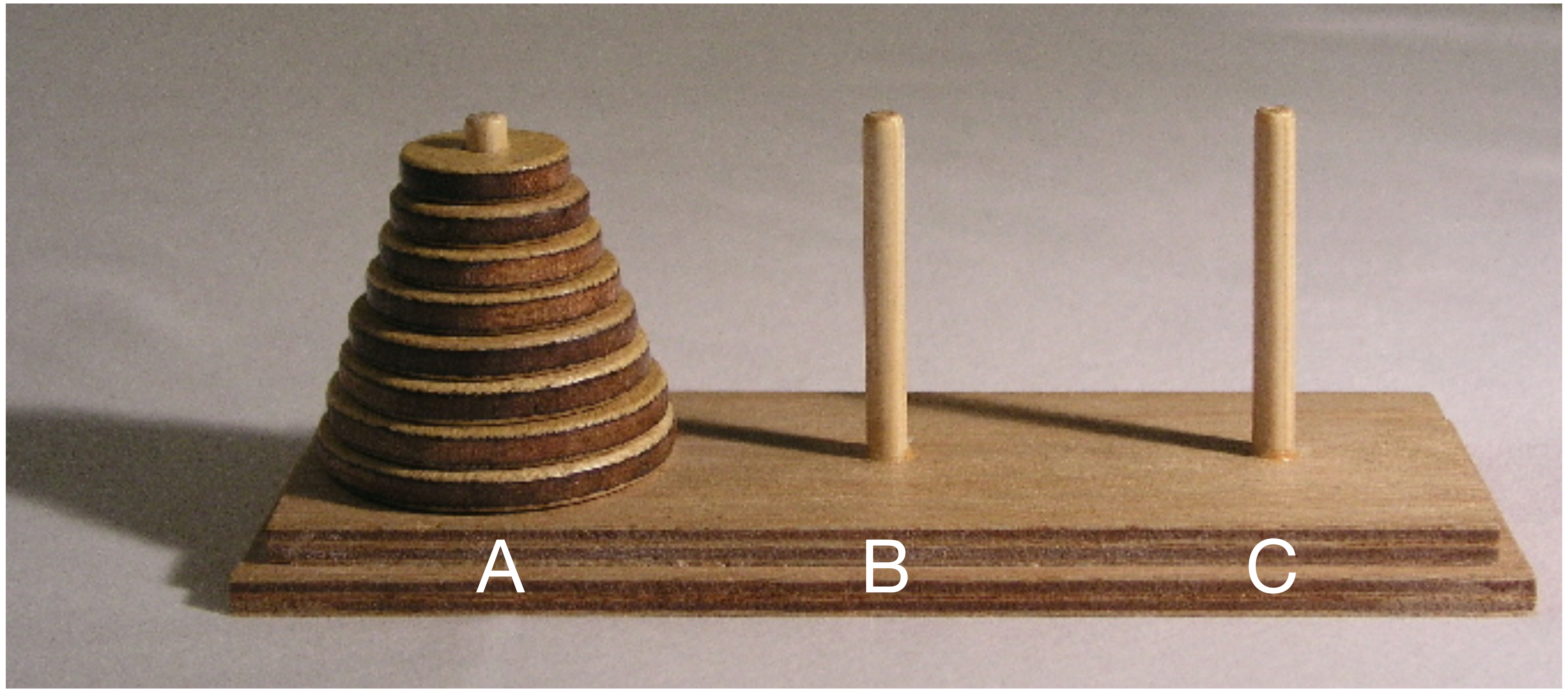
7.2: Some more algorithms (which are exponential time)

Tower of Hanoi problem

Some simple-sounding problems require extremely expensive algorithms.



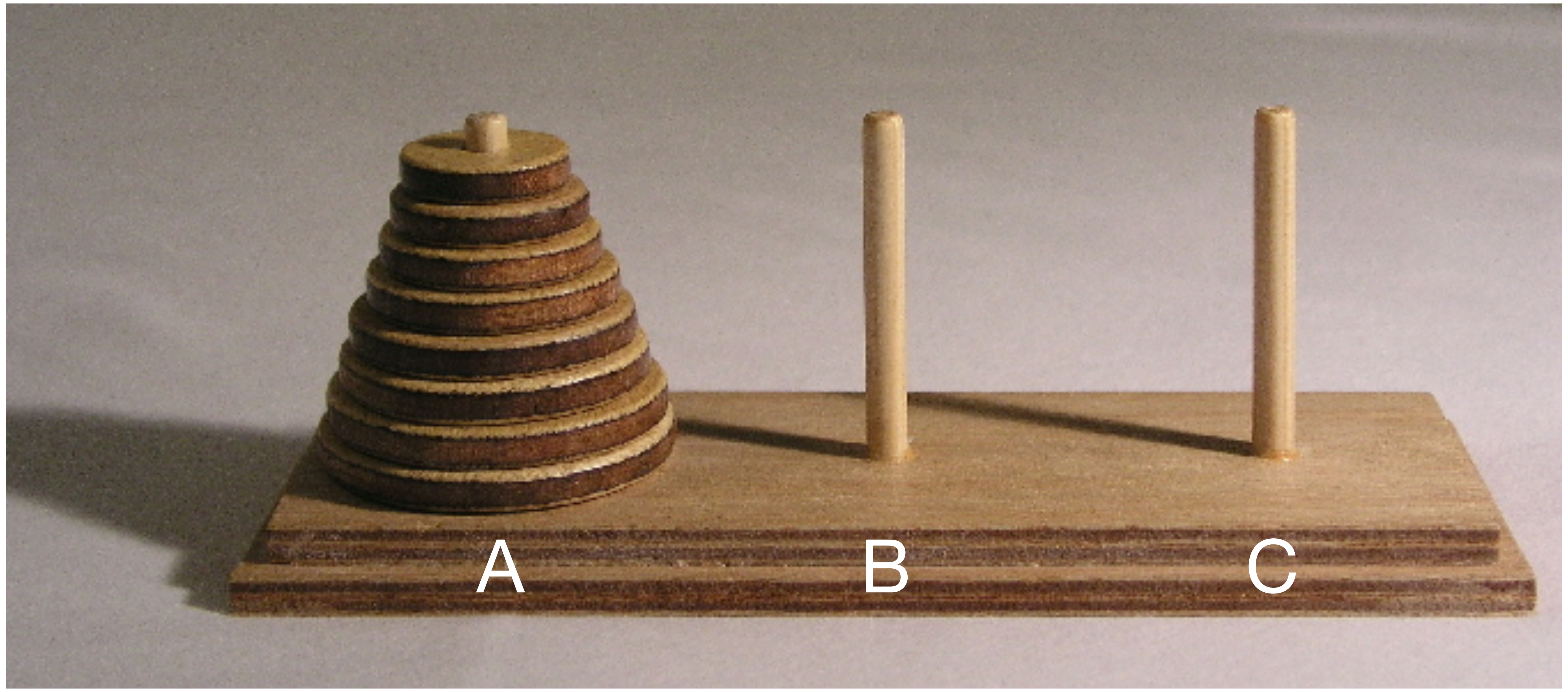
Problem: Move all the disks from the first peg to the third peg, but *without* ever placing a larger disk on top of a smaller disk.



Try it out by hand for, say, 4 disks.

Is there a systematic approach? If so, is it easy to describe?

Hint: You need to move n disks from peg A to peg C, using peg B as a spare.
Can you do this by recursively moving n–1 disks from one peg to another?



Move n–1 disks from peg A to peg B, using peg C as a spare.

Move 1 disk from peg A to peg C.

Move n–1 disks from peg B to peg C, using peg A as a spare.

To move n disks from *src* to *dst* using *aux*,

If $n = 0$, then

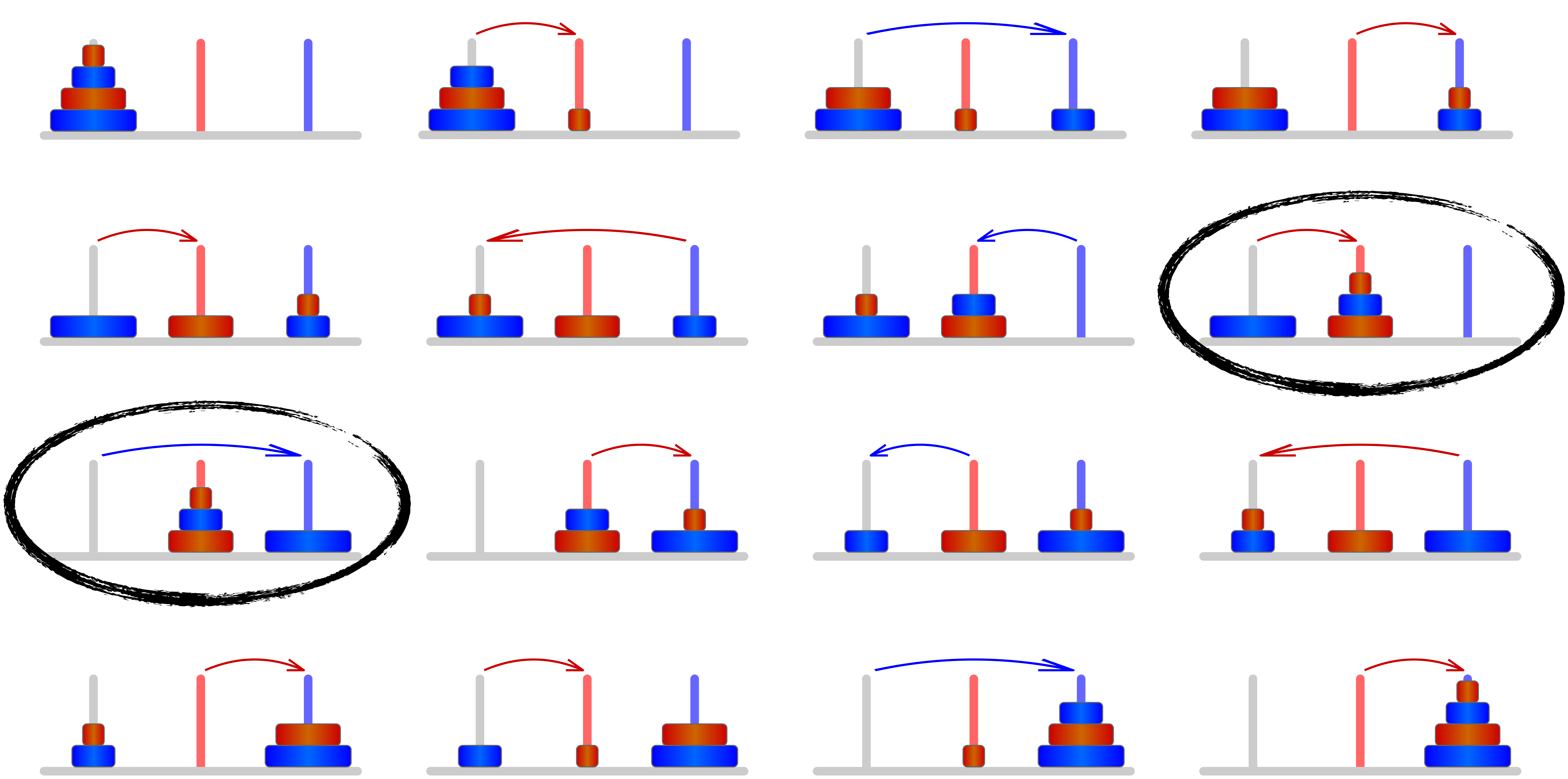
Do nothing.

Otherwise,

Move $n-1$ disks from *src* to *aux* using *dst*.

Move 1 disk from *src* to *dst*.

Move $n-1$ disks from *aux* to *dst* using *src*.



What is the time complexity, if moving 1 disk takes 1 unit of time?

$$T(0) = 0$$

$$\begin{aligned} T(n) &= T(n-1) + 1 + T(n-1) \\ &= 2T(n-1) + 1 \end{aligned}$$

If we had a guess for a formula for $T(n)$, we could try proving it via induction. But first we need a guess.

$$T(0) = 0, \quad T(1) = 1, \quad T(2) = 3, \quad T(3) = 7, \quad T(4) = 15, \quad \ldots$$

Guess: $T(n) = 2^n - 1 = O(2^n)$. Prove it via induction.

The subset sum problem

Consider a version of the change-making problem:

I have n coins/notes of various denominations in my pocket. Can I make exact change for a given amount a?

$d(1) = 5$
$d(2) = 8$
$d(3) = 12$
$d(4) = 20$
$d(5) = 31$
$d(6) = 33$

$a = 48$

$$c : \mathbb{N} \times (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \to \mathbb{B}$$

$c(a, d, n)$ is true if and only if I can make change for amount a using only (some subset of) the notes $d(1)$, $d(2)$, …, $d(n)$.

$$c(a, d, n) = \begin{cases} a = 0 & \text{if } n = 0, \\ c(a, d, n-1) \vee c(a - d(n), d, n-1) & \text{otherwise.} \end{cases}$$

If $n > 0$, either I make change without using the nth note, or I use it and make change for the remaining $a - d(n)$ amount using the rest of the notes.

$$c(a, d, n) = \begin{cases} a = 0 & \text{if } n = 0, \\ c(a, d, n-1) \vee c(a - d(n), d, n-1) & \text{otherwise.} \end{cases}$$

Let $T(n)$ = number of subtractions needed to evaluate $c(a, d, n)$.

$$T(0) = 0$$

$$T(n) = 2T(n - 1) + 3$$

Then $T(n) = 3(2^n - 1) = O(2^n)$.

What is this algorithm really doing?

We are given a set $D = \{d_1, d_2, \ldots, d_n\}$ and we want to search for a subset $S \subseteq D$ such that $\sum S = a$.

- In case $d_n \notin S$, we need to search for $S \subseteq \{d_1, d_2, \ldots, d_{n-1}\}$.
- In case $d_n \in S$, we need to search for $S' = S \setminus \{d_n\} \subseteq \{d_1, d_2, \ldots, d_{n-1}\}$ such that $\sum S' = a - d_n$.

We are exhaustively checking all subsets of D, and there are 2^n of them.

Is a polynomial-time algorithm possible? *Nobody knows!**

*The time complexity should be polynomial in n and in the total number of digits in a and d_i.
Look up "subset sum problem" and "P vs. NP" if you want to learn more about this open problem.

Next lecture

- Recorded lectures by Prof. Huzur Saran
- Tail recursion and iterative processes
    - *factorial*(n) in $O(1)$ space
    - *fibonacci*(n) in $O(n)$ time and $O(1)$ space