1. When someone points to you and says “Go!”:
   (a) Point to the person on your left
   (b) Say “Go!”

2. When the person on your left says “Done!”:
   (a) Move your box
   (b) Point to the person on your left
   (c) Say “Go!”

3. When the person on your left says “Done!” again:
   (a) Turn to the person on your right
   (b) Say “Done!”

How to transfer a stack of $n$ boxes from *orig* to *dest* (using *temp*):

If $n = 0$, there's nothing to do. Relax.

Otherwise, do the following:

Transfer the top $n - 1$ boxes from *orig* to *temp* (using *dest*).

Move box $n$ from *orig* to *dest*.

Transfer the top $n - 1$ boxes from *temp* to *dest* (using *orig*).

TRANSFER($n$, *orig*, *dest*, *temp*):

if $n > 0$

TRANSFER($n - 1$, *orig*, *temp*, *dest*)

Move box $n$ from *orig* to *dest*.

TRANSFER($n - 1$, *temp*, *dest*, *orig*)


Édouard Lucas (1883)

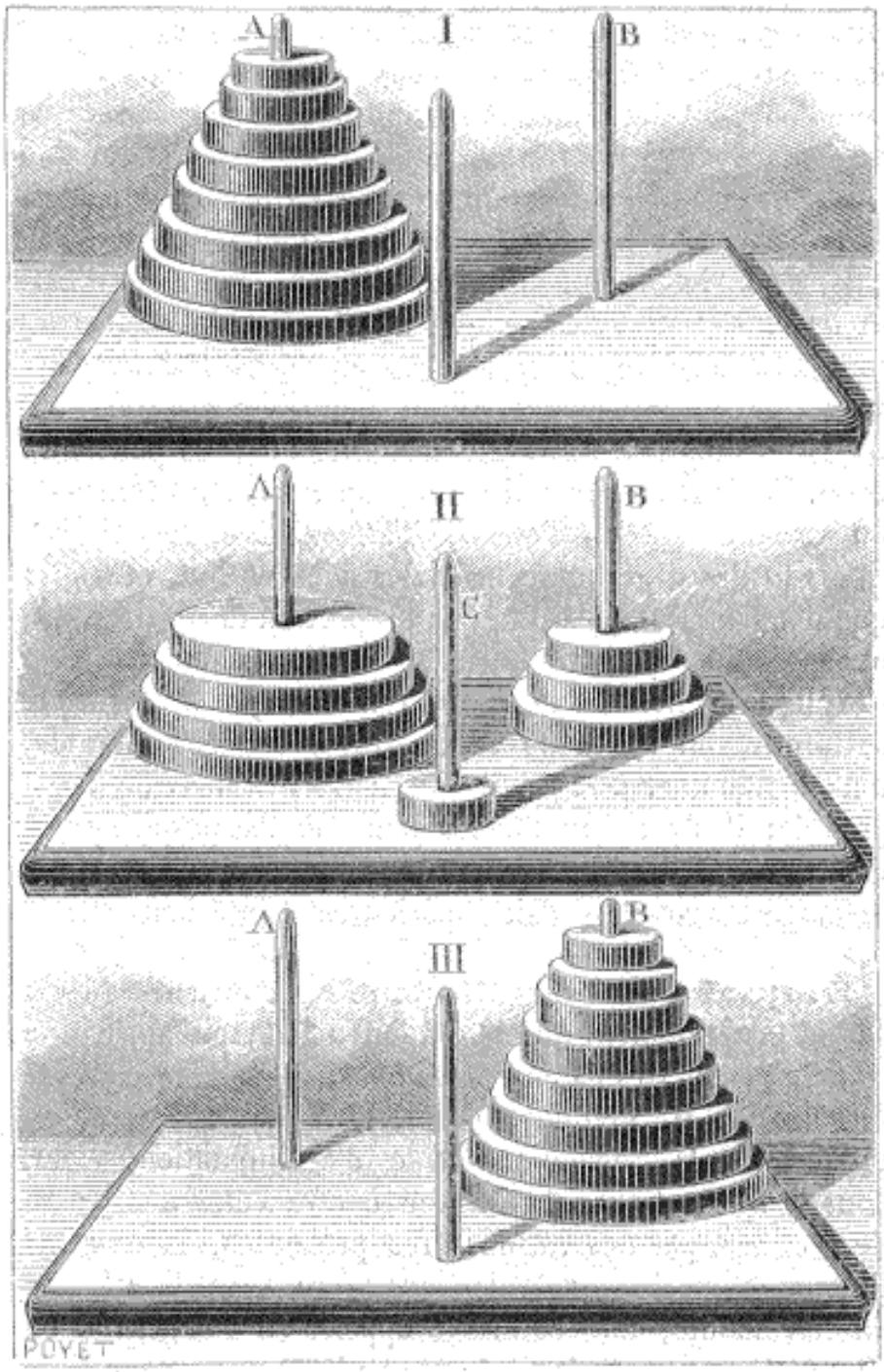


Fig. 1. — Jeu de la Tour d'Hanoï.

1. Commencement de la partie; la tour est construite en A. — II. Partie en voie d'exécution; les disques sont placés successivement sur les tiges A, B, C, par ordre décroissant. — III. Fin de la partie; la tour est reconstruite en B.

Henri de Parville, *La Nature* (1884)

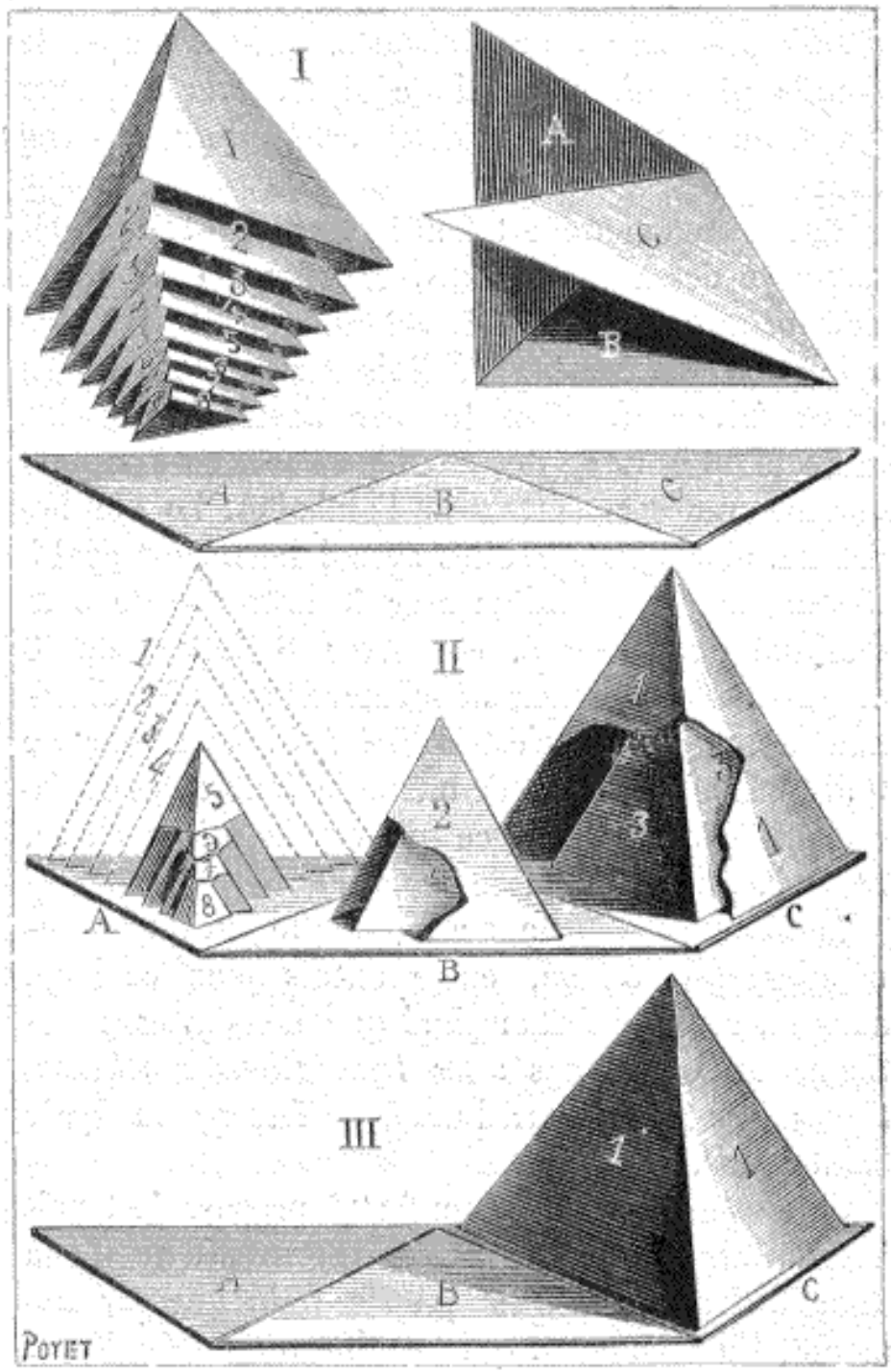


Fig. 2. — Jeu de la Question du Tonkin.

I. Pyramides de cartons, décroissantes 1 à 8, avec leur support ABC. — II. Partie en voie d'execution : figure montrant la superposition des pyramides que l'on doit faire passer de A en B et en C. — III. Fin de la partie ; les pyramides sont reconstruites en C.

Henri de Parville, *La Nature* (1884)

# The Tower of Hanoi

M. De Parville gives the following story of a remarkable puzzle. In the great temple at Benares, says he, beneath the dome which marks the center of the world, rests a plate of brass in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles at the creation was placed sixty-four discs of pure gold, the largest disc resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the tower of Bramah. Day and night unceasingly the priests transfer the disc from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle on which they were placed at the creation to one of the other needles, tower, temple and Brahmins alike will crumble into dust, and with a thunder-clap the world will vanish!

The number of separate transfers of single discs which the Brahmins must make to effect the transfer of the tower is $2^{64} - 1$, that is 18,446,744,073,709,551,615, a number which, even if the priest made no mistakes and could make one transfer per second, would require many thousands

of millions of years to carry out! As our puzzlist could not afford to spare the time to solve such a complicated puzzle, we give them just thirteen discs from the top of the tower and ask in how many transfers can the change now be made? The discs are in one pile, and you are allowed two other places to build on, but are never to place a larger disc above a smaller one.

Sam Loyd (1914)
http://www.mathpuzzle.com/loyd/

## ***How many moves?***

```
TRANSFER(n, orig, dest, temp):
  if n > 0
      TRANSFER(n − 1, orig, temp, dest)
      Move box n from orig to dest.
      TRANSFER(n − 1, temp, dest, orig)
```

$T(n) =$ number of moves required to transfer $n$ boxes (using TRANSFER)

The algorithm gives us a ***recurrence*** for $T(n)$:

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ T(n-1) \ + \ 1 \ + \ T(n-1) & \text{if } n \geq 0 \end{cases}$$

$T(0) =$ ☐

$T(1) =$ ☐

$T(2) =$ ☐

$T(3) =$ ☐

$T(4) =$ ☐

$T(5) =$ ☐

$T(6) =$ ☐

So what do you think $T(n)$ is?

$$\boxed{T(n) = \begin{cases} 0 & \text{if } n = 0 \\ T(n-1) + 1 + T(n-1) & \text{if } n \geq 0 \end{cases}}$$

**Theorem:** $T(n) = 2^n - 1$ *for all integers* $n \geq 0$.

**Proof:** Let $n$ be an arbitrary non-negative integer.

Suppose $T(k) = 2^k - 1$ for all non-negative integers $k \leq n$.

There are two cases to consider: $n = 0$ and $n \geq 1$.

- If $n = 0$, then $T(n) = 0$ and $2^n - 1 = 2^0 - 1 = 0$.
- Suppose $n \geq 1$.

$$\begin{aligned} T(n) &= 2T(n-1) + 1 && \text{[recurrence]} \\ &= 2(2^{n-1} - 1) + 1 && \text{[induction hypothesis]} \\ &= 2^n - 1 && \text{[algebra]} \end{aligned}$$

In both cases, we conclude that $T(n) = 2^n - 1$. □

We'll learn how to solve recurrences (*without guessing*!) after the break.

## *Recursive Algorithms*

- An algorithm/program/function/procedure/method is ***recursive*** if it calls *itself* as a subroutine.

- If the problem is small/simple enough, just solve it directly.

  Otherwise, ***divide and conquer***:

  - Reduce the problem to one or more *smaller/simpler* subproblems.
  - Solve each subproblem recursively.
  - Combine the subsolutions into the final solution.

  Just like induction!

- Proof of correctness by induction:

  - Base case(s) in proof = direct parts of algorithm
  - Inductive case(s) in proof = recursive parts of algorithm

- Running time computed by setting up and solving a ***recurrence***:

  - Base case(s) of recurrence = direct parts of algorithm
  - Recursive case(s) of recurrence = recursive parts of algorithm

## ***Analyzing Algorithms***

- Most instructions take $\Theta(1)$ time.

  addition, subtraction, multiplication, division, comparisons, assignments, logical operations, array lookups, pointer traversals, memory allocation[1]

- Loops become sums:

$$\boxed{\begin{array}{l}\text{for } i \leftarrow 1 \text{ to } n \\ \quad \textbf{\textit{Something taking } f(i) \textit{ time}}\end{array}} \implies \sum_{i=1}^{n} f(i) \; + \; \Theta(n)$$

- Subroutine calls become functions:

$$\boxed{\begin{array}{l}\textsc{Foo}(n/2) \\ \text{for } i \leftarrow 1 \text{ to } n \\ \quad \textsc{Bar}(i) \\ \textsc{Foo}(n/2)\end{array}} \implies 2 \cdot T_{\textsc{Foo}}(n/2) + \sum_{i=1}^{n} T_{\textsc{Bar}}(i) + \Theta(n)$$

- Recursive calls become recurrences:

$$\boxed{\begin{array}{l}\underline{\textsc{Squee}(n)\text{:}} \\ \quad \text{for } i \leftarrow 1 \text{ to } n-1 \\ \quad\quad \textsc{Squee}(i)\end{array}} \implies T_{\textsc{Squee}}(n) = \sum_{i=1}^{n} T_{\textsc{Squee}}(i) + \Theta(n)$$

[1] Dynamic memory management requires some nontrivial work behind the scenes, but in practice, we can *usually* pretend it's free. However, dealing with multilevel caches and virtual memory is a *lot* more complicated.