Learning to think in a digital society

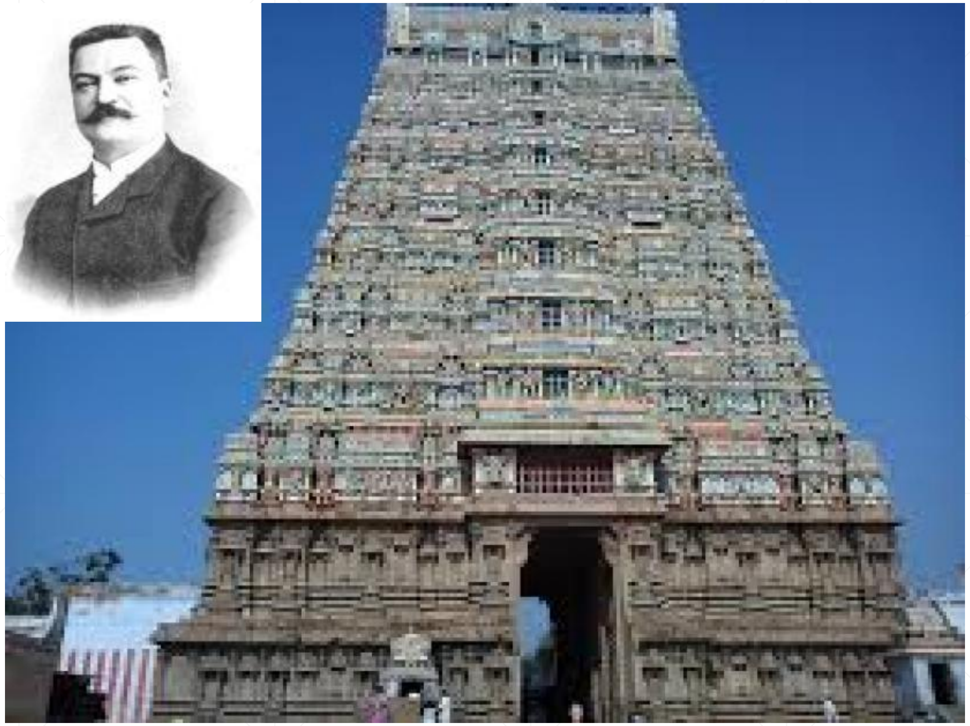
ΚΑΛΛΙΤΕΧΝΙΚΟ
ΣΧΟΛΕΙΟ ΓΕΡΑΚΑ
ΓΥΜΝΑΣΙΟ ΜΕ ΛΥΚΕΙΑΚΕΣ ΤΑΞΕΙΣ

Playing with the shadows: The Tower of Brahma

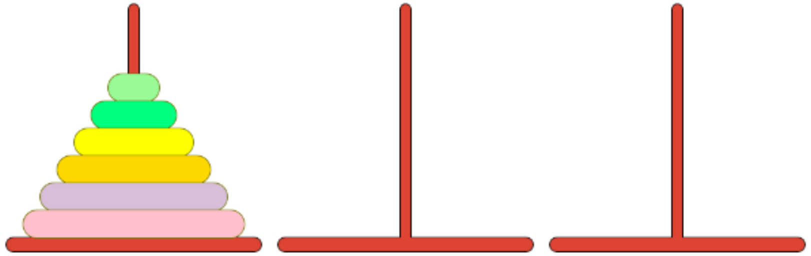
Grontas Panagiotis, Kotarinou Panagiota, Kouletsi Eirini, Milonis Konstantinos, Pliakou Maria, Hadzi Maria, Houpi Maria

The towers of Brahma

- Also known as towers of Hanoi
- A set of 64 gold disks on 3 diamond needles
- in the temple of Kashi Vishwanath in Varanasi, Uttar Pradesh, India
- The monks are trying to move all the gold disks between the needles
- When they succeed the world will end
- An ancient legend, or the imagination of Édouard Lucas (1883)?
  - Lucas numbers: $2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, ..$



Problem statement



- Three towers (rods that can take *stacks of disks*)
- A number of disks of different sizes, which can slide onto any tower.
- Game begins with a stack of n disks placed on the first tower.
- **Objective**:
  - move the entire stack to the last tower (or to the middle tower) using a helper tower
- **Rules**:
  - Only **one** disk can be moved at a time.
  - Each move consists of taking the **upper** disk from one of the stacks and placing it on **top** of another stack.
  - No disk may be placed on top of a **smaller** disk.

Solution:

- The Feynman ‘Method’
  - Read the problem, carefully
  - **Think very hard**
  - Write down the answer
- How to think very hard?
- **Apply methods from computational thinking**

Computational Thinking

- *Computer Science* is no more about (electronic) computers
  - than *astronomy* is about telescopes (Edsger W. Dijkstra**)**
  - than *biology* is about the microscope
  - than *chemistry* is about the test tube
  - Add *your favorite science* and its *tools*

Computational Thinking

- the universal methods of computer science that can be applied to real world problems
- the computing processes, **whether they are executed by a human** or by a machine (Jeannette M. Wing)
- Answers and raises fundamental questions about:
  - What is computable (doable)
  - How difficult/efficient is it to compute something
  - What can humans/computers do better than computers/humans
- Advanced by electronic computers in the same way that reading and writing was advanced by typography

Real world examples of computational processes

- Finding the minimum and/or maximum in a sequence of numbers
  - Application in discovering the range of a musical score
- Looking up a name in an alphabetically sorted list
  - start at the top or
  - start in the middle
- Sorting a class of students by height
  - When all are in the same room
  - As they come through the door
- Packing a bag with books/clothes
  - Abstraction
- A lawyer tries to find a loophole to acquit a defendant
  - A programmer tries to find why the program does not work correctly
  - A doctor tries to explain an unusual symptom
- In general: **problem solving**

Algorithm

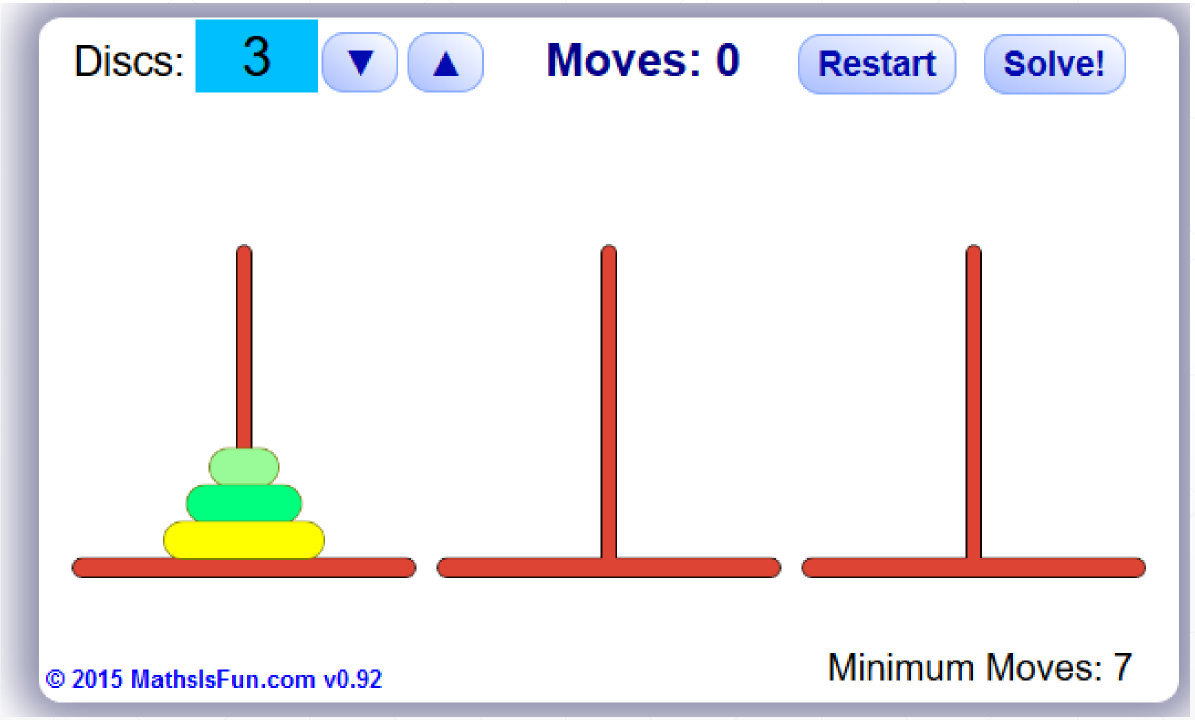
- The central concept of computer science
  - Describes the *steps* required to solve a problem
  - Consists of well defined actions
    - That execute in a particular order
    - Cause small changes
    - That add up to create the problem solution
  - We **are more interested in the steps** than the solution itself
- Properties:
  - Correctness
  - Resource use

Algorithm Building Blocks

- **Sequence**:
  - Put actions in order
  - Describe what must be done first, second etc.
- **Choice**:
  - Select actions to perform based on criteria.
  - Satisfy problem constraints
- **Iteration - Recursion**:
  - Identify patterns – actions that are repeated
  - Need not be identical – small differences are allowed
- These structures describe our everyday life – nothing special about computers

Back to Brahma - Practice

- Visit:
  - https://www.mathsisfun.com/games/towerofhanoi.html
- Try it for yourself some times
- Increase the number of disks
  - 4 and 5 will do
- Retry
- Record the number of moves



Keep in mind

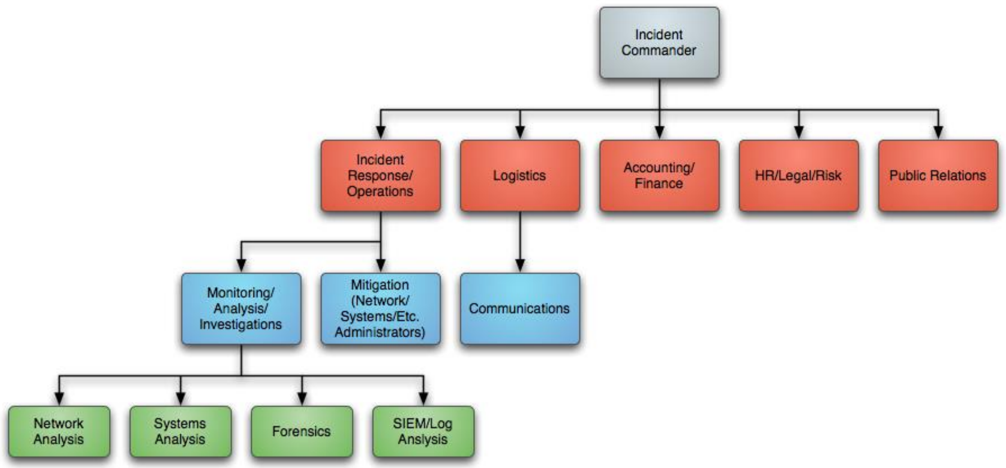
- Try not to move disks randomly
- Think that you are playing *in slow motion*
  - Identify the basic steps
  - Identify the constraints
  - Identify the repetitions
- **Tip**: Try to look some steps ahead:
  - **What** will it take for a particular disc to be placed on its final position
- Can we do better?
  - What is the minimum number of moves?

Problem Analysis

- Back to Feynman
  - How to think very hard at solving the towers game?
- We can break the difficult problem, into smaller subproblems
- Hopefully these smaller subproblems will be easier
- If not, repeat
- Break each subproblem to simpler *subsub*problems
- Until, we have reached problems that can be solved easily or even trivially

… and Synthesis

- Having solved the trivial problems, we combine the solutions
- Until we have solved the original problem
- How do we identify the subproblems
  - Smaller scale
  - Different aspects
  - It takes practice
- Computer science provides us with the vocabulary to transform our analysis into working computer instructions



Abstraction

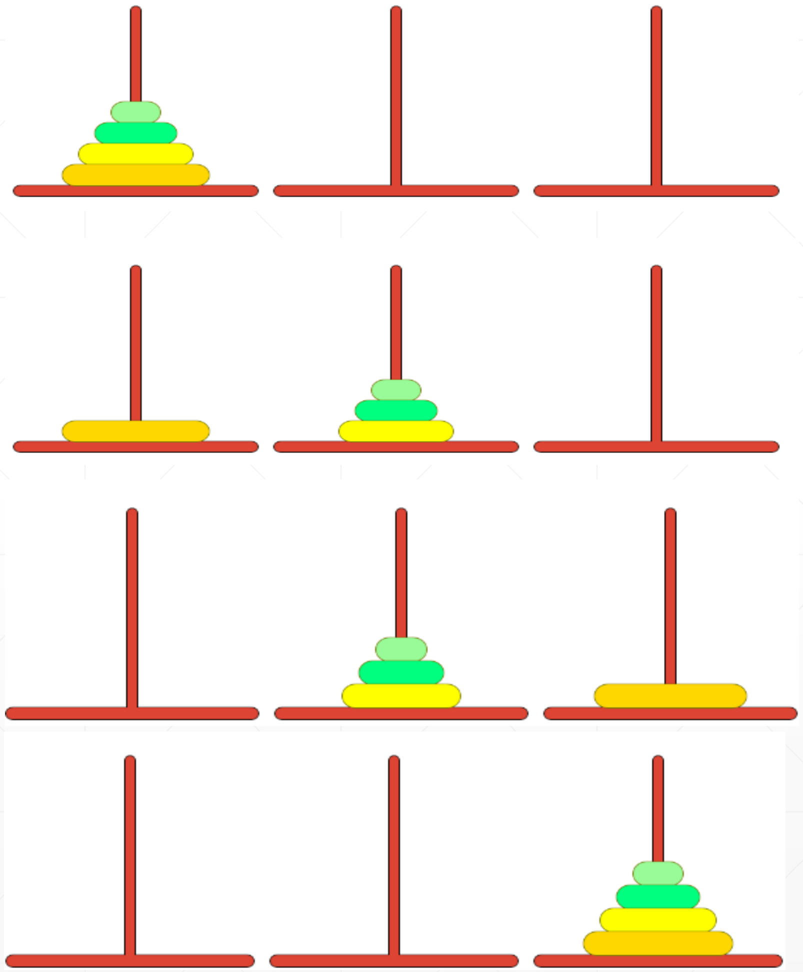
- If a subproblem remains hard there is a **magic** trick:
- **Assume** that we have solved it
  - Give it a *name*
- Can we use the solution as a black box to solve more difficult problems
- Black boxes are called **functions** in computer science
- As a result the initial problem is transformed into:
  - A trivial problem that can be solved
  - A difficult problem that cannot be solved but can be utilized
- Can we combine the two?

Case in point

- What are the subproblems?
- Since we are only moving disks the only possible subproblem is to move **fewer** disks
- The full problem:
  - `towers(n,A,C):` move n disks from tower A to tower C
- A smaller problem can be:
  - `towers(n-1,A,B)`
- The trivial problem is:
  - `towers(1,A,C):` move 1 disk from tower A to tower C
- How to combine

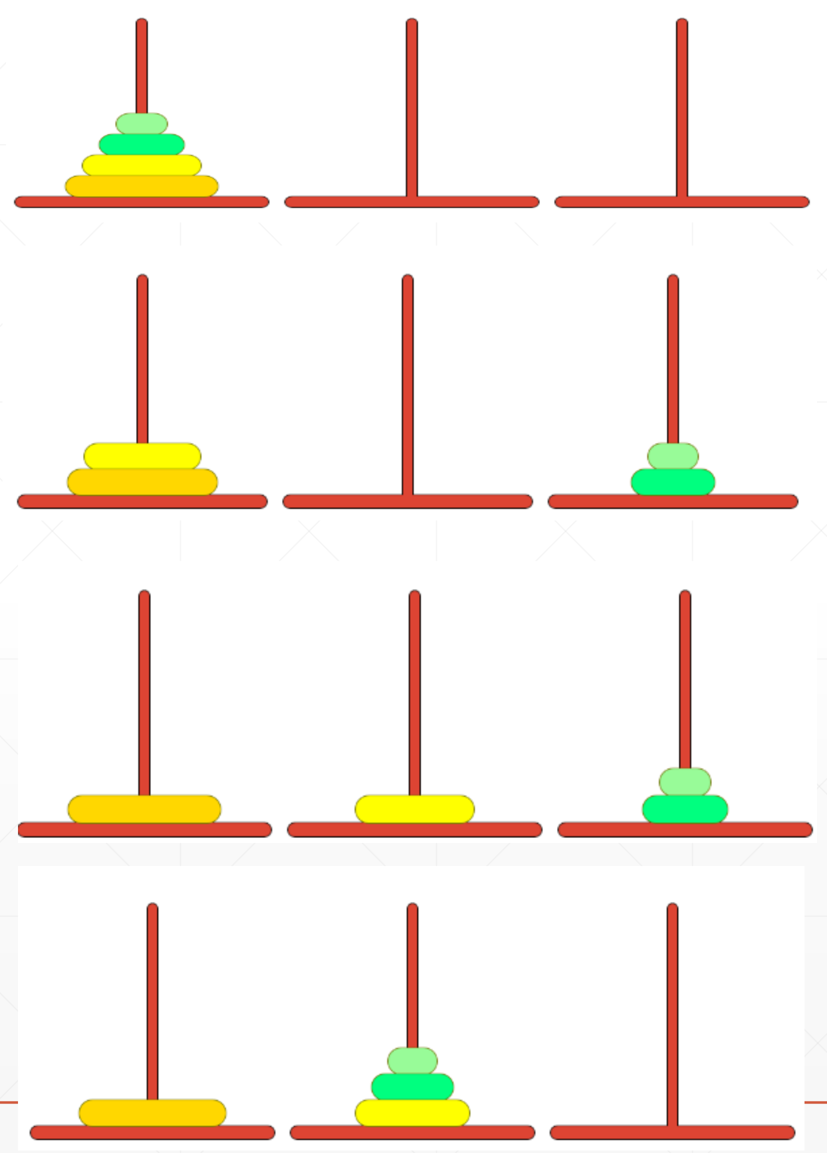
Subproblems: One less + One Move + One less

- To solve `towers(4,A,C)`
  - Solve `towers(3,A,B)`
    - Tower B is used as a helper
  - Solve `towers(1,A,C)`
    - Move disk 4 to tower C
    - The trivial step
  - Solve `towers(3,B,C)`



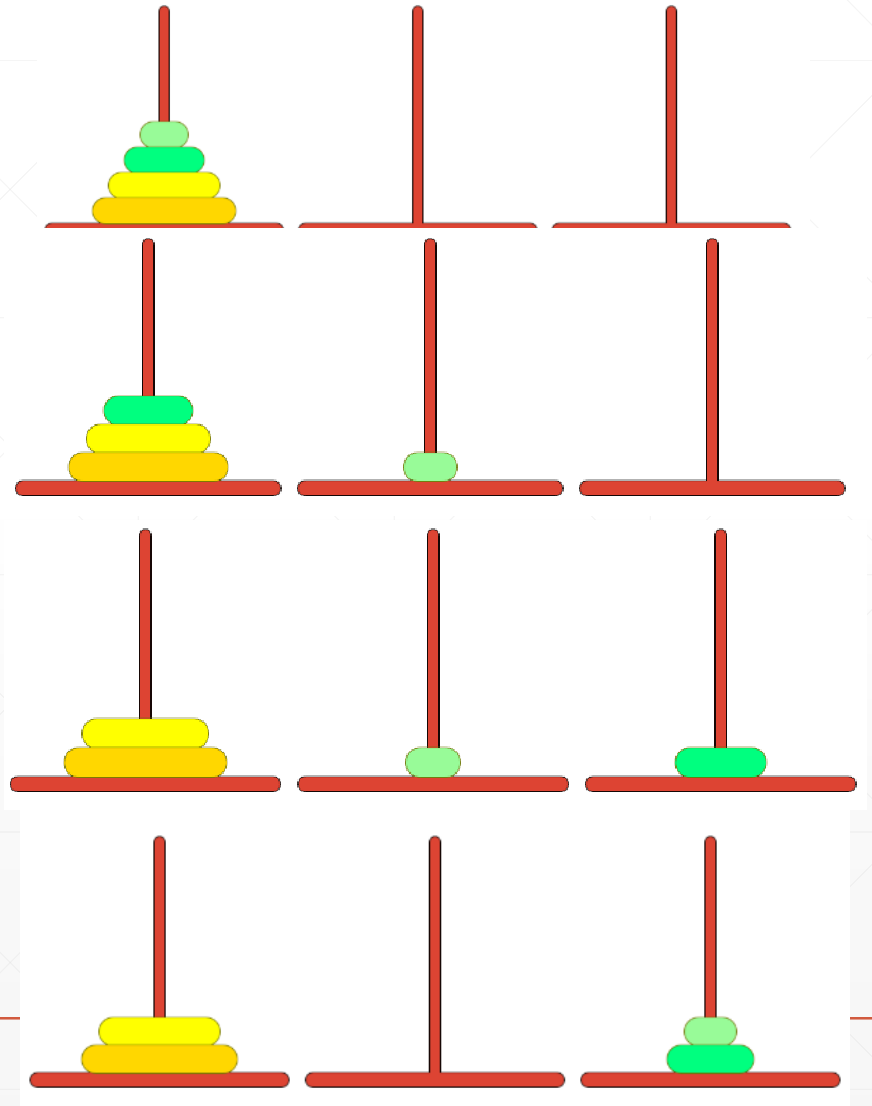
Repeat: One less + One Move + One less

- But: how to solve `towers(3,A,B)` ?
- **Abstract**!
  - Solve `towers(2,A,C)`
  - Solve `towers(1,A,B)`
  - Solve `towers(2,C,B)`



Repeat: One less + One Move + One less

- Again: how to solve `towers(2,A,C)` ?
- **Abstract**!
  - Solve `towers(1,A,B)`
  - Solve `towers(1,A,C)`
  - Solve `towers(1,B,C)`
- Trivial problems: Move a single disk



Coding Time

- Visit https://repl.it/KiNR/11
- A program in Python that solves the problem is presented
- Press the `fork` button to create a duplicate to play
- You have to complete the program using the presented analysis

```
game = [[],[],[]]
move = 0

def init(n):

def towerFromIndex(i):

def moveDisk(fromTower,toTower):
```

```
from helper import init,moveDisk

def towers(nd, source, dest):
    if nd>0:
        temp = 3-source-dest

n = 4
init(n)
towers(n,0,2)
```

Explanation & Instructions

Do **NOT** touch (helper.py)

- `game`: represents the current state of the towers
- `moves`: counts the moves
- `init`: housekeeping before the game is played
- `moveDisk`: moves the upmost disk between the towers specified

Do touch (main.py)

- `n`: to adjust the number of disks
- `towers`: the actions needed to solve the game
- **Hints**:
  - 0: tower A, 1: tower B, 2: tower C
  - They sum to 3, so to find temp tower subtract from 3 the sum of the other two
- Use only the `moveDisk` and `towers` functions

Your turn

Computational Complexity

- How many steps did the algorithm perform in relation to the input?
- Answer: For n disks: $t(n) = 2^n - 1$
- Why:

$$
\begin{aligned}
t(n) &= 2t(n-1) + 1 = \\
&= 2(2t(n-2)+1)+1 = \\
&= 4t(n-2)+3 = \\
&= 8t(n-3)+7 = \\
&= \ldots = \\
&= 2^{n-1}t(1) + 2^{n-1} - 1 = \\
&= 2^{n-1}\,1 + 2^{n-1} - 1 = \\
&= 2^n - 1
\end{aligned}
$$

- Is this ok?

Computational Complexity

- Answer: **no**
- Why?
  - For 64 discs: = $2^{64} - 1 = 1{,}845 \times 10^{19}$ steps
  - Assuming $1GHz$ CPU with one move per operation:
  - Solving will take $1{,}845 \times 10^{10}$ sec = 585 years
- Is this usable in practice? **NO**
- **But notice the following characteristic of the problem**:
  - **Easy to verify, difficult to compute**
- **(The) Open Question in CS:**
  - Can we make all solutions that are easy to verify easy to compute as well?