Analysis of Nonrecursive Algorithms

Analysis of Nonrecursive Algos

Example: Find the value of the **largest element** in a list of n real numbers.

Algorithm MaxElement($A[0..n-1]$)

// Input: An array $A[0..n-1]$

//Output: The value the max element in A

$max \leftarrow A[0]$

for $i \leftarrow 1$ **to** $n-1$ **do**

if $A[i] > max$

$max \leftarrow A[i]$

return max

Two operations in the loop

Basic operation: the comparison (it is executed in each repetition of the loop!)

Analysis of Nonrecursive Algos

Analysis:

- n is the number of times the comparison is executed.
- The **number of comparisons** will be the **same**.
- **No need to distinguish** among the worst, average and best cases.

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$

General Plan for Analysis

- Decide on parameter n indicating **input size**
- Identify algorithm's **basic operation**
- Determine **worst**, **average**, and **best** case efficiencies for input of size n
- Set up a **sum** for the number of times the basic operation is executed
- **Simplify** the sum using standard formulas and rules

Analysis of Nonrecursive Algos

Example: **The element uniqueness problem**: check if all elements in a given array of n elements are distinct.

Algorithm UniqueElement($A[0..n-1]$)

// Input: An array $A[0..n-1]$

//Output: Return "True" if all elements in A are distinct & "False" otherwise

for $i \leftarrow 0$ **to** $n-2$ **do**

for $j \leftarrow i+1$ **to** $n-1$ **do**

if $A[i] = A[j]$ **return** False

return True

Analysis of Nonrecursive Algos

- **The input size:** n, the number of the elements in an array
- **The basic operation:** the comparison operation in the innermost loop
- The number of comparisons depends not only on n but also on if there are **equal elements in the array**, if there are, which **array position** they occupy.
- **The worst-case inputs:**
  - arrays with no equal elements
  - arrays with the last two equal elements only

Analysis of Nonrecursive Algos

- **Worst-case analysis:**

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} \left[(n-1) - \left((i+1) - 1\right)\right]$$

$$= \sum_{i=0}^{n-2} (n - 1 - i) = (n-1) + (n-2) + \cdots + 1$$

$$= \frac{(n-2)(n-1)}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2)$$

Analysis of Nonrecursive Algos

Example: The **number of binary digits** in the binary representation of a positive integer.

Algorithm Binary(n)

// Input: A positive decimal integer n

//Output: The number of binary digits in the bin. repr.

$\text{count} \leftarrow 1$

while $n > 1$ **do**

$\quad \text{count} \leftarrow \text{count} + 1$

$\quad n \leftarrow \lfloor n/2 \rfloor$

return count

Analysis of Nonrecursive Algos

- **The basic operation:** the number of times of the **comparisons** executed (which is larger than the number of repetitions of the loop's body by exactly one)

- **The input size:** the loop variable n takes only a few values between its lower and upper limits:

$$\mathrm{n}, \left\lfloor \frac{\mathrm{n}}{2} \right\rfloor, \left\lfloor \frac{\mathrm{n}}{4} \right\rfloor, \left\lfloor \frac{\mathrm{n}}{8} \right\rfloor, \dots, 1$$

- **The total number:** In each loop repetition, n is about halved, thus,

the total $\# \approx \log_2 \mathrm{n}$

Analysis of Recursive Algorithms

General Plan for Analysis

- Decide on a parameter indicating an **input's size**.
- Identify the algorithm's **basic operation**.
- Check whether the number of times the basic operation is executed **may vary** on different inputs of the same size. (If it may, the worst, average, and best cases must be investigated separately.)
- Set up a **recurrence relation** with an appropriate initial condition expressing the number of times the basic operation is executed.
- **Solve the recurrence** (or, at the very least, establish its solution's order of growth).

Tower of Hanoi Puzzle

Tower of Hanoi Puzzle

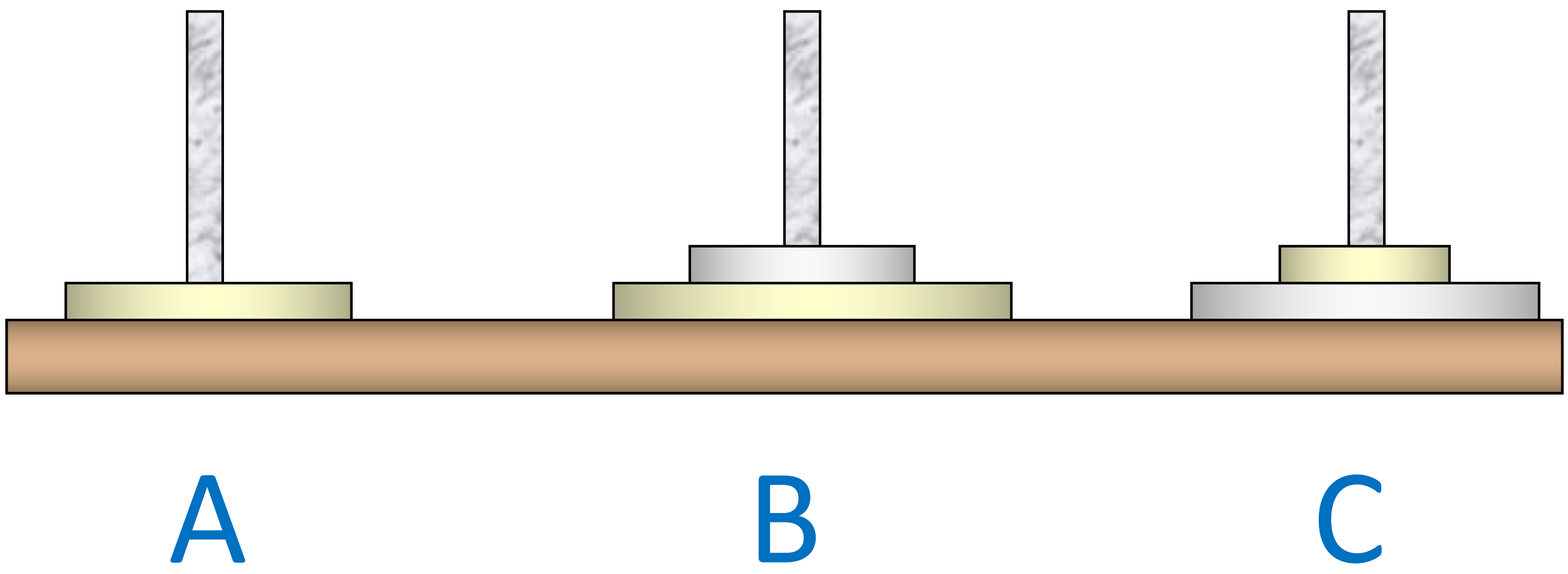
1. There are n **disks of different sizes** that can slide onto any of three pegs/towers.

2. Initially, all disks are on the first peg in order of size, the largest on the bottom and the smallest on the top.

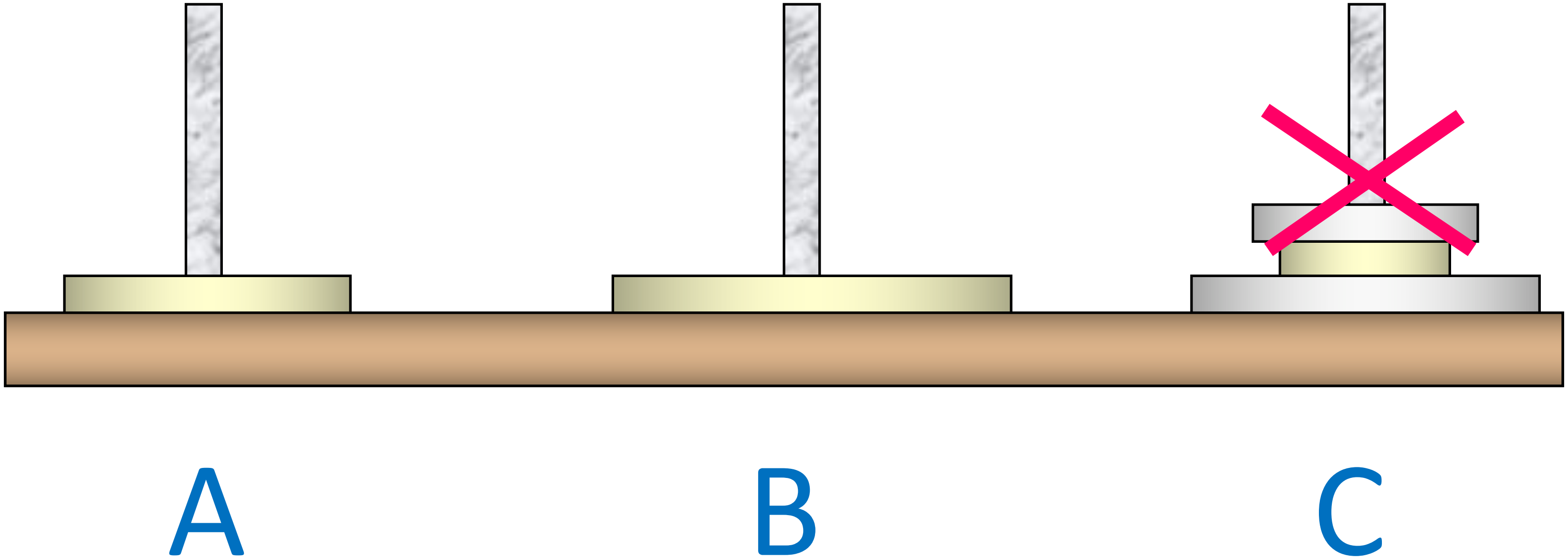
3. The goal is to move all the disks to the third peg, using the second as an auxiliary.

4. At a time, only one disk can be moved and it is forbidden to place a larger disk on the top of a smaller one.

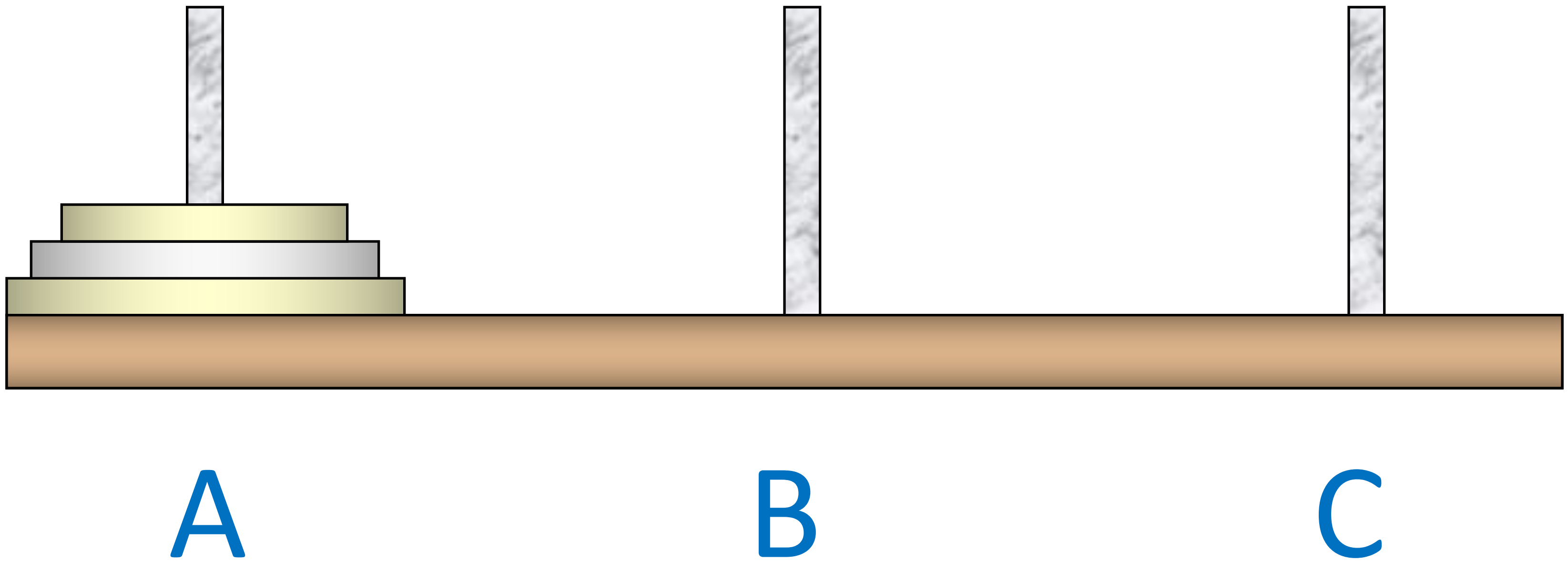
Tower of Hanoi Puzzle



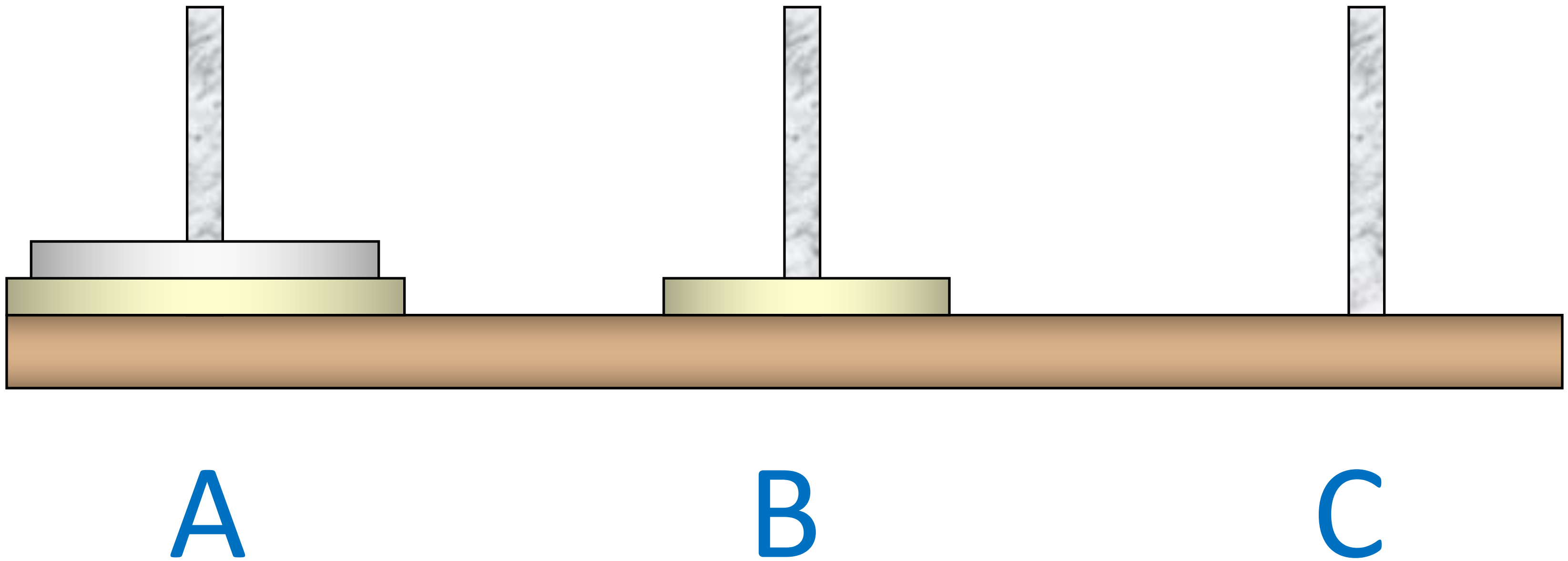
Tower of Hanoi Puzzle



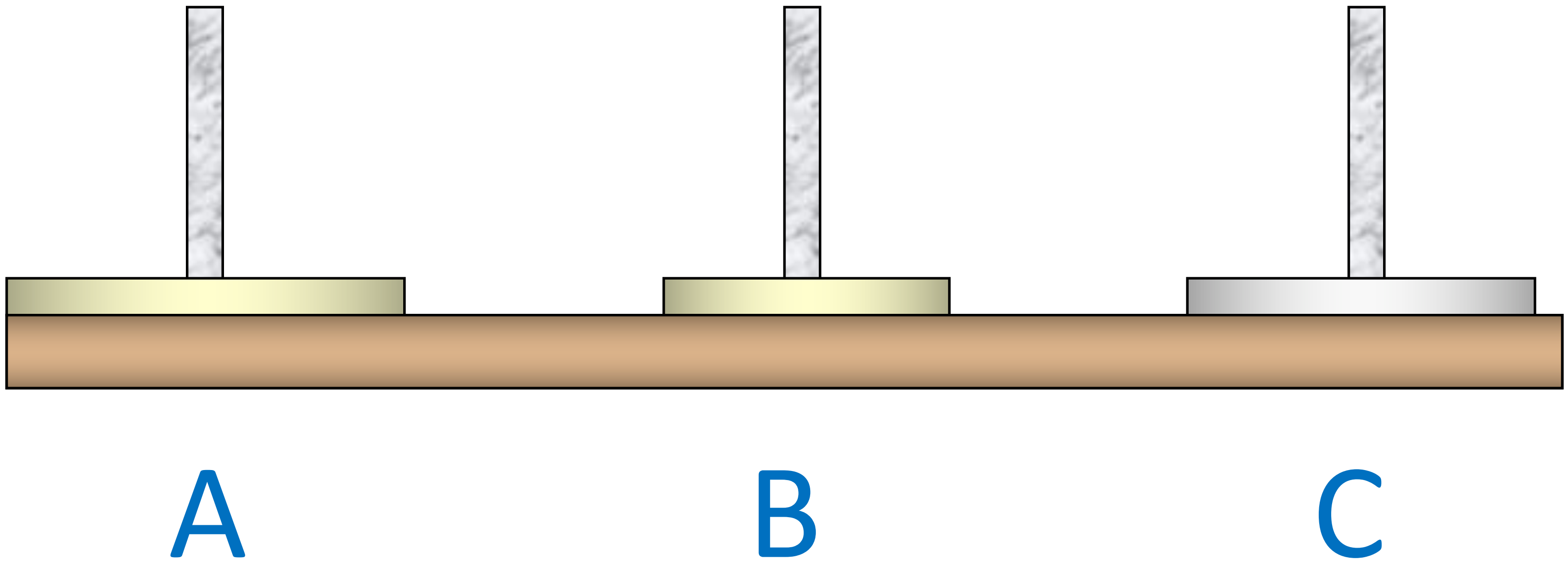
Tower of Hanoi Puzzle



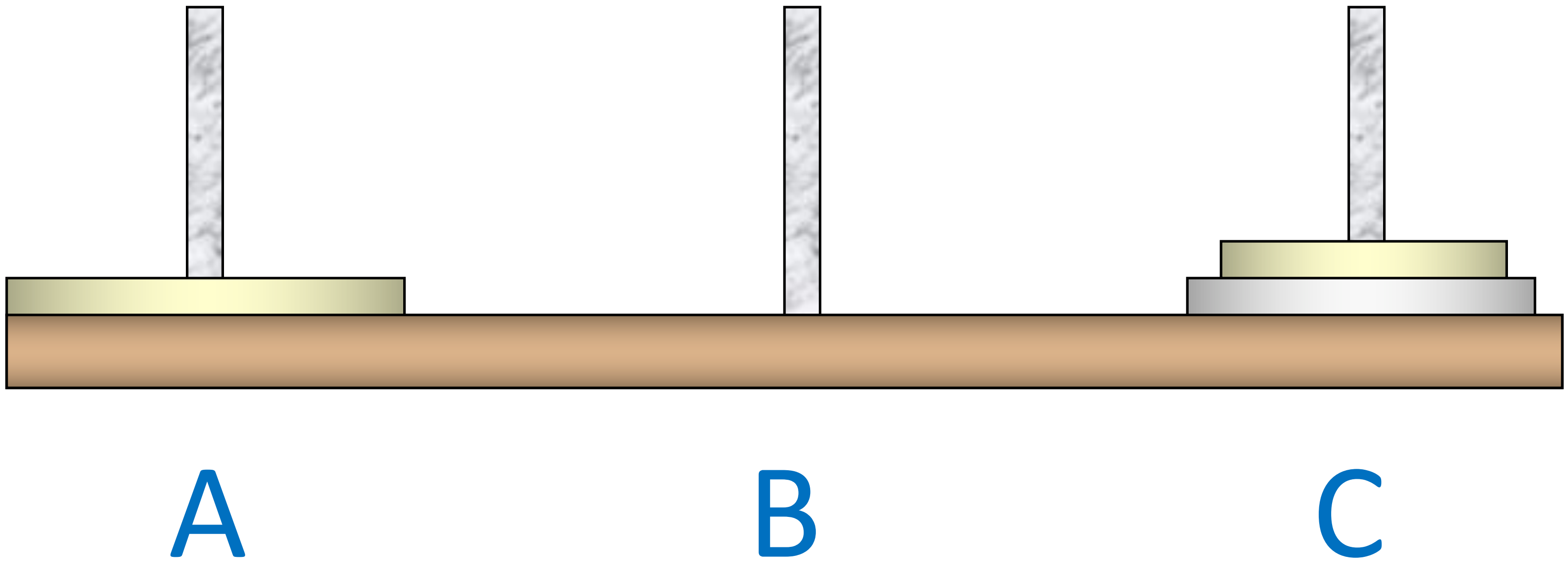
Tower of Hanoi Puzzle



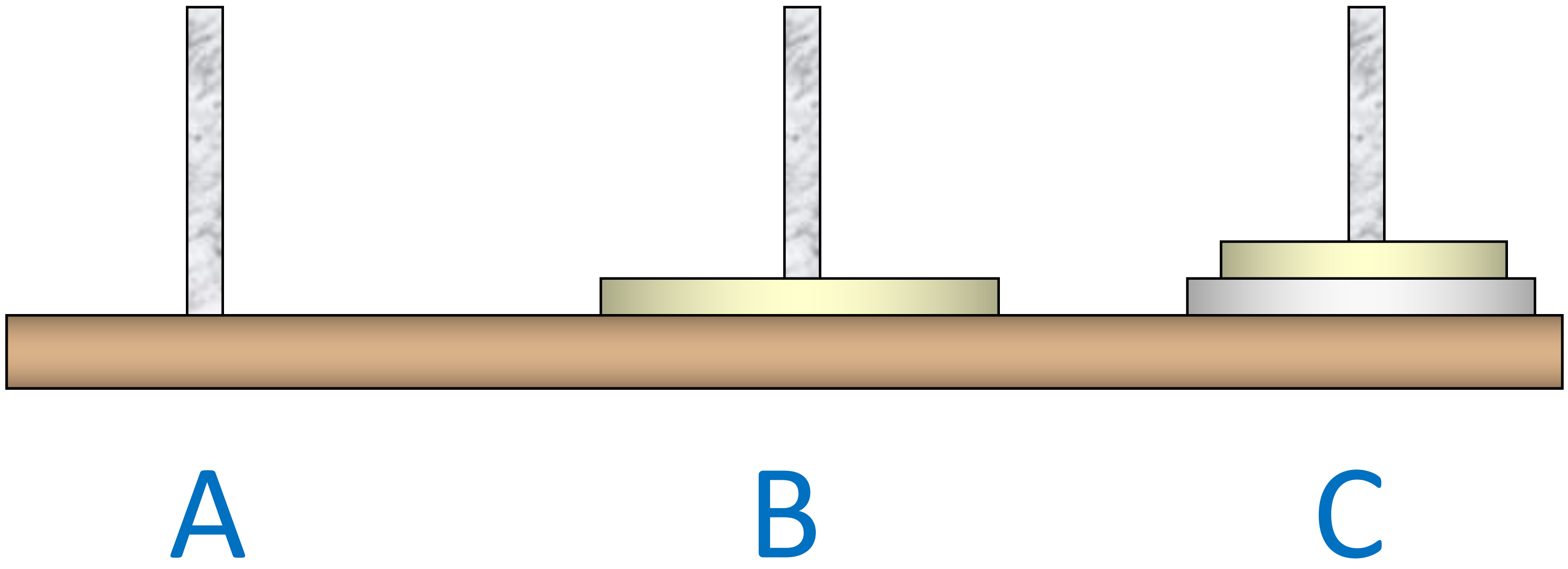
Tower of Hanoi Puzzle



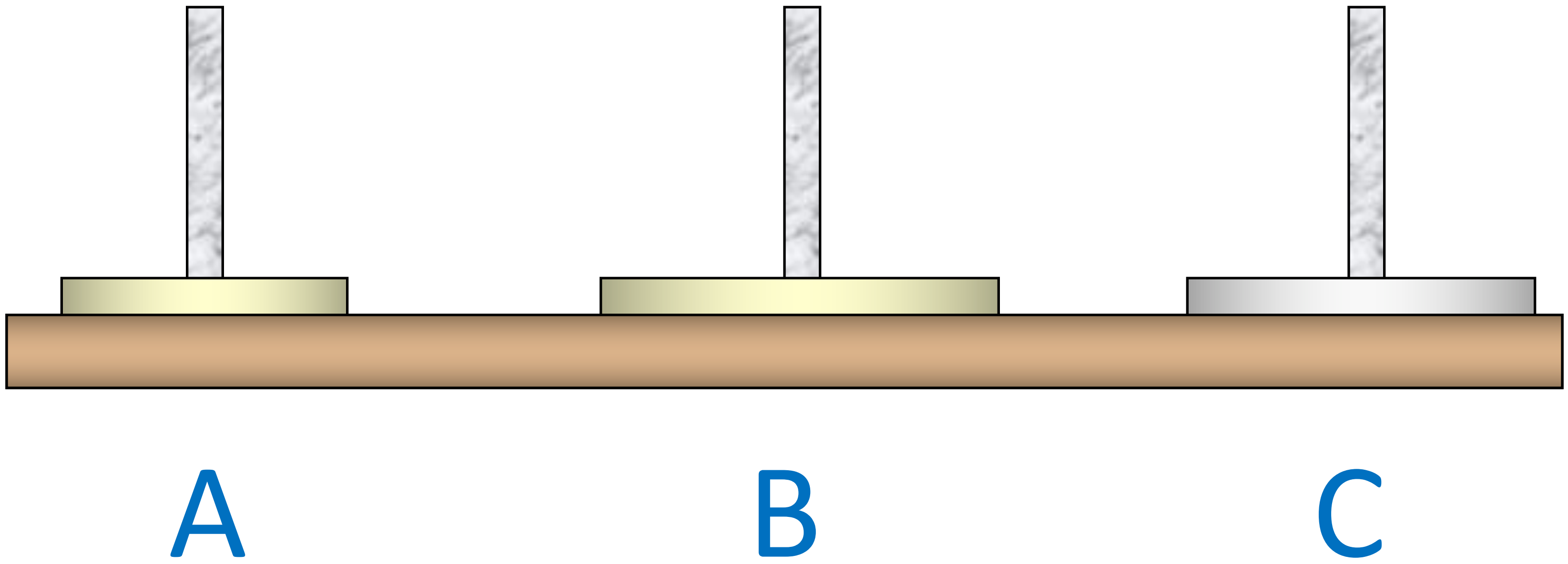
Tower of Hanoi Puzzle



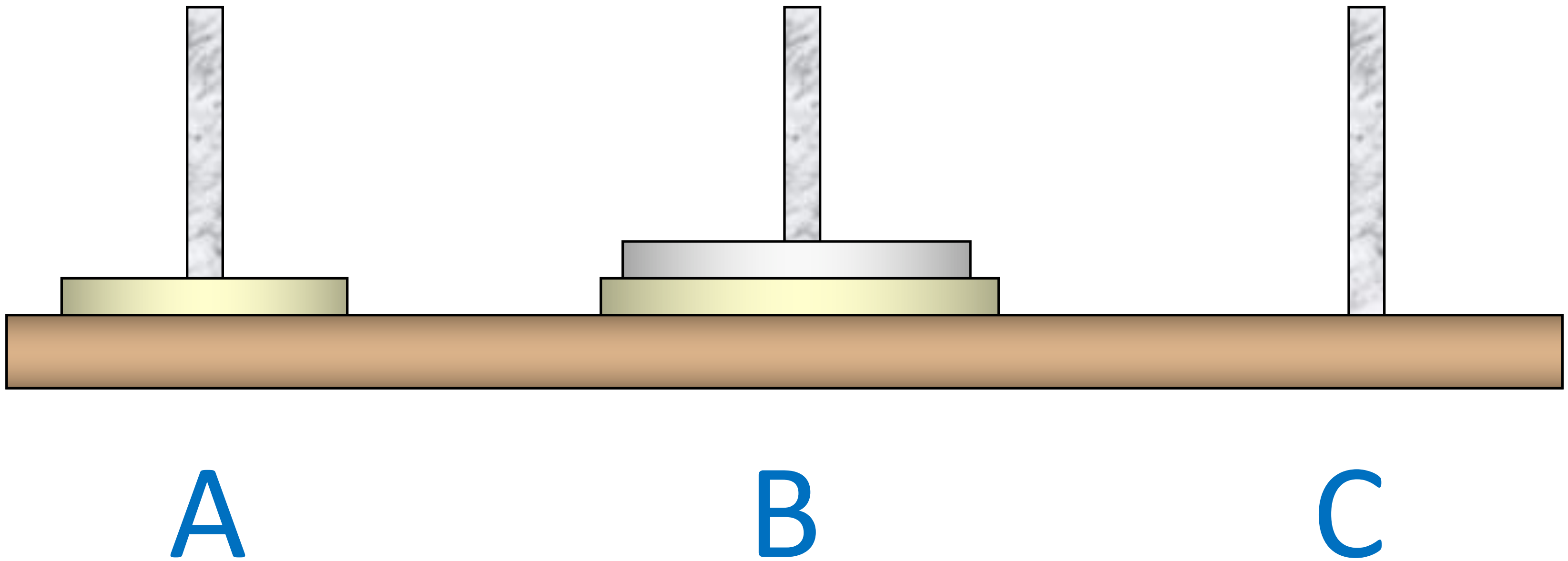
Tower of Hanoi Puzzle



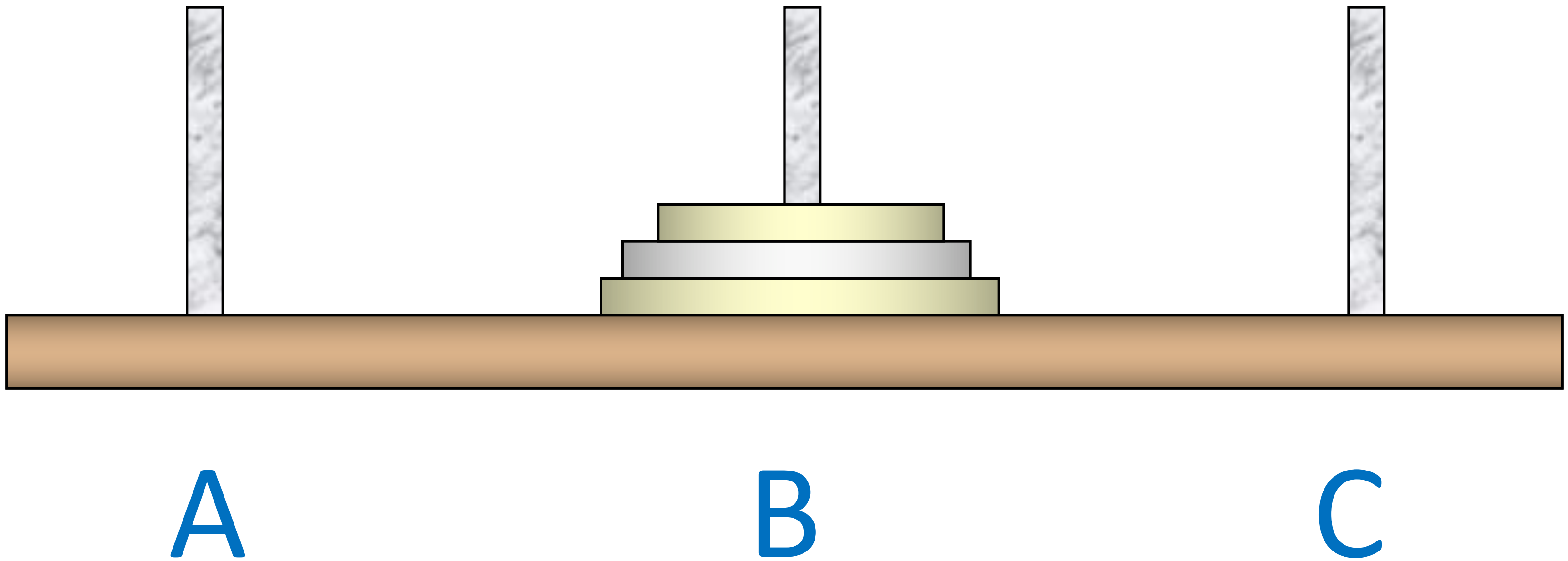
Tower of Hanoi Puzzle



Tower of Hanoi Puzzle



Tower of Hanoi Puzzle



Tower of Hanoi Puzzle

Recursive solution: To move n disks from peg 1 to peg 3 (peg 2 is auxiliary),

- we first move **recursively** $n - 1$ disks from peg 1 to peg 2 (peg 3 is auxiliary),
- then we move the largest disk from peg 1 to peg 3 directly and **recursively** move $n - 1$ disks from peg 2 to peg 3 (peg 1 is auxiliary).
- If $n = 1$, we move the single disk from peg 1 to peg 3.

Try: http://www.mathsisfun.com/games/towerofhanoi.html

Tower of Hanoi Puzzle

Analysis:

- The **input size** indicator is the number of disks n.
- The **basic operation** is moving one disk.
- The total number of moves $\mathrm{M(n)}$ depends on n only.
- The **recurrence equation** is

$$\mathrm{M(n) = M(n-1) + 1 + M(n-1)}$$

$$\mathrm{M(1) = 1}$$

Tower of Hanoi Puzzle

Use the **method of backward substitutions**:

$$M(n) = 2M(n-1) + 1 = 2(M(n-2) + 1) + 1$$

$$= 2^2 M(n-2) + 2 + 1$$

$$= 2^3 M(n-3) + 2^2 + 2 + 1$$

$$\ldots$$

$$= 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2 + 1$$

$$\ldots$$

$$= 2^{n-1} M(1) + 2^{n-2} + \cdots + 2 + 1 = 2^n - 1 \in O(2^n)$$

Analysis of Recursive Algos

Example: Find the **number of binary digits** in the binary representation of a positive integer.

Algorithm BinRec(n)

// Input: A positive decimal integer n

//Output: The number of binary digits in the bin. repr.

if $\mathrm{n} = 1$ **return** 1

else return BinRec($\lfloor \mathrm{n}/2 \rfloor$) $+ 1$

Analysis of Recursive Algos

Example: Find the **number of binary digits** in the binary representation of a positive integer.

Algorithm BinRec(n)

// Input: A positive decimal integer n

//Output: The number of binary digits in the bin. repr.

if $n = 1$ **return** 1

else return BinRec($\lfloor n/2 \rfloor$) $+ 1$

Let $A(n)$ be the **total number of additions**.

$$A(n) = A(\lfloor n/2 \rfloor) + 1, n > 1, \qquad A(1) = 0$$

Analysis of Recursive Algos

For the simplicity we choose $n = 2^k$ (the smoothness rule)

$$
\begin{aligned}
A(2^k) &= A\left(\frac{2^k}{2}\right) + 1 = A(2^{k-1}) + 1 \\
&= \left(A(2^{k-2}) + 1\right) + 1 = A(2^{k-2}) + 2 \\
&= \left(A(2^{k-3}) + 1\right) + 2 = A(2^{k-3}) + 3 \\
&= \cdots \\
&= A(2^{k-i}) + i \\
&= \cdots \\
&= A(2^{k-k}) + k = A(1) + k = k = \log_2 n
\end{aligned}
$$