Fundamentals

Mathematical Analysis of Non-recursive Algorithms

- Decide on parameter *n* indicating input size
- Identify algorithm's basic operation
- Determine worst, average, and best cases 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

Useful Formulas and Rules

$$\sum_{l}^{u} 1 = 1 + 1 + \cdots + 1 = u - l + 1$$

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

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \in \Theta(n^2)$$

$$\sum_{i=1}^{n} i^2 = 1 + 4 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \in \Theta(n^3)$$

$$\sum_{i=1}^{n} a^i = a + a^2 + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1}$$

$$\sum_{i=1}^{n} 2^i = 2 + 4 + \cdots + 2^n = \frac{2^{n+1} - 1}{2 - 1} \in \Theta(2^n)$$

Useful Formulas and Rules

$$\sum (a_i \mp b_i) = \sum a_i \mp \sum b_i$$

$$\sum c a_i = c \sum a_i$$

$$\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=m+1}^{u} a_i$$

$$\sum_{i=l}^{u} a_i = \sum_{i=1}^{u} a_i - \sum_{i=1}^{l-1} a_i$$

Recursion

- When a function or procedure calls itself.
- Applies to both mathematics and programming
- Recursive mathematical equations are called recurrence equations, recurrence relations, or simply recurrences.
- Recurrences naturally occur in the analysis of recursive algorithms.

Recursion

$x(n) = x(n-1) + n$

- $x(n)$ is a generic term representing the n^{th} term in some numerical sequence
- The recurrence says that the n^{th} term in the sequence can be determined by adding n to the $(n-1)^{th}$ term in the sequence
- To determine the actual sequence, we also need some initial condition (multiple initial conditions may be needed)

Mathematical Analysis of Recursive Algorithm

- Decide on a parameter indicating an input's size.
- Identify the algorithm's basic operation.
- Check whether the number of times the basic op. 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 op. is executed.
- Solve the recurrence (or, at the very least, establish its solution's order of growth) by backward substitutions or another method.

Solving Recurrences

- Solving a recurrence means to find an explicit (non-recursive) formula for $x(n)$, or to prove the no such sequence exists

- In algorithm analysis we need this solution to:
  - Compare functions
  - Plot functions

- Recurrences can be expensive to compute directly

Forward Substitution

- Start from the initial condition
- Substitute into recurrence
- Repeat until you can see a pattern

Backward Substitution

- Start from the recurrence
- Substitute into itself to get function for previous value
- Repeat until you can see a pattern

Decrease-by-One Recurrences

- Use the solution to an $n-1$ size problem to solve an n size problem

- Typical efficiency recurrence:

$$T(n) = T(n-1) + f(n)$$

- Backward substitution yields:

$$T(n) = T(0) + \sum_{i=1}^{n} f(i)$$

Decrease-by-Constant Recurrences

- Use the solution to an $\frac{n}{b}$ size problem to solve an size n problem (typically $b = 2$)

- Typical efficiency recurrence:

$$T(n) = T(\frac{n}{b}) + f(n)$$

- Backward substitution yields:

$$T(n) = T(1) + \sum_{i=1}^{k} f(b^i)$$

$$k = \log_b n$$

Divide-and-Conquer Recurrences

- Divide-and-conquer algorithms divide the problem into several smaller problems, which are solved recursively, then combines the solutions

- Typical efficiency recurrence:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

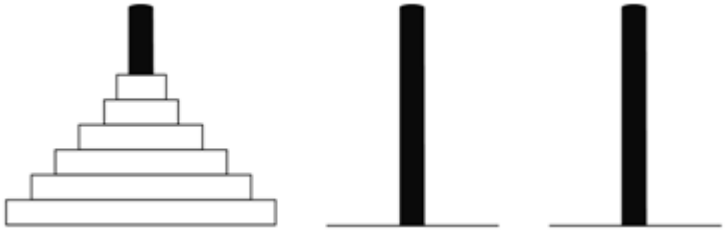
- Backward substitution yields:

$$T(n) = n^{\log_b a}[T(0) + \sum_{i=1}^{k} \frac{f(b^i)}{a^i}]$$

$$k = \log_b n$$

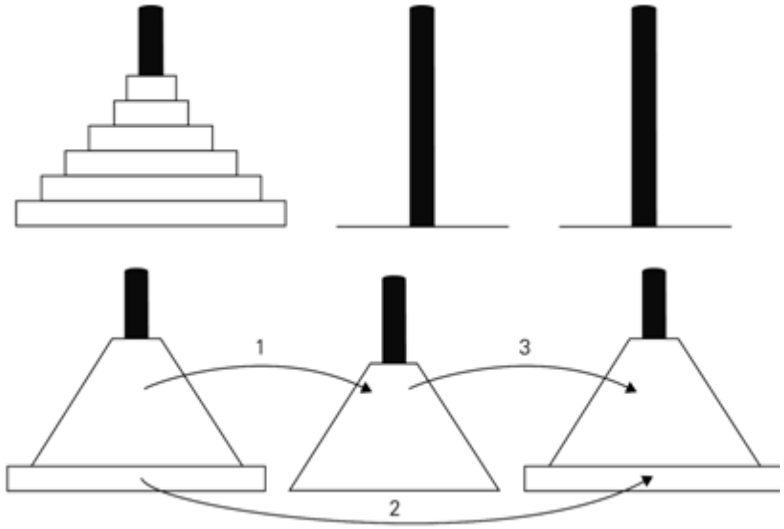
The Tower of Hanoi Puzzle

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

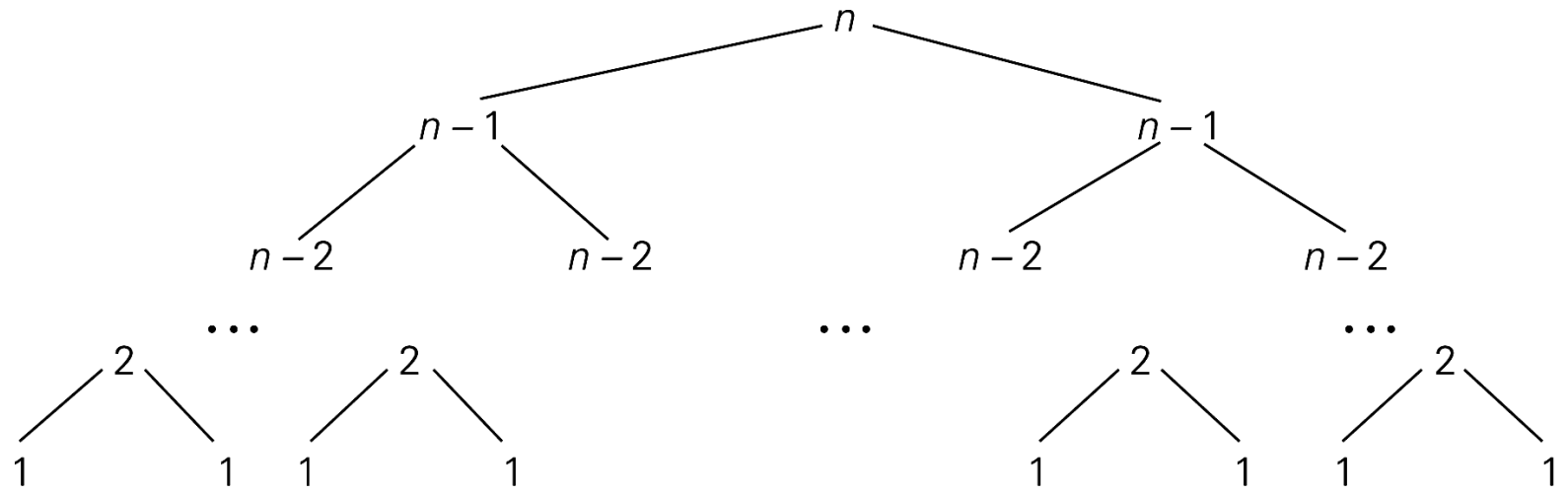


The Tower of Hanoi Puzzle

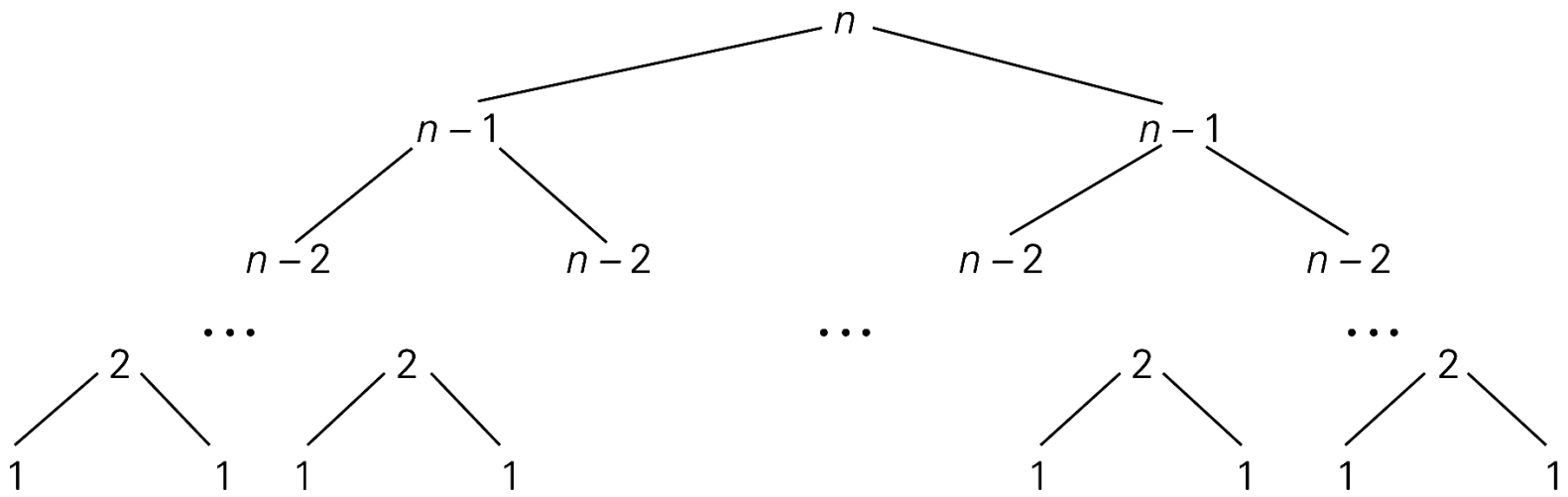
- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.



Tree of Calls



Tree of Calls



$$C(n) = \sum_{l=0}^{n-1} 2^l = 2^n - 1$$