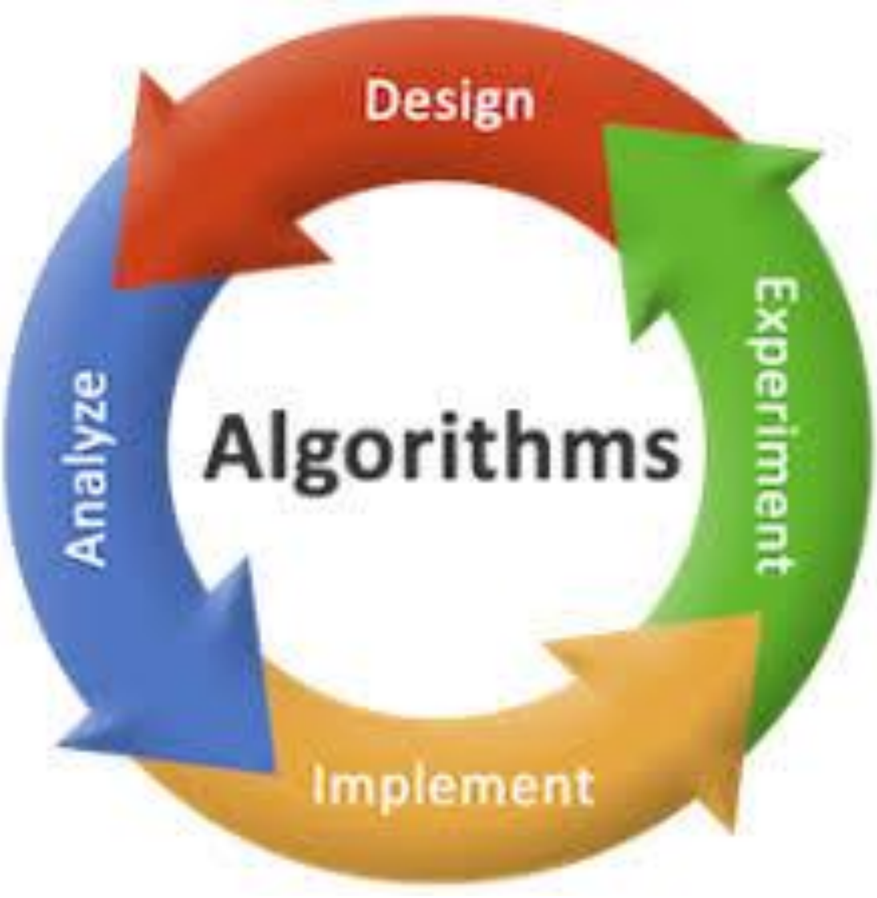


Design & Analysis of Algorithms

Lecture 9

Mathematical Analysis of Recursive Algorithms-II

Mathematical Analysis of Recursive Algorithms

General Plan for Analyzing the Time Efficiency

- **1.** Decide on a parameter (or parameters) indicating an input's size.
- **2.** Identify the algorithm's basic operation.
- **3.** Check whether the number of times the basic operation is executed can vary on different inputs of the same size; if it can, the **worst-case**, **average-case**, and **best-case** efficiencies must be investigated separately.

Mathematical Analysis of Recursive Algorithms

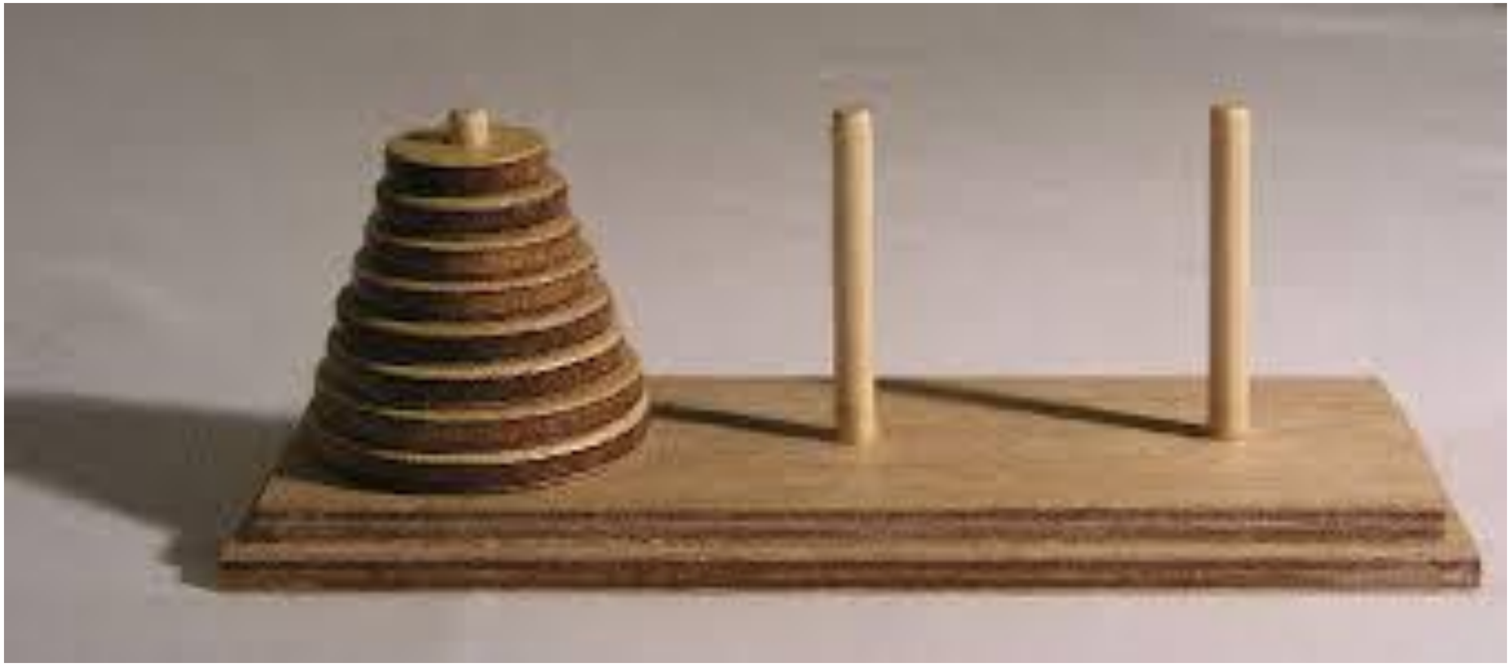
General Plan for Analyzing the Time Efficiency

- **4.** Set up a recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed.
- **5.** Solve the recurrence or, at least, ascertain the order of growth of its solution.

Mathematical Analysis of Recursive Algorithms

Example 2

- Solving the **Tower of Hanoi** puzzle.

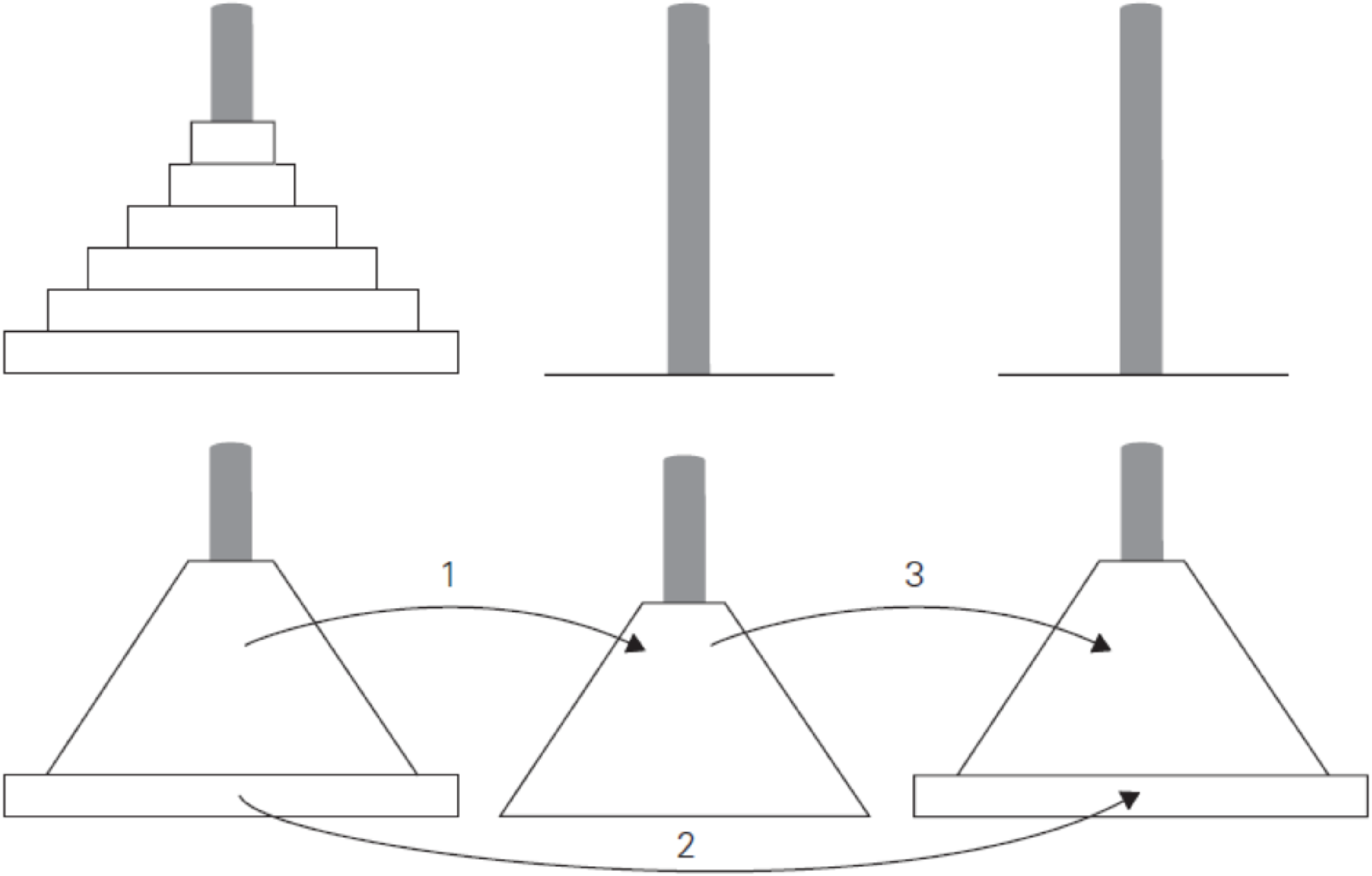


Mathematical Analysis of Recursive Algorithms

Tower of Hanoi

- In this puzzle, we have ***n*** disks of different sizes that can slide onto any of three pegs.
- Initially, all the disks are on the first peg in order of size, the largest on the bottom and the smallest on top.
- The **goal** is to move all the disks to the third peg, using the second one as an auxiliary, if necessary.
- We can move only one disk at a time, and it is forbidden to place a larger disk on top of a smaller one.

Mathematical Analysis of Recursive Algorithms



Recursive solution to the Tower of Hanoi puzzle

Mathematical Analysis of Recursive Algorithms

Recursive solution to the Tower of Hanoi puzzle

- To move ***n*>1** disks from peg 1 to peg 3 (with peg 2 as auxiliary), we first move recursively ***n* − 1** disks from peg 1 to peg 2 (with peg 3 as auxiliary), then move the largest disk directly from peg 1 to peg 3, and, finally, move recursively $n - 1$ disks from peg 2 to peg 3 (using peg 1 as auxiliary).

- Of course, if ***n* = 1**, we simply move the single disk directly from the source peg to the destination peg.

Mathematical Analysis of Recursive Algorithms

Recursive solution to the Tower of Hanoi puzzle

- The input's size indicator- the number of disks ***n***
- Basic operation- **moving one disk**
- The number of moves ***M(n)*** depends on ***n*** only, and the recurrence equation is the following:

$$M(n) = M(n-1) + 1 + M(n-1) \quad \text{for } n > 1.$$

- With the **initial condition**, we get the recurrence relation as

$$M(n) = 2M(n-1) + 1 \quad \text{for } n > 1,$$

$$M(1) = 1.$$

Mathematical Analysis of Recursive Algorithms

Recursive solution to the Tower of Hanoi puzzle

$$M(n) = 2M(n-1) + 1 \qquad \text{sub. } M(n-1) = 2M(n-2) + 1$$

$$= 2[2M(n-2) + 1] + 1 = 2^2M(n-2) + 2 + 1 \qquad \text{sub. } M(n-2) = 2M(n-3) + 1$$

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

after *i* substitutions, we get

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

for $n = 1$, which is achieved for $i = n - 1$, we

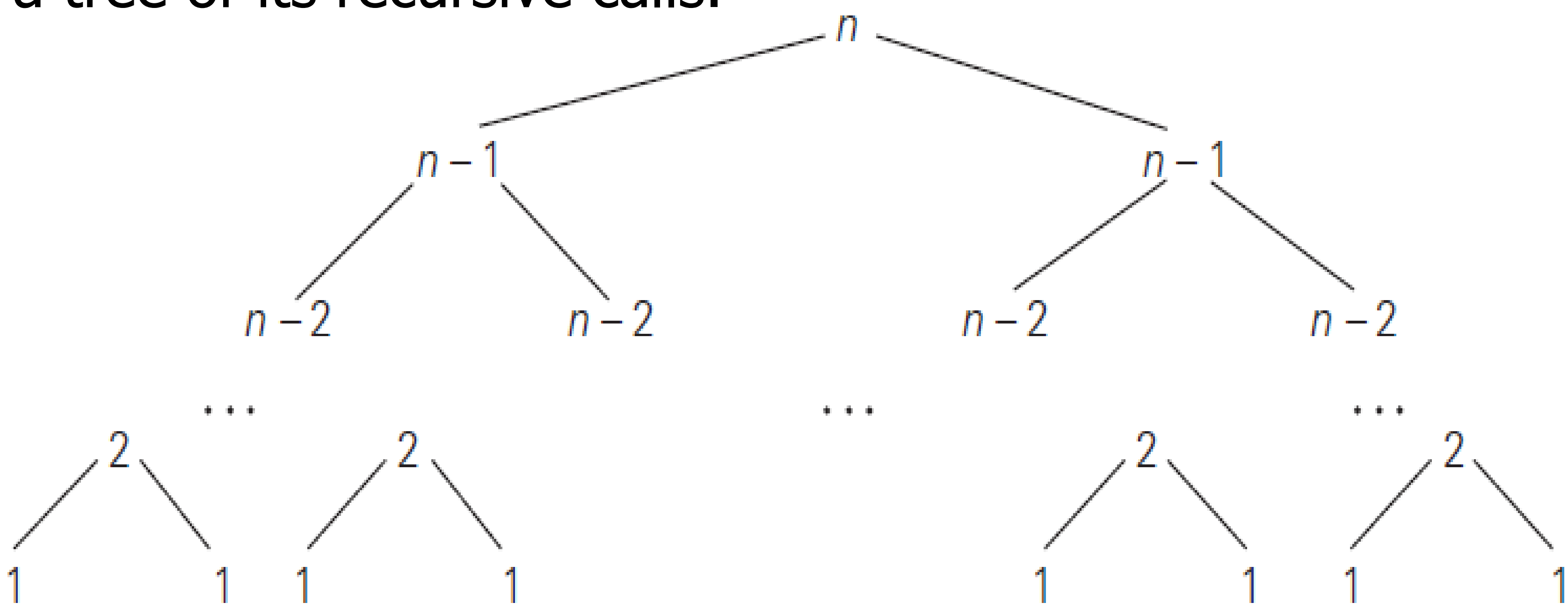
$$M(n) = 2^{n-1}M(n-(n-1)) + 2^{n-1} - 1$$

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

Mathematical Analysis of Recursive Algorithms

Recursive solution to the Tower of Hanoi puzzle

- When a recursive algorithm makes more than a single call to itself, it can be useful for analysis purposes to construct a tree of its recursive calls.



Mathematical Analysis of Recursive Algorithms

Recursive solution to the Tower of Hanoi puzzle

- By counting the number of nodes in the tree, we can get the total number of calls made by the Tower of Hanoi algorithm:

$$C(n) = \sum_{l=0}^{n-1} 2^l \text{ (where } l \text{ is the level in the tree in Figure)} = 2^n - 1$$

Mathematical Analysis of Recursive Algorithms

Example 3

- Binary number representation.

ALGORITHM $BinRec(n)$

//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
if $n = 1$ **return** 1
else return $BinRec(\lfloor n/2 \rfloor) + 1$

Mathematical Analysis of Recursive Algorithms

Binary number representation.

- Recurrence relation

$$A(n) = A(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1$$

$$A(1) = 0$$

- Standard approach to solving such a recurrence is to solve it only for $\boldsymbol{n = 2^k}$ and then take advantage of the theorem called the ***smoothness rule.***

$$A(2^k) = A(2^{k-1}) + 1 \quad \text{for } k > 0,$$

$$A(2^0) = 0.$$

Mathematical Analysis of Recursive Algorithms

- Recurrence relation Solution

$$A(2^k) = A(2^{k-1}) + 1 \qquad \text{substitute } A(2^{k-1}) = A(2^{k-2}) + 1$$
$$= [A(2^{k-2}) + 1] + 1 = A(2^{k-2}) + 2 \qquad \text{substitute } A(2^{k-2}) = A(2^{k-3}) + 1$$
$$= [A(2^{k-3}) + 1] + 2 = A(2^{k-3}) + 3 \qquad \dots$$
$$\dots$$
$$= A(2^{k-i}) + i$$
$$\dots$$
$$= A(2^{k-k}) + k.$$

Thus, we end up with

$$A(2^k) = A(1) + k = k,$$

or, after returning to the original variable $n = 2^k$ and hence $k = \log_2 n$,

$$A(n) = \log_2 n \in \Theta(\log n).$$

References

- **Chapter 2**: Anany Levitin, “Introduction to the Design and Analysis of Algorithms”, Pearson Education, Third Edition, 2017.

- **Chapter 2**: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, “Introduction to Algorithms”, MIT Press/PHI Learning Private Limited, Third Edition, 2012.

Homework

- Fibonacci Number Generation
  - Non-recursively
  - Recursively