UNIT I INTRODUCTION

Introduction to Algorithms – Fundamentals of Algorithmic Problem Solving – Fundamentals of the Analysis of Algorithmic Efficiency – Analysis Framework – Asymptotic Notations and Basic Efficiency Classes – Mathematical Analysis of Recursive Algorithms – Mathematical Analysis of Non-recursive Algorithms

Mathematical Analysis of Recursive Algorithms

Plan for Analysis of Recursive Algorithms

- 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.

Smoothness Rule

- Let f(n) be a nonnegative function defined on the set of natural numbers. f(n) is call smooth if it is eventually nondecreasing and $f(2n) \in \Theta(f(n))$
  - Functions that do not grow too fast, including *logn*, *n*, *nlogn*, and n^{α} where $\alpha >= 0$ are smooth.
- Smoothness rule

  Let T(n) be an eventually nondecreasing function and f(n) be a smooth function. If $T(n) \in \Theta(f(n))$ for values of n that are powers of b, where b>=2, then $T(n) \in \Theta(f(n))$ for any n.

Example 1: Recursive evaluation of *n*!

Definition: $n! = 1 * 2 * \ldots * (n-1) * n$ for $n \geq 1$ and $0! = 1$

Recursive definition of *n*!: $F(n) = F(n-1) * n$ for $n \geq 1$ and

$F(0) = 1$

ALGORITHM $F(n)$

//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

Size: n
Basic operation: Multiplications
Recurrence relation: M(n) = M(n-1) + 1, M(0) = 0

Solving the recurrence for M(*n*)

$M(n) = M(n-1) + 1, \quad M(0) = 0$

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

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

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

...

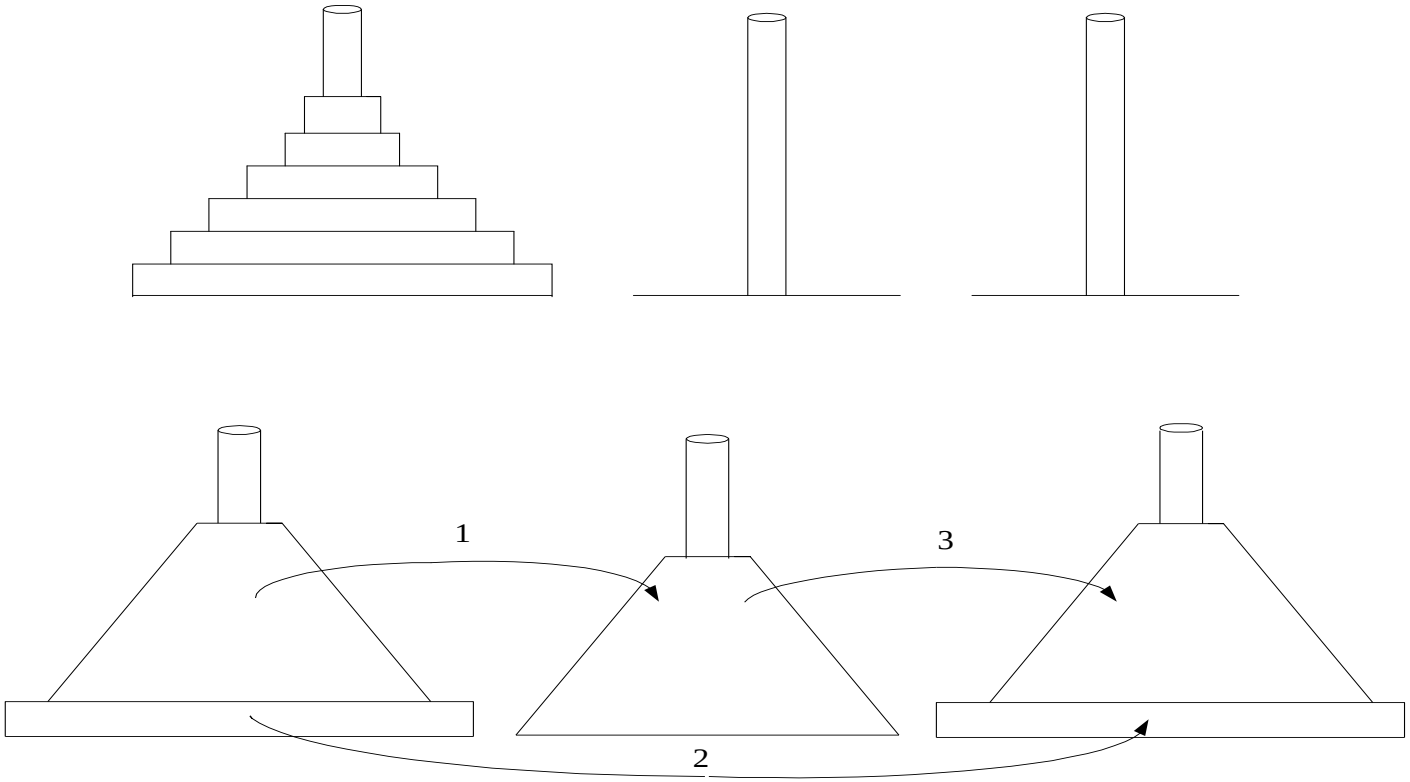
$= M(n-i) + i$

$= M(0) + n$

$= n$

The method is called backward substitution.

Example 2: The Tower of Hanoi Puzzle



Recurrence for number of moves: M(n) = 2M(n-1) + 1

Solving recurrence for number of moves

$M(n) = 2M(n-1) + 1, \quad M(1) = 1$

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

$= 2(2(M-2) + 1) + 1 = 2^2*M(n-2) + 2^1 + 2^0$

$= 2^2*(2M(n-3) + 1) + 2^1 + 2^0$

$= 2^3*M(n-3) + 2^2 + 2^1 + 2^0$

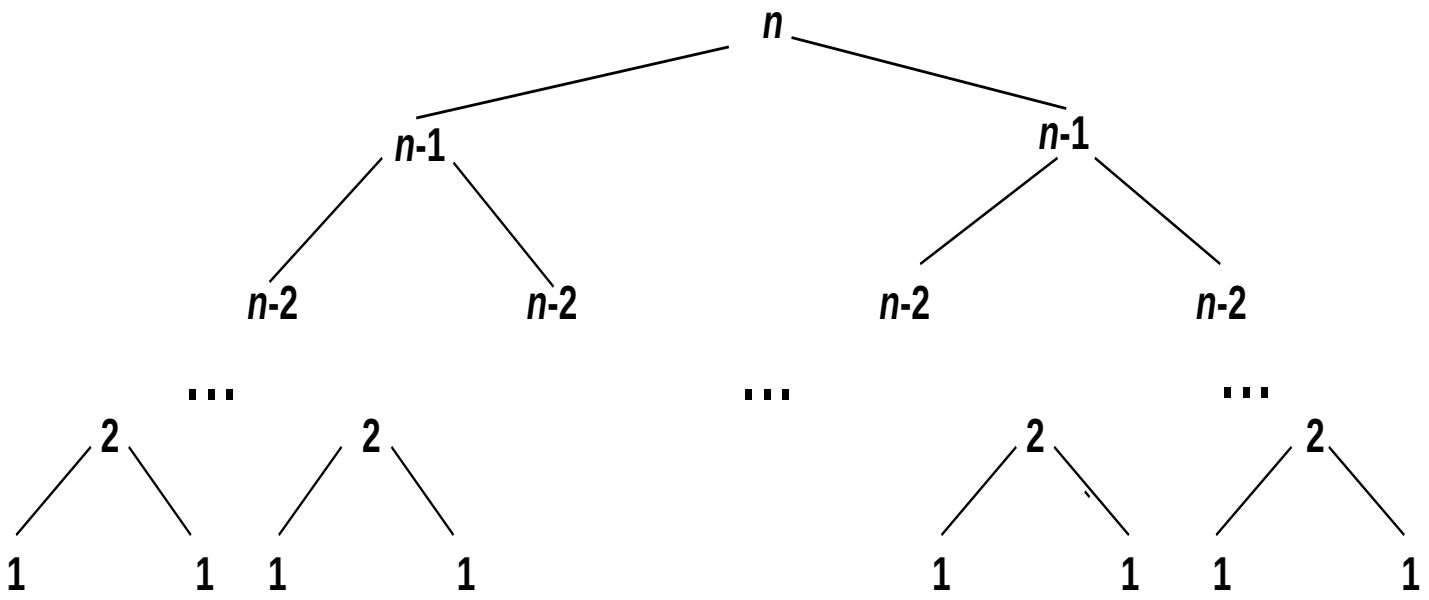
$= \ldots$

$= 2^{(n-1)}*M(1) + 2^{(n-2)} + \ldots + 2^1 + 2^0$

$= 2^{(n-1)} + 2^{(n-2)} + \ldots + 2^1 + 2^0$

$= 2^n - 1$

Tree of calls for the Tower of Hanoi Puzzle



Example 3: Counting #bits

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*$(\lfloor n/2 \rfloor) + 1$

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

$A(2^k) = A(2^{k-1}) + 1, \quad A(2^0) = 0$ (using the Smoothness Rule)

$= (A(2^{k-2}) + 1) + 1 = A(2^{k-2}) + 2$

$= A(2^{k-i}) + i$

$= A(2^{k-k}) + k = k$

$= \log_2 n$


Thank You