Design and Analysis of Algorithms

Vandana M L
Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Mathematical Analysis of Recursive Algorithms

Slides courtesy of **Anany Levitin**

Vandana M L

Department of Computer Science & Engineering

Steps in Mathematical Analysis of Recursive Algorithms

- Decide on parameter n indicating input size
- Identify algorithm's basic operation
- If the number of times the basic operation is executed varies with different inputs of same sizes , investigate worst, average, and best case efficiency separately
- Set up a recurrence relation and initial condition(s) for C(n)-the number of times the basic operation will be executed for an input of size n
- Solve the recurrence or estimate the order of magnitude of the solution

- Decrease-by-one recurrences

A decrease-by-one algorithm solves a problem by exploiting a relationship between a given instance of size *n* and a smaller size *n – 1*.

Example: n!

The recurrence equation has the form

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

- Decrease-by-a-constant-factor recurrences

A decrease-by-a-constant algorithm solves a problem by dividing its given instance of size *n* into several smaller instances of size *n/b*, solving each of them recursively, and then, if necessary, combining the solutions to the smaller instances into a solution to the given instance.

Example: binary search.

The recurrence has the form

$$T(n) = aT(n/b) + f(n)$$

- One (constant) operation reduces problem size by one.

  $T(n) = T(n-1) + c \qquad T(1) = d$

  Solution: $T(n) = (n-1)c + d$ ***linear***

- A pass through input reduces problem size by one.

  $T(n) = T(n-1) + c\, n \qquad T(1) = d$

  Solution: $T(n) = [n(n+1)/2 - 1]\, c + d$ ***quadratic***

Methods to solve recurrences

- Substitution Method
    - Mathematical Induction
    - Backward substitution
- Recursion Tree Method
- Master Method (Decrease by constant factor recurrences)

Recursive Evaluation of n!

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

Recursive definition of $n!$:

$F(n) = F(n-1) * n$ for $n \geq 1$

$F(0) = 1$

ALGORITHM $F(n)$

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

input size?

basic operation?

Best/Worst/Average Case?

$M(n) = M(n-1) + 1$ for $n > 0$,

$M(0) = 0$.

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

M(n) = n

Overall time Complexity: Θ(n)

Counting number of binary digits in binary representation of a number

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

input size?

basic operation?

Best/Worst/Average Case?

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

$$\begin{aligned}
A(2^k) &= A(2^{k-1}) + 1 && \text{substitute } A(2^{k-1}) = A(2^{k-2}) + 1 \\
&= [A(2^{k-2}) + 1] + 1 = A(2^{k-2}) + 2 && \text{substitute } A(2^{k-2}) = A(2^{k-3}) + 1 \\
&= [A(2^{k-3}) + 1] + 2 = A(2^{k-3}) + 3 && \ldots \\
&\ldots \\
&= A(2^{k-i}) + i \\
&\ldots \\
&= A(2^{k-k}) + k.
\end{aligned}$$

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

Tower of Hanoi

```
Algorithm TowerOfHanoi(n, Src, Aux, Dst)
    if (n = 0)
        return
    TowerOfHanoi(n-1, Src, Dst, Aux)
    Move disk n from Src to Dst
    TowerOfHanoi(n-1, Aux, Src, Dst)
```

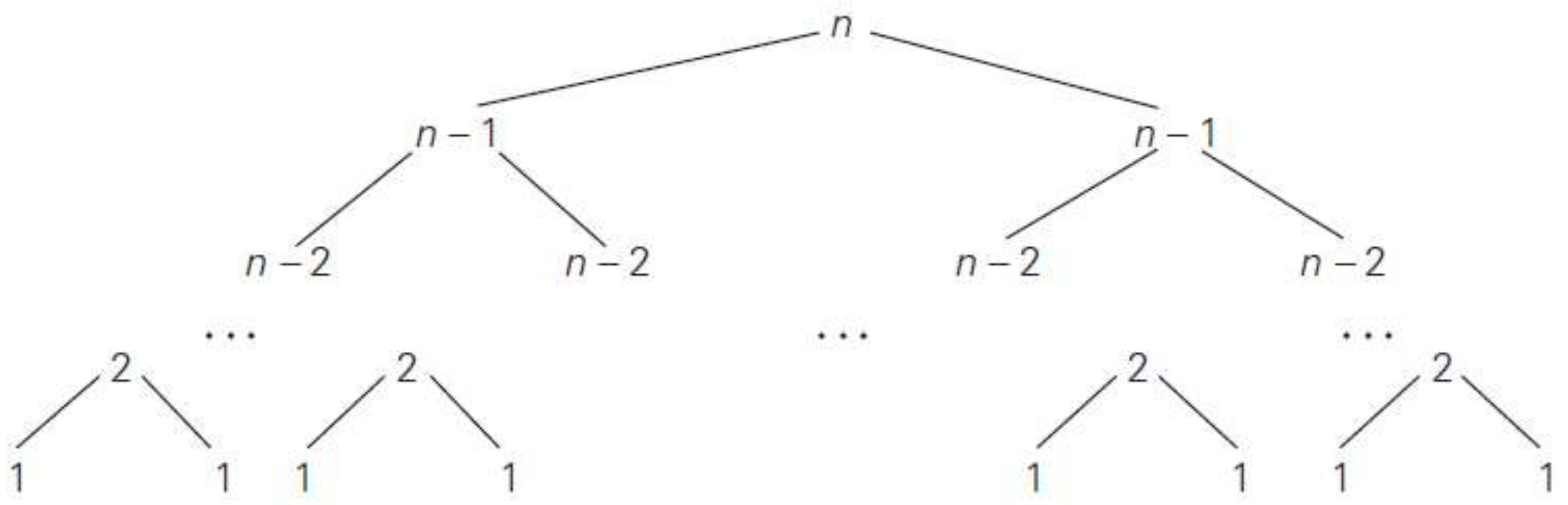
Input Size: **n**

Basic Operation : **Move disk n from Src to Dst**

$C(n) = 2C(n-1) + 1$ for $n > 0$ and $C(0)=0$

$= 2^n - 1 \in \Theta(2^n)$

Tower of Hanoi : Tree of Recursive calls



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

THANK YOU

Vandana M L
Department of Computer Science & Engineering

vandanamd@pes.edu