

DSE 2256 DESIGN & ANALYSIS OF ALGORITHMS

Lecture 8 & 9

Mathematical Analysis of Recursive Algorithms

Instructors:

Dr. Savitha G,
Assistant Professor, DSCA, MIT, Manipal

Dr. Abhilash K. Pai,
Assistant Professor, DSCA, MIT, Manipal



Recap of L6 & L7

- Mathematical analysis of non-recursive algorithms
 - General Plan for analysing time efficiency
 - Summation formulas
 - Algorithm : Max. element in an array
 - Algorithm : Unique elements in an array
 - Algorithm : Multiplication of two $n \times n$ matrices

Mathematical analysis of Recursive Algorithms I

- **Example 1: Factorial of a number**

ALGORITHM $F(n)$

//Computes $n!$ recursively

//Input: A nonnegative integer n

//Output: The value of $n!$

if $n = 0$ **return** 1

else return $F(n - 1) * n$

- **Definition:**

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

- **Recursive definition (recurrence relation):**

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

$F(0) = 1$ -----> Initial Condition

Input size	:	n
Basic Operation	:	Multiplication
Recurrence relation	:	$M(n) = M(n-1) + 1$ $M(0) = 0$

To compute $F(n-1)$

To multiply $F(n-1)$ by n

the calls stop when $n = 0$ $\xrightarrow{\quad}$ $M(0) = 0.$ $\xleftarrow{\quad}$ no multiplications when $n = 0$

Mathematical analysis of Recursive Algorithms II

- **Example 1: Solving the recurrence for $M(n)$**

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

$$M(0) = 0$$

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

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

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

...

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

...

$$= M(0) + n$$


$$= n$$

The method is called **backward substitution**.

Mathematical analysis of Recursive Algorithms II

- **Example 1: Solving the recurrence for $M(n)$**

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

$$M(0) = 0$$

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

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

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

...

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

...

$$= M(0) + n$$


$$= n$$

The method is called **backward substitution**.

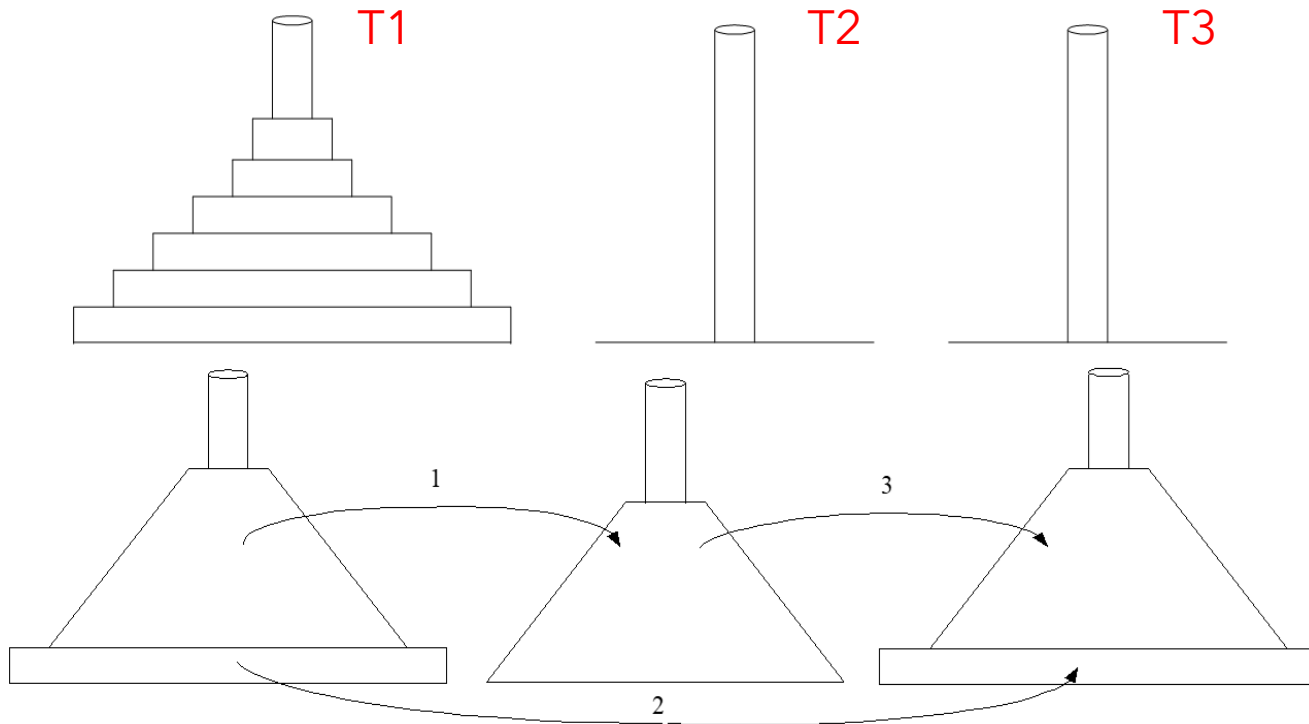
Mathematical analysis of Recursive Algorithms IV

- **General Plan for Analysis of Recursive Algorithms:**

1. Decide on a parameter indicating an input's size.
2. Identify the algorithm's basic operation.
3. Check whether the number of times the basic operation is executed may vary on different inputs of the same size.
4. Set up a recurrence relation with an appropriate initial condition expressing the number of times the basic operation is executed.
5. Solve the recurrence by backward substitutions or another method.

Mathematical analysis of Recursive Algorithms V

- Example 2: Tower of Hanoi**



ALGORITHM Tower_of_hanoi (n,s,d,a)

// Recursively moves disks from source (s) to destination (d)

// Input : the no. of disks (n), source (s), destination (d), auxiliary (a) towers. Initially, a and d towers are empty.

// Output is the sequence of moves

```
if (n==1) then do
    print ("Move from s to d")
    return
```

else

Tower_of_hanoi (n-1, s, a, d) -----> 1. Recursive call to move n-1 disks from T1 to T2

print ("Move from s to d") -----> 2. Move the disk from source to destination

Tower_of_hanoi (n-1, a, d, s) -----> 3. Recursive call to move n-1 disks from T2 to T3

end if

Mathematical analysis of Recursive Algorithms VI

Solving recurrence for number of moves:

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

$$M(1) = 1$$

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

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

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

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

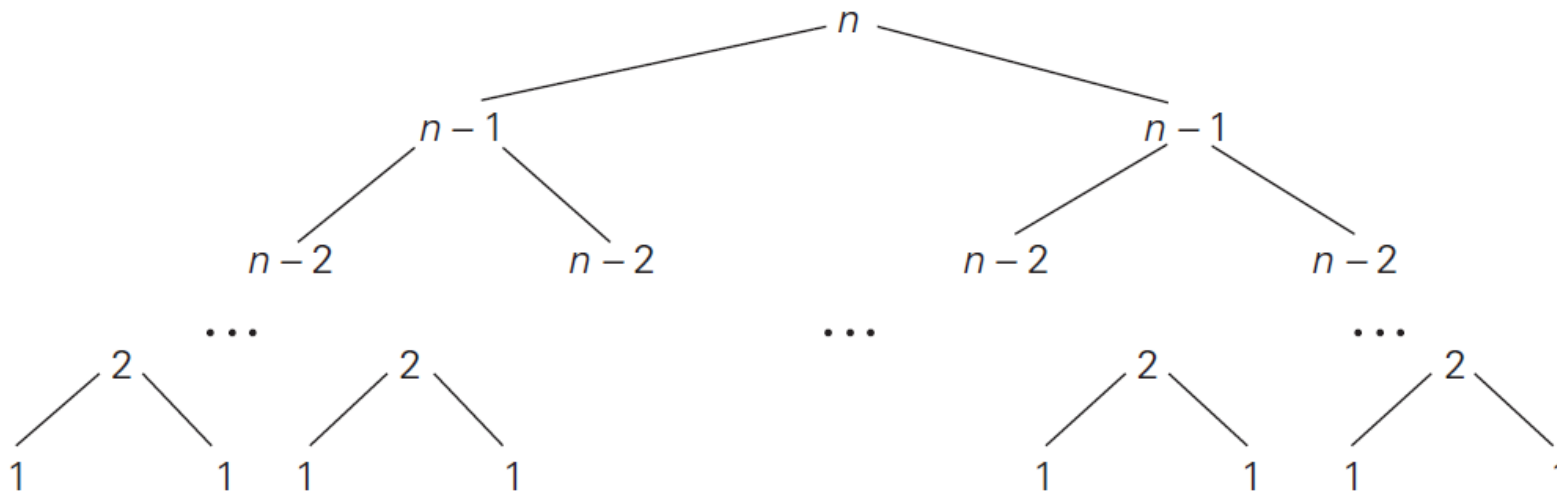
$$= \dots$$

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

$$= 2^{n-1} + 2^{n-2} + \dots + 2^1 + 2^0$$

$$= \mathbf{2^n - 1}$$

Mathematical analysis of Recursive Algorithms VII



Total no. of calls made by the Tower of Hanoi algorithm:

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

l = level of the tree

Thank you!

Any queries?