Analysis of Algorithms

Chapter 2.4, Appendix B

ROAD MAP

- Analysis of algorithms
- Running time functions
- Mathematical Analysis of Nonrecursive Algorithms
- **Mathematical Analysis of Recursive Algorithms**
  - Exact Solution
    - Forward substitution
    - Backward substitution
  - Asymptotic Solution
    - Master theorem

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.

Recurrence Relations

- **Solution Methods**
  - **Exact**
    - Forward substitution
    - Backward substitution
  - **Asymptotic**
    - Master theorem

Example 1 – Factorial Calculation

- Goal:

  Computing the factoral function **`F(n)=n!`** for an arbitrary non-negative integer **`n`**.

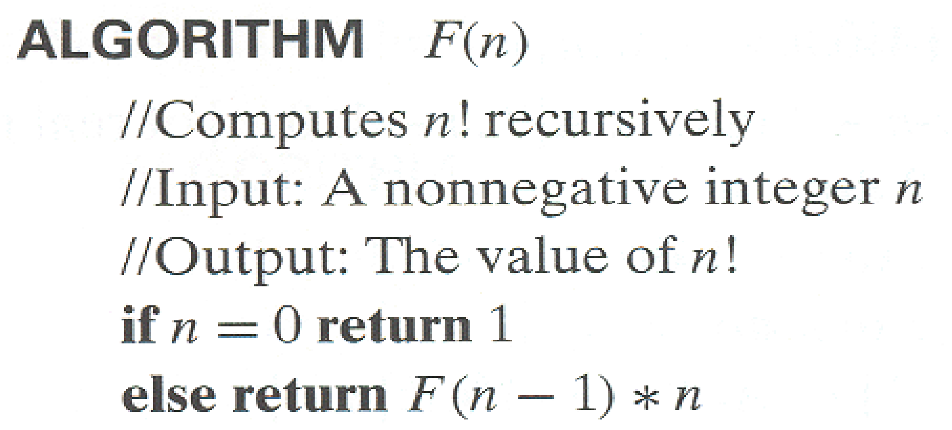
  Since

  $$n! = n\times(n-1)\times\ldots\times 1 = n\times(n-1)! \quad \text{for} \quad n\geq 1 \quad \text{and} \quad 0! = 1$$

  by definition.

  $$F(n) = F(n-1)\times n \quad \text{for} \quad n>0$$
  $$F(0) = 1$$

Example : Recursive evaluation of *n*!



```
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:
Basic operation:
Recurrence relation:

Factorial Calculation

Algorithm :

```
if n=0 return 1
else return F(n-1)*n
```

Analysis:

$$M(n) = \underbrace{M(n-1)}_{\text{to calculate } F(n-1)} + \underbrace{1}_{\text{to multiply } F(n-1) \text{ by } n}$$

$$M(0) = 0$$

ROAD MAP

- Analysis of algorithms
- Running time functions
- Mathematical Analysis of Nonrecursive Algorithms
- **Mathematical Analysis of Recursive Algorithms**
    - Exact Solution
        - Forward substitution
        - Backward substitution
    - Asymptotic Solution
        - Master theorem

Forward Substitution

- Start with the initial condition
- Generate first few terms of the solution
  - Using the recurrence equation and values of previous terms
- Try to guess a pattern
- Form a closed-form formula
- Check the validity of the formula
  - Using induction or
  - Substituting in the recurrence equation and initial condition

Factorial Function

Using forward substitution:

$$M(0) = 0$$

$$M(1) = M(0) + 1 = 1$$

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

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

$$\vdots$$

$$M(n) = n$$

Need to prove the resulting formula

ROAD MAP

- Analysis of algorithms
- Running time functions
- Mathematical Analysis of Nonrecursive Algorithms
- **Mathematical Analysis of Recursive Algorithms**
  - Exact Solution
    - Forward substitution
    - Backward substitution
  - Asymptotic Solution
    - Master theorem

Backward Substitution

- Start with the recurrence equation
- Substitute *f(n-1)*
  - with its value using recurrence equation
- Perform similar substitution few more times
  - for *f(n-2)*, *f(n-3)* etc..
- Try to guess a pattern for *f(n)* in terms of *f(n-i)*
- Check the validity of the pattern
  - usually using induction
- Pick an *i* which makes *n-i* to reach the initial condition
- Obtain a closed-form formula

Factorial Function

Using back substitution:

$$
\begin{aligned}
M(n) &= M(n-1)+1 \\
M(n) &= M(n-2)+1+1 \\
M(n) &= M(n-3)+1+1+1 \\
&\vdots \\
M(n) &= M(n-i)+i
\end{aligned}
$$

for $n = i$

$$M(n) = M(0)+n = n$$

Example 2- Towers of Hanoi

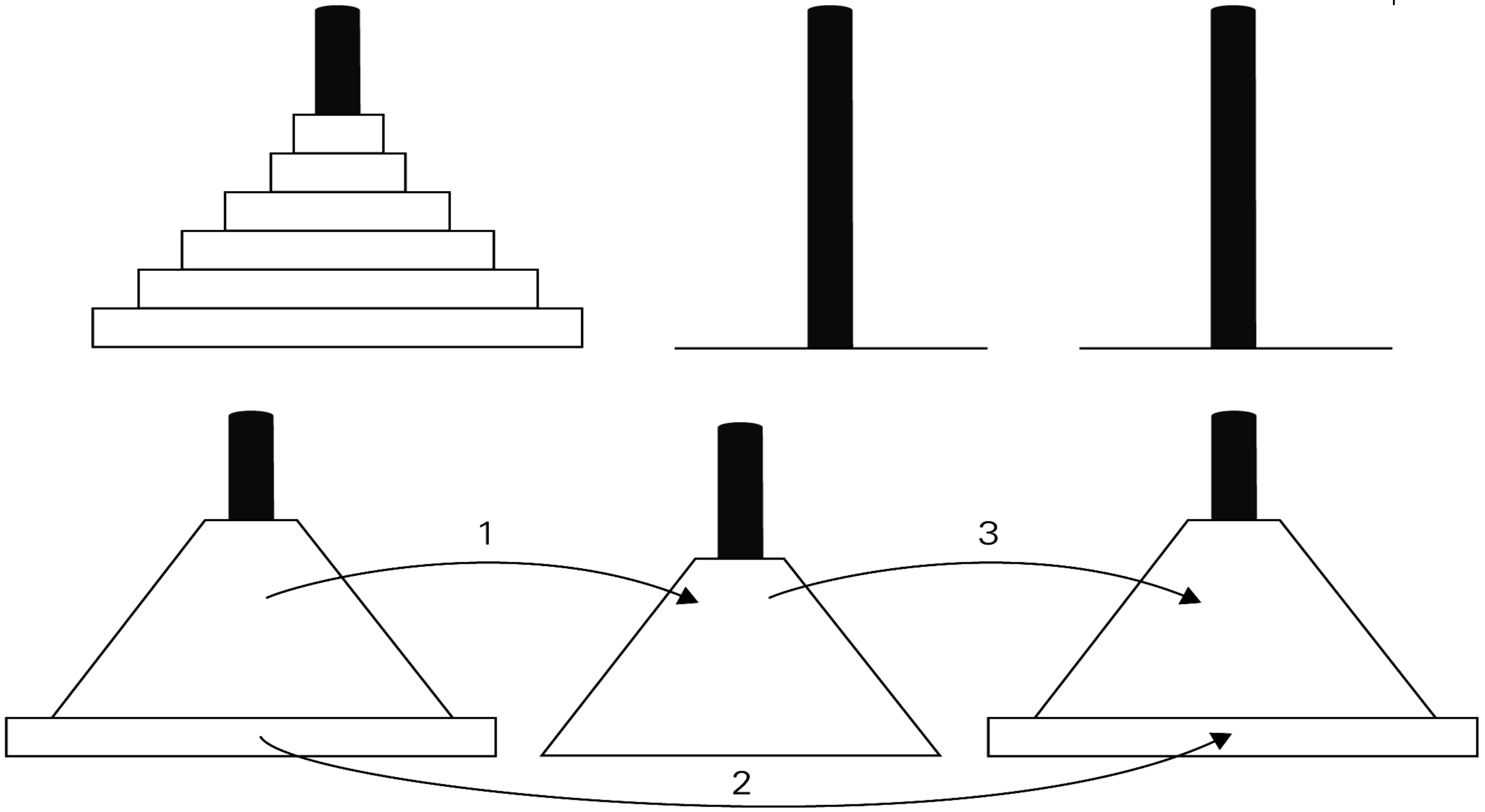


FIGURE 2.4 Recursive solution to the Tower of Hanoi puzzle

Example 2- Towers of Hanoi

<u>Goal:</u>

Transfer *n* disks from peg *A* to peg C using peg B

<u>Approach:</u> (recursive)

- transfer n-1 disks from *A* to *B* using *C*
- move largest disk from *A* to *C*
- transfer n-1 disks from *B* to *C* using *A*

Total number of moves

`T(n) = 2T(n-1) + 1`

`T(1) = 1`

Tree of calls for the Tower of Hanoi Puzzle

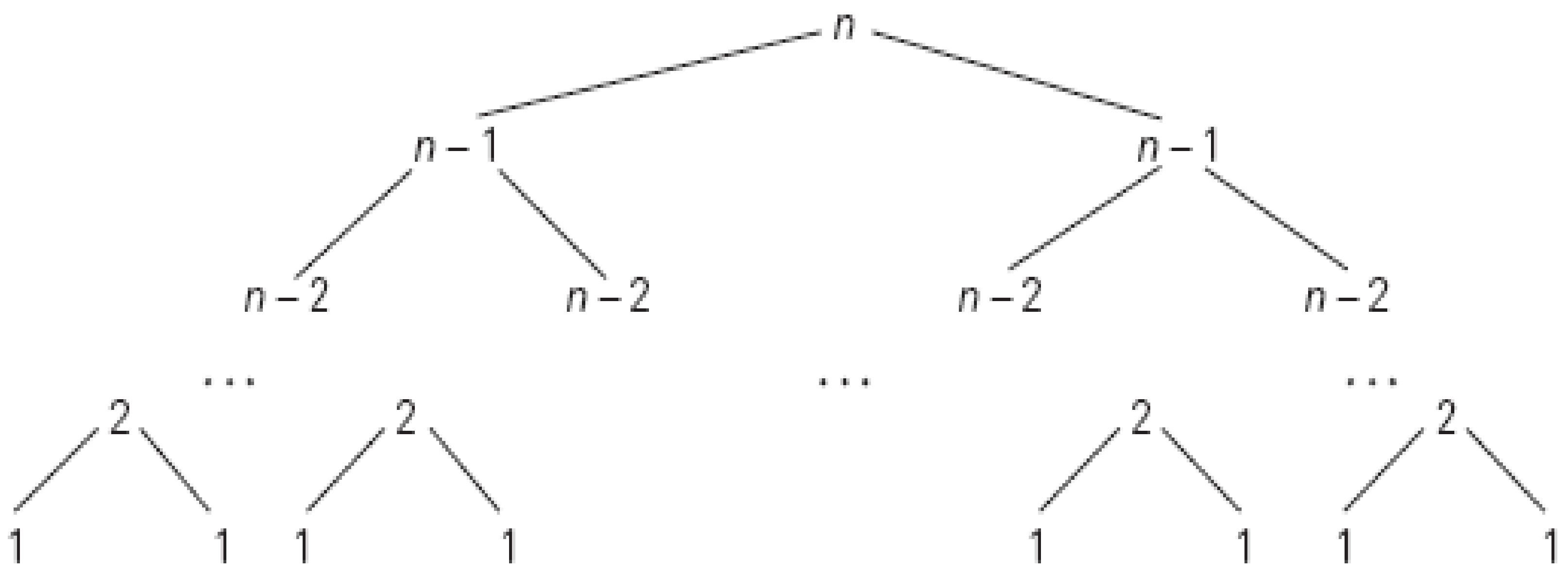


FIGURE 2.5 Tree of recursive calls made by the recursive algorithm for the Tower of Hanoi puzzle.

Towers of Hanoi

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

$$T(1) = 1$$

Solution by backward substitution

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

$$T(n) = 4\big(T(n-2)\big) + 2 + 1$$

$$T(n) = 4\big(2T(n-3) + 1\big) + 2 + 1$$

$$T(n) = 8T(n-3) + 4 + 2 + 1$$

$$\vdots$$

$$T(n) = 2^i T(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 2^1 + 1$$ **← needs proving**

$$when \qquad i = n-1$$

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

$$T(n) = 2^n - 1 = \theta(2^n)$$

Example

$$T(n) = 2T(\sqrt{n}) + 1 \qquad T(2) = 0$$

$$T(n) = 2T(n^{1/2}) + 1$$

$$T(n) = 2\left(2T(n^{1/4}) + 1\right) + 1$$

$$T(n) = 4T(n^{1/4}) + 1 + 2$$

$$T(n) = 2^3 T(n^{1/8}) + 1 + 2 + 2^2$$

$$\vdots$$

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

$$n^{1/2^i} = 2 \quad \Rightarrow \quad i = \log\log(n)$$

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

ROAD MAP

- Analysis of algorithms
- Running time functions
- Mathematical Analysis of Nonrecursive Algorithms
- **Mathematical Analysis of Recursive Algorithms**
  - Exact Solution
    - Forward substitution
    - Backward substitution
  - Asymptotic Solution
    - Master theorem

Master Theorem

Let T(n) be an eventually nondecreasing function that satisfies the recurrence

$$T(n) = a\,T(n/b) + f(n) \qquad \text{for } n = b^k, k = 1, 2, \ldots$$
$$T(1) = c,$$
$$where\ a \geq 1,\ b \geq 2, c > 0.$$
$$If\ f(n) \in\ \Theta\left(n^d\right)\ where\ d \geq 0, then$$

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d, \\ \Theta(n^d \log n) & \text{if } a = b^d, \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

Examples

$$T(n) = T(n/2) + c$$

$$T(n) = 2\,T(n/2) + c\,n$$

$$T(n) = 7\,T(n/2) + 18\,n^2$$

$$T(n) = 9\,T(n/3) + n$$

$$T(1) = \varepsilon$$

Common Recurrence Types in Algorithm Analysis

- Decrease-by-One

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

- Decrease-by-a-Constant-Factor

$$T(n) = T\left({}^{n}/_{b}\right) + f(n)$$

- Divide-and-Conquer

$$T(n) = a\,T\left({}^{n}/_{b}\right) + f(n)$$