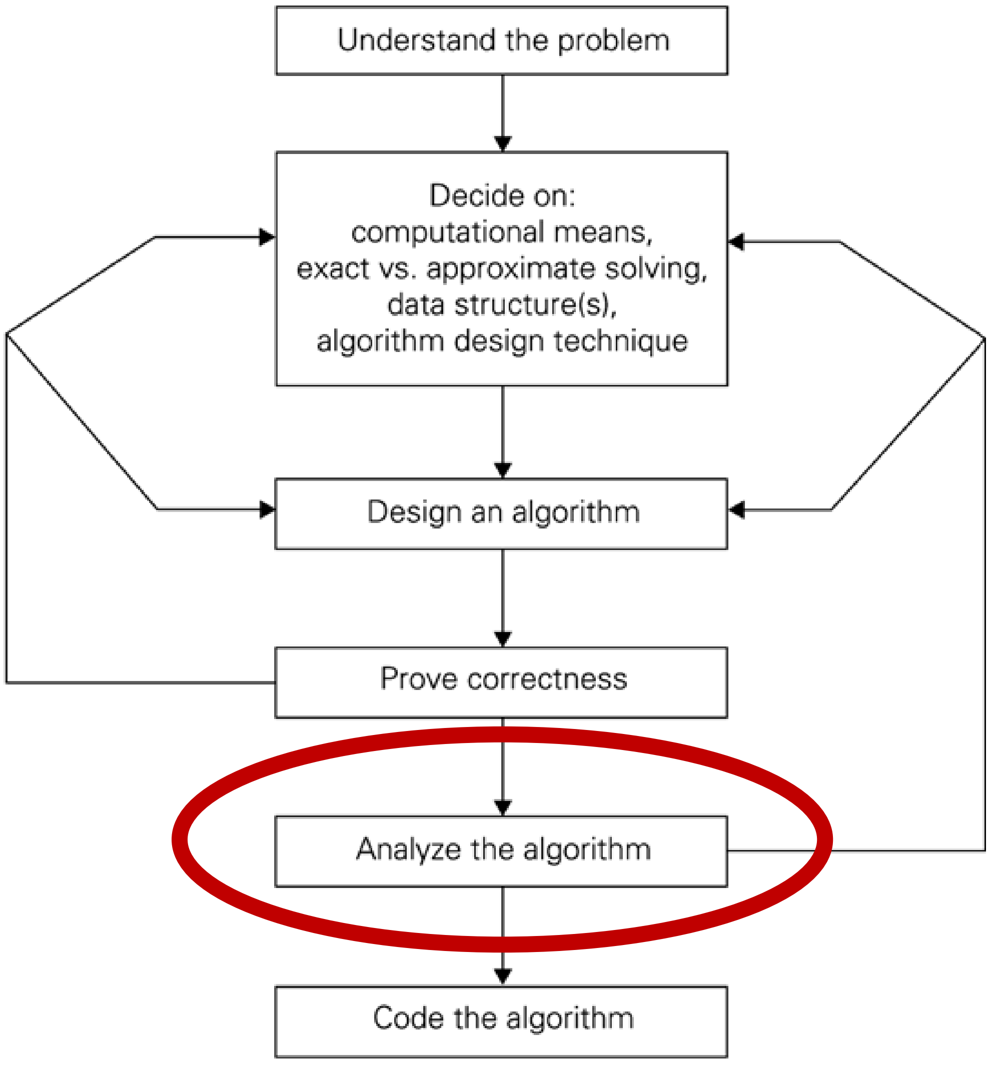
Growth of Functions

Chapter 2 & 3

Mei-Chen Yeh

What is a *good* algorithm?

Algorithm design and analysis process



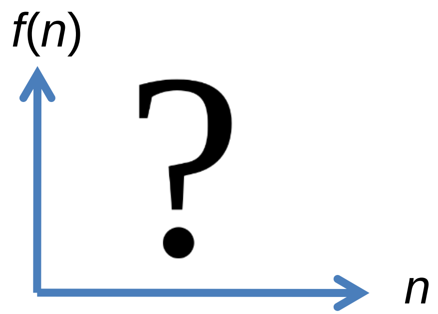
Analysis Framework

- Run time vs. input size

  Run time = *f*(input size)

- # Basic operations
- Order of growth

n $\log n$ $n\log n$ 2^n $n!$ n^2 n^3



n	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3\cdot10^1$	10^2	10^3	10^3	$3.6\cdot10^6$
10^2	6.6	10^2	$6.6\cdot10^2$	10^4	10^6	$1.3\cdot10^{30}$	$9.3\cdot10^{157}$
10^3	10	10^3	$1.0\cdot10^4$	10^6	10^9		
10^4	13	10^4	$1.3\cdot10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7\cdot10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0\cdot10^7$	10^{12}	10^{18}		

Analysis Framework (cont.)

- Worst-case, best-case, average-case

ALGORITHM *Mystery*(A[0..n-1], K)

$i \leftarrow 0$
while $i < n$ **and** $A[i] \neq K$ **do**
    $i \leftarrow i + 1$
if $i < n$ **return** i
else return -1

Pseudo code

Average-case
is or **isn't**
the average of worst and best case?

Analysis Framework (cont.)

- Time (Space) efficiency is measured as a function of the algorithm's input size.
- Time efficiency is measured by counting the number of times the basic operation is executed.
- Need to distinguish between the worst-case, average-case, and best-case efficiencies.

Run time = *f*(∞)

Asymptotic Notations

O (big oh) Ω(big omega) Θ(big theta)

Characterize the order of growth of an algorithm’s basic operation count

Asymptotic Notations (cont.)

- *O*-notation (upper bound)

  $O(g(n))$ is the set of functions with a smaller or same order of growth as $g(n)$.

  True or ***false***?

$$n \in O(n^2)$$

$$100n + 5 \in O(n^2)$$

$$0.5n(n-1) \in O(n^2)$$

$$0.00001n^3 \in O(n^2)$$

Asymptotic Notations (cont.)

- Definition:

$O(g(n)) = \{ f(n) :$ there exist constants $c > 0,\ n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0 \}$

- Example: $\underline{100n + 5} \in O(\underline{n^2})$

$$100n + 5 \le 100n + n \ (\mathit{for\ all}\ n \ge 5) = 101n \le 101n^2$$

n_0 $\quad$ c

Asymptotic Notations (cont.)

- Ω-notation (lower bound)

  Ω(*g*(*n*)) is the set of functions with a larger or same order of growth as *g*(*n*).

$$n^3 \in \Omega(n^2)$$

$$0.5n(n-1) \in \Omega(n^2)$$

$$100n + 5 \in \Omega(n^2)$$

Asymptotic Notations (cont.)

- Definition:

$$\Omega(g(n)) = \{ f(n) : \text{there exist constants } c > 0,\ n_0 > 0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0 \}$$

- Example: $n^3 \in \Omega(n^2)$

Select $c = 1$ and $n_0 = 1$

Asymptotic Notations (cont.)

- Θ-notation

  Θ(*g*(*n*)) is the set of functions that have the same order of growth as *g*(*n*).

$$5n^2 + 2n + 9 \in \Theta(n^2)$$

Asymptotic Notations (cont.)

- Definition:

$\Theta(g(n)) = \{ f(n)$: there exist constants $c_1 > 0$, $c_2 > 0$, $n_0 > 0$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$

- Example: $0.5n(n-1) \in \Theta(n^2)$

Select $c_1 = 0.25$, $c_2 = 0.5$ and $n_0 = 2$

o- and ω- notations

- O-notation and Ω-notation are like ≤ and ≥.
- o-notation and ω-notation are like < and >.
- Example:

$$2n \in O(n^2), \quad but\ 2n^2 \notin o(n^2)$$

If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$

$f_1(n) + f_2(n) \in O(?)$

(1) $g_1(n) + g_2(n)$

(2) $\max\{g_1(n), g_2(n)\}$

(3) $\min\{g_1(n), g_2(n)\}$

(4) $0.5\,(g_1(n) + g_2(n))$

Rank the following functions in terms of their order of growth

n^2 $\log n$ n 1 $n!$ $n \log n$ 2^n n^3

1 $\log n$ n $n \log n$ n^2 n^3 2^n $n!$

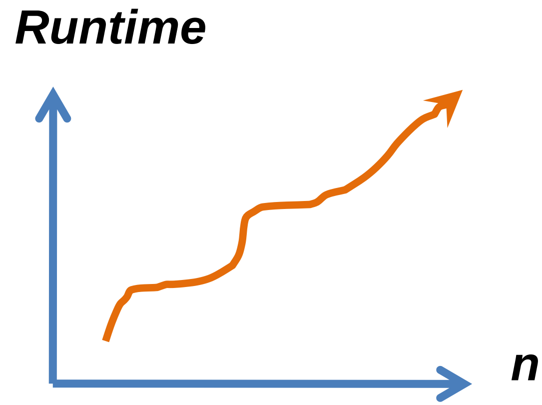
Practice (1)

- Order of growth:

$$n! > 2^n > n^3 > n^2 > n \log n > n > \log n > 1$$

$\sqrt{n}$

4^n



Practice (2)

$\log_2 n \quad vs. \quad \log_{100} n \quad ?$

Recall $\log_b a = \dfrac{\log_c a}{\log_c b}$

$$\log_2 n = \frac{\log_{100} n}{\log_{100} 2}$$

$$= \frac{1}{\log_{100} 2} \times \log_{100} n$$

$$= 6.643 \times \log_{100} n$$

a constant!

Practice (3)

- Function(s) ***not*** in $O(n^2)$?

n^2	n
$n^2 + n$	$n/1000$
$n^2 + 1000n$	$n^{1.99999}$
$1000n^2 + 1000n$	$n^2 / \mathrm{lglglg}\, n$

What's the message?

- Given n numbers, the time complexity of my algorithm for sorting them is $O(n^2)$.
- Given n numbers, the time complexity of my algorithm for sorting them is $\Theta(n^2)$.
- Given n numbers, the time complexity of my algorithm for sorting them is $\Omega(n^2)$.

- Analysis examples
- Three methods for solving recurrences

Analysis Examples

Example (1)

ALGORITHM *X-Algorithm*(A[0..n-1])

```
val ← A[0]
for i ← 1 to n-1 do
    if A[i] > val
        val ← A[i]
return val
```

Basic operation?

Complexity?

Example (2)

ALGORITHM *X-Algorithm*(n)

count $\leftarrow$ 1
while $n > 1$ **do**
  count $\leftarrow$ *count* + 1
  $n \leftarrow \text{floor}(n/2)$
return *count*

Complexity?

Example (3)

ALGORITHM $F(n)$

if n=0 **return** 1
else return $F(n\text{-}1)*n$

To compute $F(n\text{-}1)$

To multiply $F(n\text{-}1)$ by n

Complexity?

The recurrence relation: $M(n) = M(n\text{-}1) + 1$ for $n>0$

The initial condition: $M(0) = 0$

[Backward substitution]

Example (3)

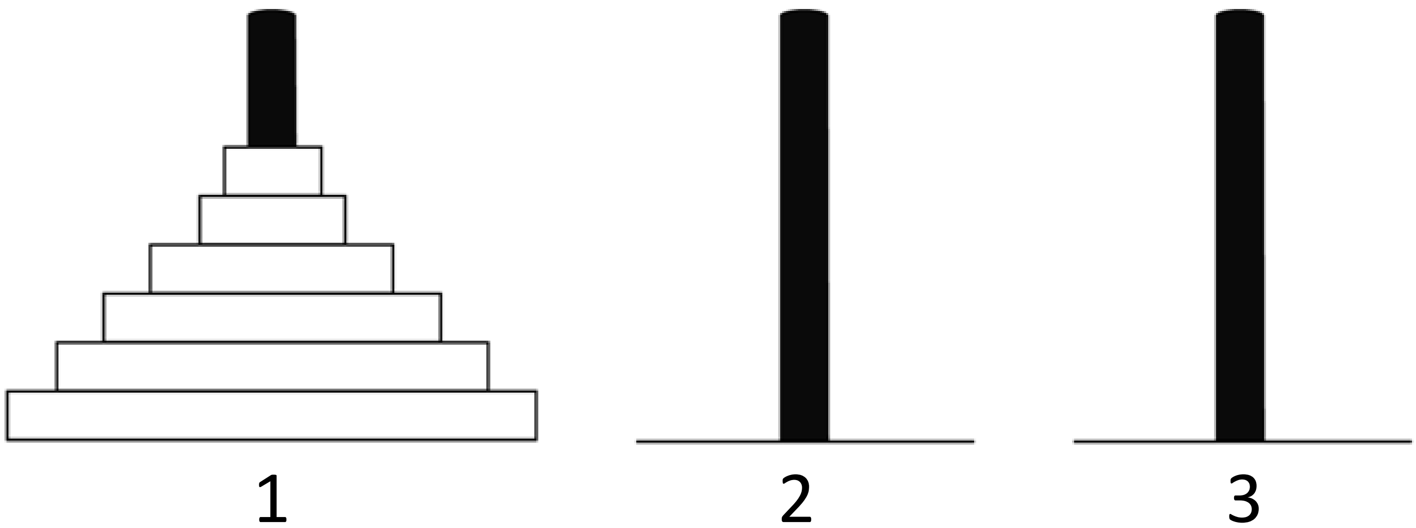
- Backward substitutions

$$M(n) = M(n-1)+1$$
$$= [M(n-2)+1] + 1 = M(n-2)+2$$
$$= [M(n-3)+1] + 2 = M(n-3)+3$$
$$\vdots$$
$$= \text{? ? ?}$$

Solving Recurrences

Chapter 4.3 - 4.5

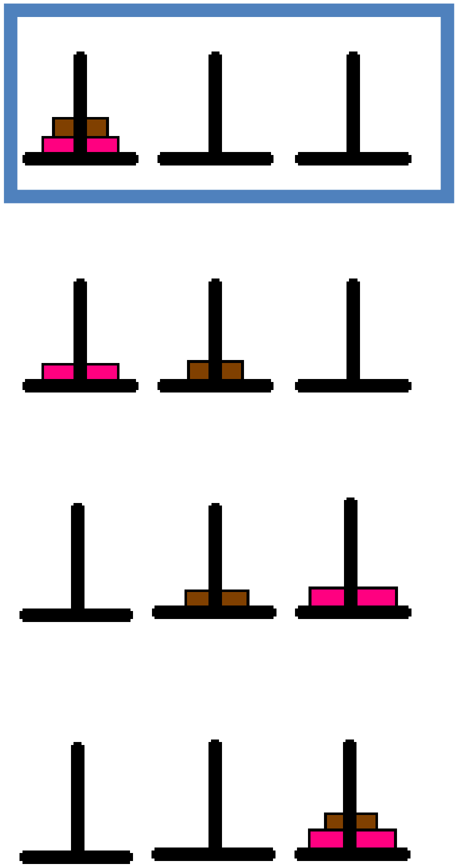
Tower of Hanoi



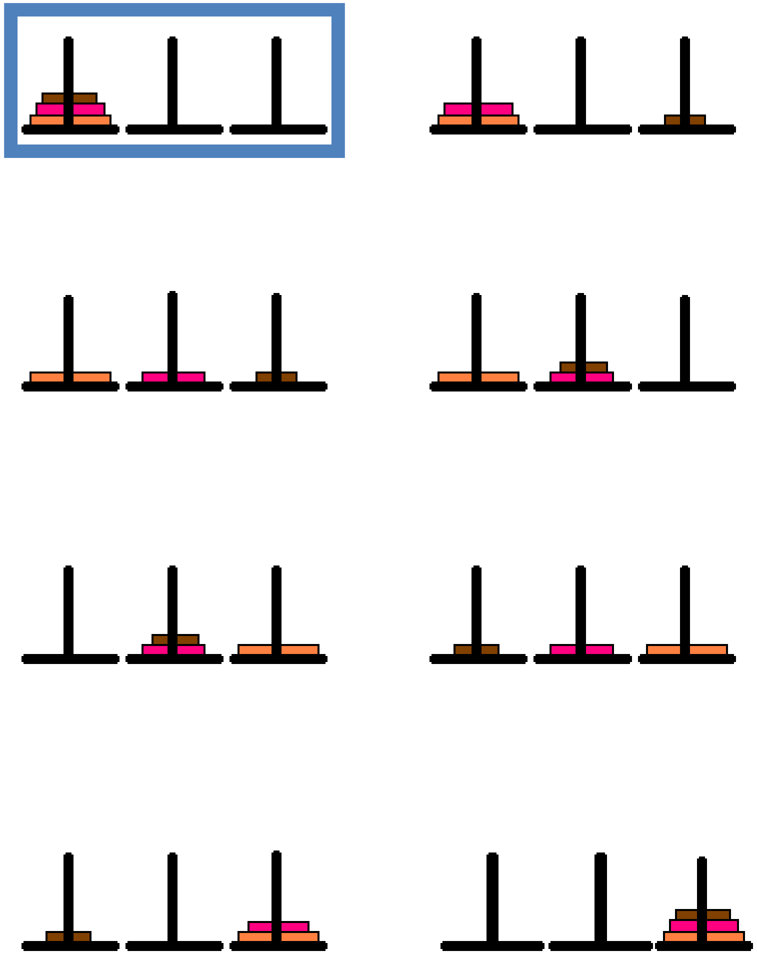
How many moves are required to move disks from peg 1 to peg 3?

Tower of Hanoi

- $n = 2$

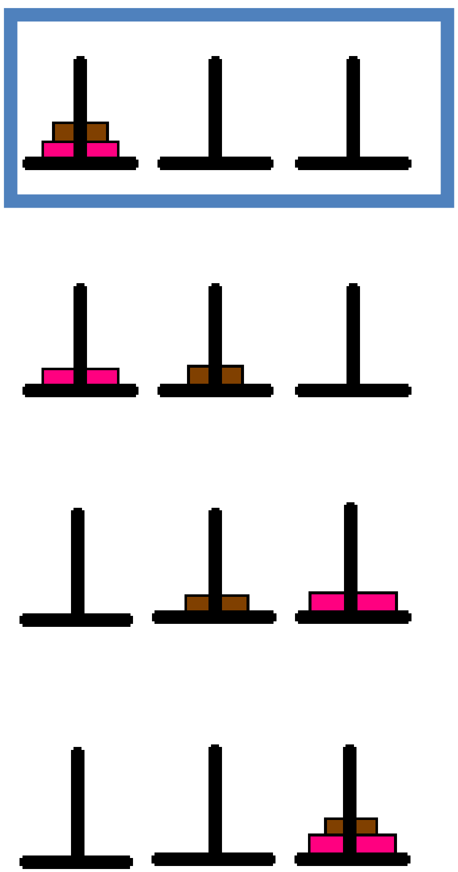


- $n = 3$

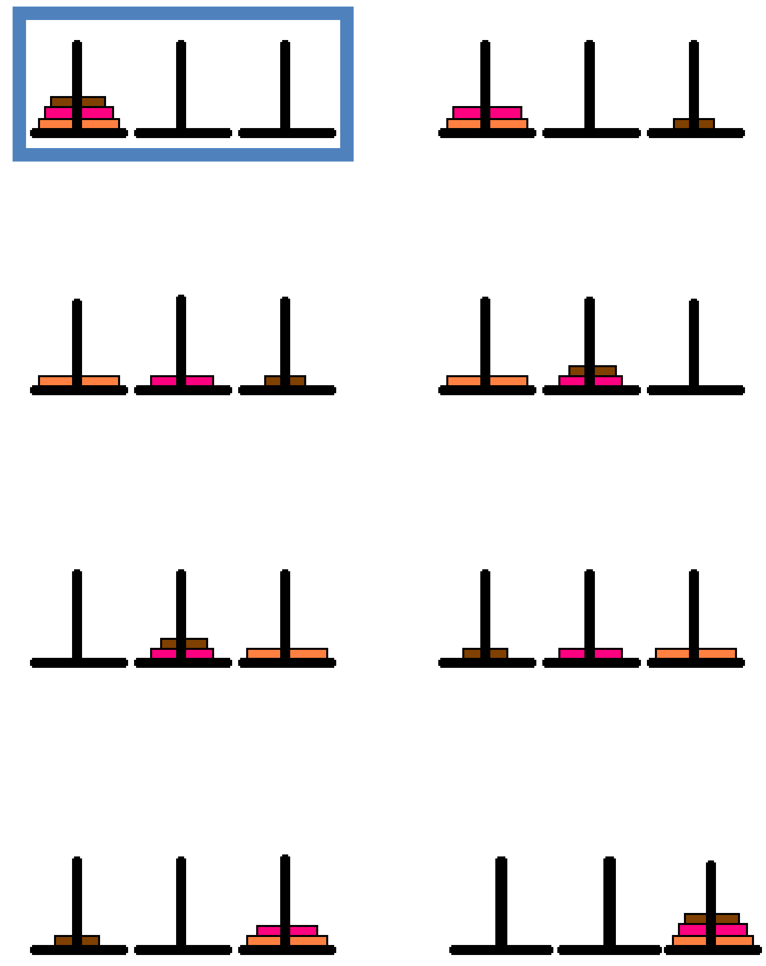


#moves vs. #disks?

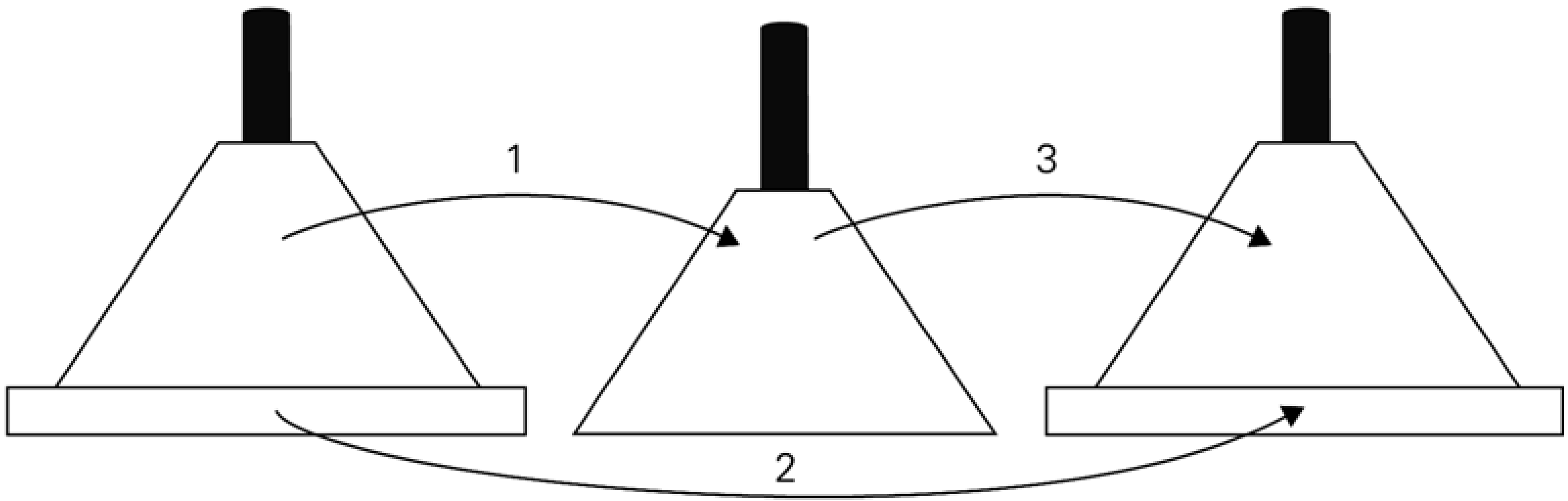
- $n = 2$



- $n = 3$



$M(n)$ = The number of required moves given n disks



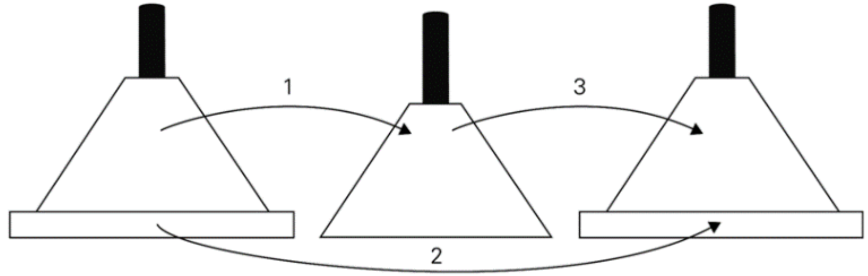
Recurrence:

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

$M(1) = 1$

$M(n) = \Theta(\textbf{?})$

1. n
2. $n\log n$
3. n^2
4. 2^n



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

$M(1) = 1$

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

$= 2[2M(n\text{-}2)+1] + 1 = 4M(n\text{-}2) + 3$

$= 4[2M(n\text{-}3)+1] + 3 = 8M(n\text{-}3) + 7$

$\vdots$

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

Exponential algorithm!

$\#$ total moves $= 2^{n\text{-}1}M(1) + 2^{n\text{-}1} - 1 = 2^n - 1 = \Theta(2^n)$

Fibonacci Numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

Recurrence:

$F(n) = F(n\text{-}1) + F(n\text{-}2)$ for $n>1$

$F(0) = 0$, $F(1) = 1$

$F(n) = \Theta(\textbf{?})$

1. Linear n
2. Polynomial n^c
3. Exponential c^n
4. Factorial $n!$

- Analysis examples
- Three methods for solving recurrences

Solving recurrences

- $T(n) = 16T(n/4) + n^2$
- $T(n) = 7T(n/3) + n^2$
- $T(n) = T(n/2) + T(n/4) + T(n/8) + n$
- $T(n) = 2T(n/4) + 1$

Solving recurrences

- The substitution method
  - Forward substitution
  - Backward substitution
- The recursion tree method
- The master method

Backward substitution

- Recurrence

$$x(n) = x(n\text{-}1) + n$$
$$x(1) = 1$$

- Solution

$$\begin{aligned} x(n) &= x(n\text{-}1) + n \\ &= [x(n\text{-}2)+n\text{-}1] + n \\ &= [x(n\text{-}3)+n\text{-}2] + (n\text{-}1) + n \\ &= x(n\text{-}i) + (n\text{-}i+1) + (n\text{-}i+2) + \ldots + n \end{aligned}$$

$$x(n) = 1+2+3+\ldots+n = n(n+1)/2 = \Theta(n^2)$$

Forward substitution

- Recurrence

$$x(n) = 2x(n\text{-}1) + 1 \text{ for } n>1$$

$$x(1) = 1$$

- Solution

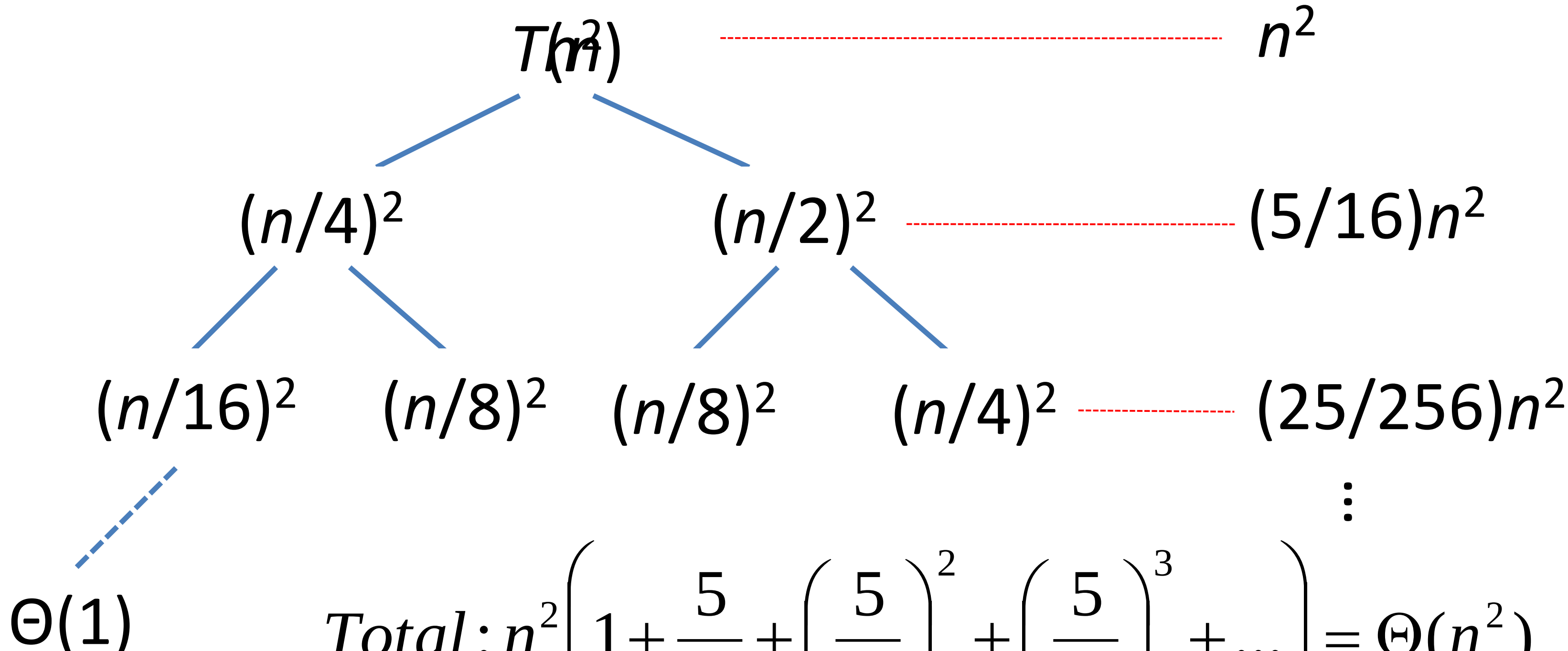
$$x(1) = 1$$
$$x(2) = 2*1 + 1 = 3$$
$$x(3) = 2*3 + 1 = 7$$
$$x(4) = 2*7 + 1 = 15$$

$$x(n) = 2^n\text{-}1 = \Theta(2^n)$$

Recursion tree

- Recurrence

$$T(n) = T(n/4) + T(n/2) + n^2$$



$$Total: n^2\left(1+\frac{5}{16}+\left(\frac{5}{16}\right)^2+\left(\frac{5}{16}\right)^3+\ldots\right)=\Theta(n^2)$$

The master theorem

- The master method applies to recurrences of the form

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

where $a \geq 1$, $b > 1$, and f is asymptotically positive.

(An asymptotically positive function is one that is positive for all sufficiently large n.)

Examples

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

- $T(n) = 4T(n/2) + n$
- $T(n) = 2T(n/3) + n^3$
- $T(n) = 5T(3n/4) + n^2$
- ……

$$f(n) \text{ vs. } n^{\log_b a}$$

The master theorem: three cases

$T(n) = aT(n/b) + f(n)$ $f(n)$ vs. $n^{\log_b a}$

– Case 1: $f(n) < n^{\log_b a}$

$$\boxed{T(n) = \Theta(n^{\log_b a})}$$

and polynomially smaller!

$f(n) = O(n^{\log_b a - \varepsilon})$ *for some* $\varepsilon > 0$

– Case 2: $f(n) = n^{\log_b a}$

$$\boxed{T(n) = \Theta(n^{\log_b a} \lg n)}$$

– Case 3: $f(n) > n^{\log_b a}$

$$\boxed{T(n) = \Theta(f(n))}$$

and polynomially larger!

$f(n) = \Omega(n^{\log_b a + \varepsilon})$ *for some* $\varepsilon > 0$

and satisfy the regularity condition!

$af(n/b) \le cf(n)$ *for some* $c < 1$

The master theorem: examples

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

- $T(n) = 4T(n/2) + n$
  - $a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2$
  - $f(n) = n \Rightarrow f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$

CASE 1!

$\boldsymbol{T(n) = \Theta(n^2)}$

The master theorem: examples

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

- $T(n) = 4T(n/2) + n^2$
  - $a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$
  - $f(n) = n^2$

CASE 2!

$\boldsymbol{T(n) = \Theta(n^2 \lg n)}$

The master theorem: examples

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

The regularity condition:

$af(n/b) \le cf(n) \quad for\ some\ c < 1$

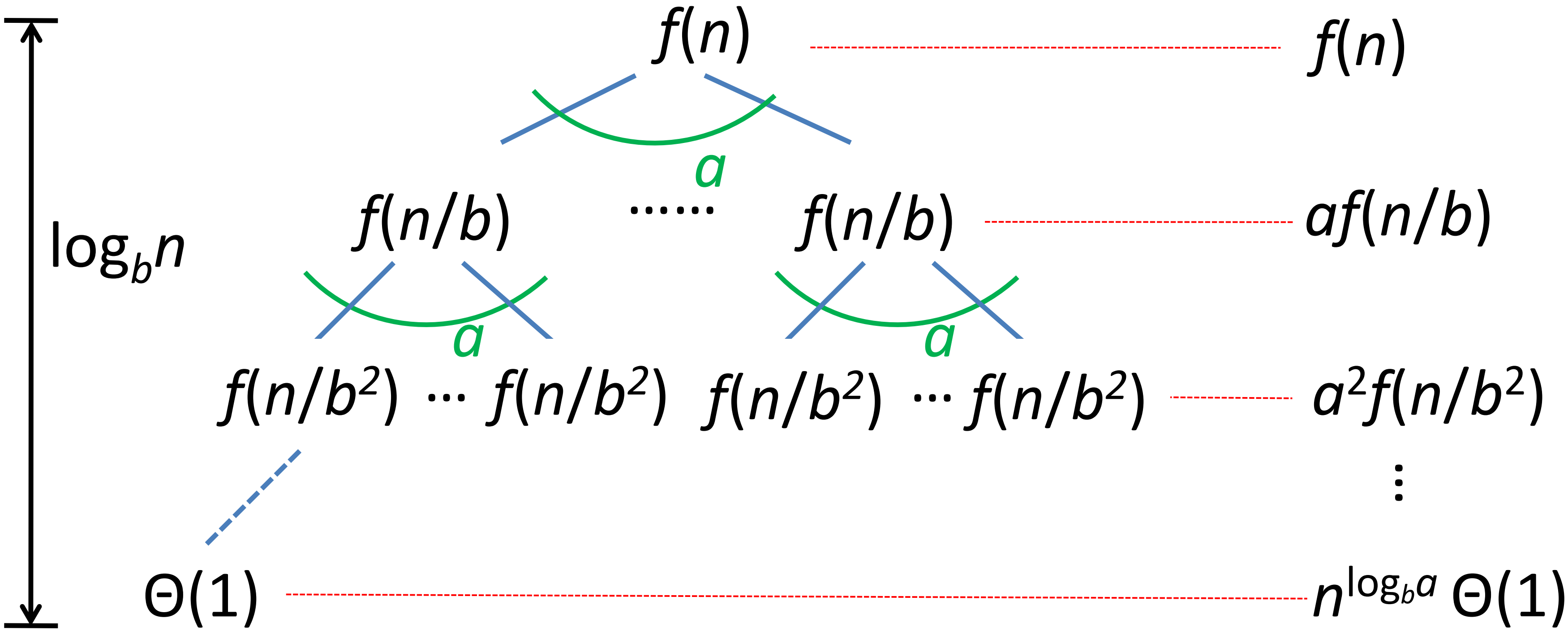
- $T(n) = 4T(n/2) + n^3$
  - $a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2$
  - $f(n) = n^3 \Rightarrow f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$
    - Check the regularity condition: $4(n/2)^3 \le cn^3$ for $c = 1/2$

CASE 3!

$\boldsymbol{T(n) = \Theta(n^3)}$

Idea of the master theorem

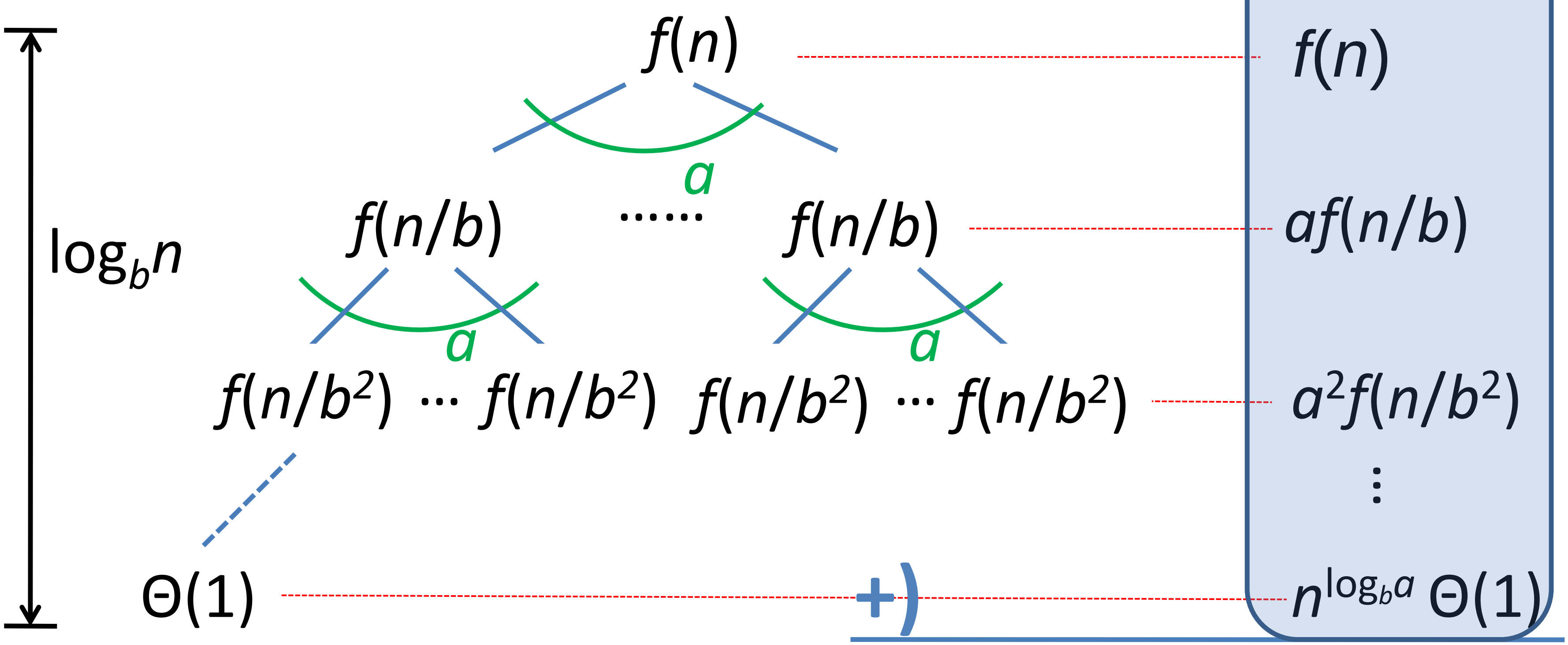
$T(n) = aT(n/b) + f(n)$



leaves $= a^h = a^{\log_b n} = n^{\log_b a}$

Idea of the master theorem

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



Case 1 ($f(n) < n^{\log_b a}$): dominated by the cost in the leaves

Case 2 ($f(n) = n^{\log_b a}$): evenly distributed among levels of the tree

Case 3 ($f(n) > n^{\log_b a}$): dominated by the costs of the root

Idea of the master theorem

- Case 1: $f(n) < n^{\log_b a} \Rightarrow \Theta(n^{\log_b a})$
  - The weight increases geometrically from the root to the leaves.
- Case 2: $f(n) = n^{\log_b a} \Rightarrow \Theta(n^{\log_b a} \lg n)$ or $\Theta(f(n) \lg n)$
  - The weight is approximately the same on each level.
- Case 3: $f(n) > n^{\log_b a} \Rightarrow \Theta(f(n))$
  - The weight decreases geometrically from the root to the leaves.

Coming up

- Analysis examples
- Three methods for solving recurrences
- Divide and conquer (Chapter 4.1 – 4.2)