CS3230 Lecture 3

NUS National University of Singapore
School of Computing

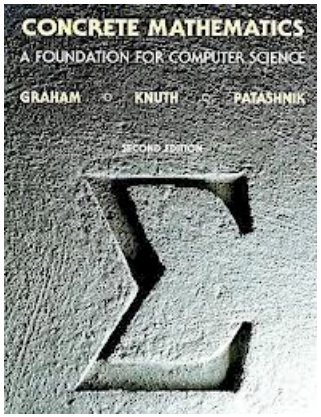
"Asymptotics, Summation, Divide & Conquer Algorithms, Recurrences and Master Theorem"

- **Lecture Topics and Readings**
  - Asymptotic, Summation [CLRS]-C3
  - Divide and Conquer Algorithms [CLRS]-C2
  - Recurrences, Master Theorem [CLRS]-C4

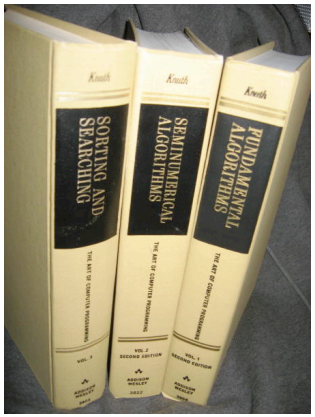
Algorithms & Discrete Math have very happy, stable marriage!

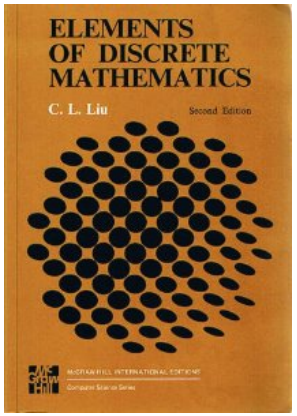
Early pioneers of Math for CS

Don Knuth
Stanford



C. L. Liu,
MIT, UIUC,
NTHU (tw)

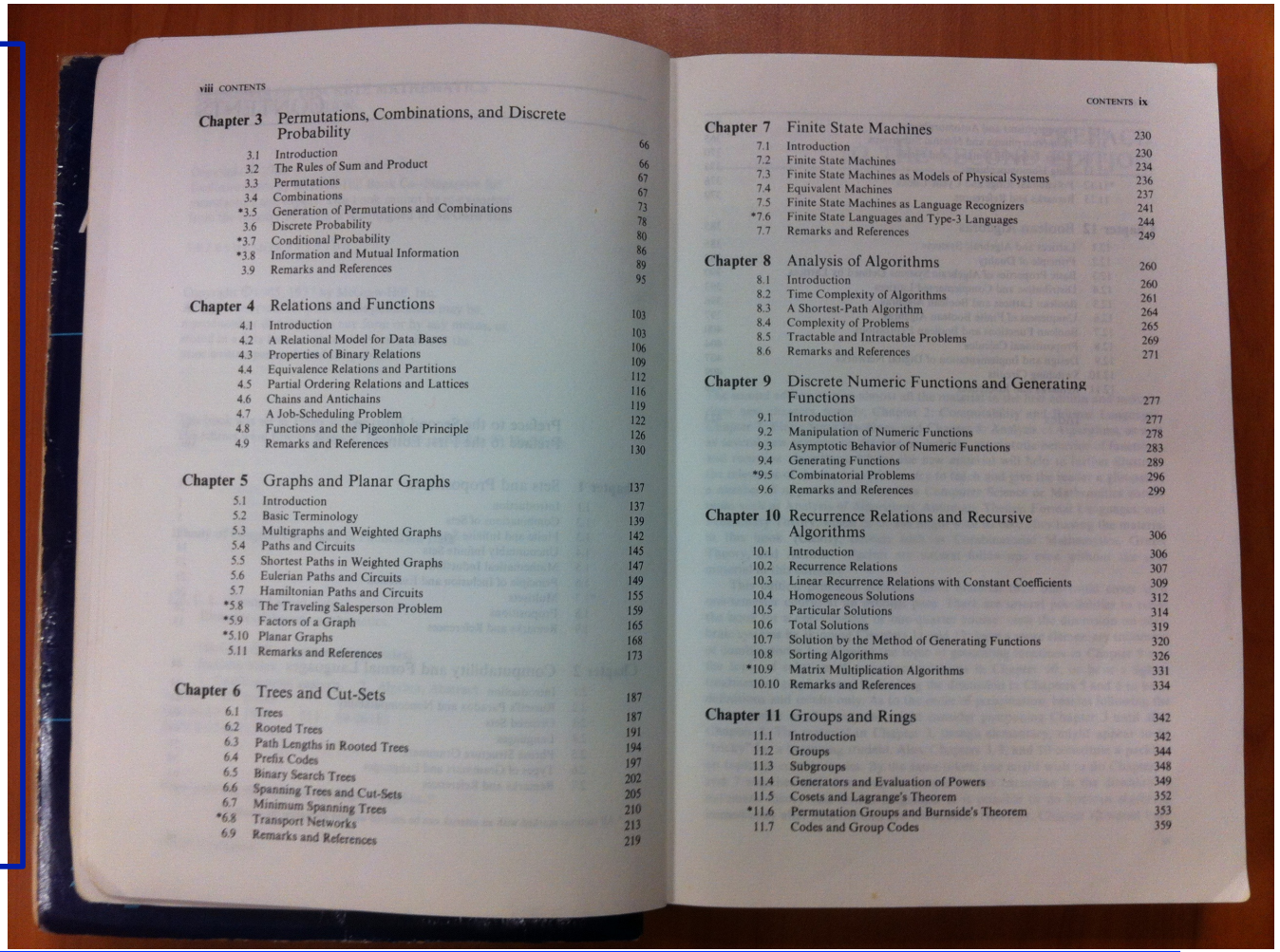




Elements of Discrete Math, 1977

"Elements of Discrete Mathematics," C. L. Liu, McGraw-Hill, (1977), has the most important topics in DM for CS.

- **Relations & Functions**
- **Graphs & Planar Graphs**
- **Trees and Cut-Sets**
- **Finite State Machines**
- **Analysis of Algorithms**
- **Recurrence Relations**
- . . .



viii CONTENTS

CONTENTS ix

David Plaisted

David Plaisted
UIUC 1978-84
UNC, 1985--

Spring 1980, LeongHW took CS373 by Plaisted;

Summer 1984, Plaisted: UIUC ----> UNC

Spring 1985 @UIUC: CS373 by Plaisted

Dave Liu recommend PhD student LeongHW to the Dept Head for CS373

Herbrand Award, 2010
(for distinguished contribution to Automated Reasoning)

http://www.cs.miami.edu/~geoff/Conferences/CADE/HerbrandAward.html

Volker Strassen

Volker Strassen

Knuth Prize, 2009

"seminal and influential contributions
To the design and analysis
of efficient algorithms"

http://en.wikipedia.org/wiki/Volker_Strassen

Thank you.

Q & A

CS3230 Lecture 3

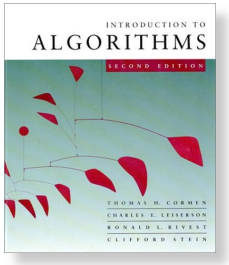
NUS National University of Singapore
School of Computing

"Asymptotics, Summation, Divide & Conquer Algorithms, Recurrences and Master Theorem"

❑ Lecture Topics and Readings

- ❖ Asymptotics, Summation [CLRS]-C3
- ❖ Divide and Conquer Algorithms [CLRS]-C2
- ❖ Recurrences, Master Theorem [CLRS]-C4

Algorithms & Discrete Math have very happy, stable marriage!

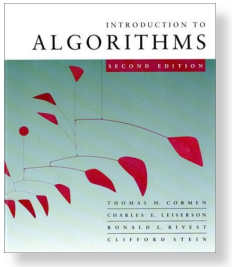


Merge sort

MERGE-SORT $A[1 \ldots n]$

1. If $n = 1$, done.
2. Recursively sort $A[\,1 \ldots \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \ldots n\,]$.
3. "***Merge***" the 2 sorted lists.

Key subroutine: **MERGE**



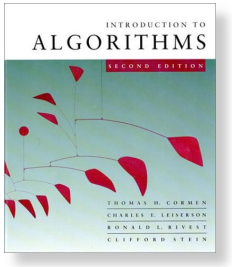
Merging two sorted arrays

20 12

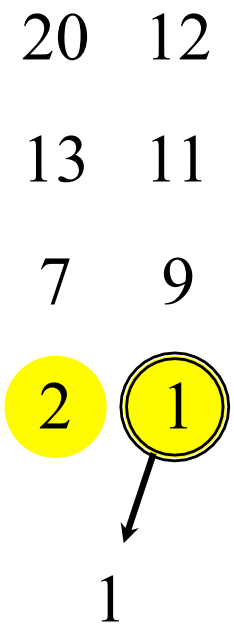
13 11

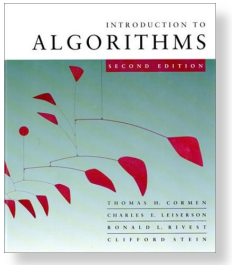
7 9

2 1

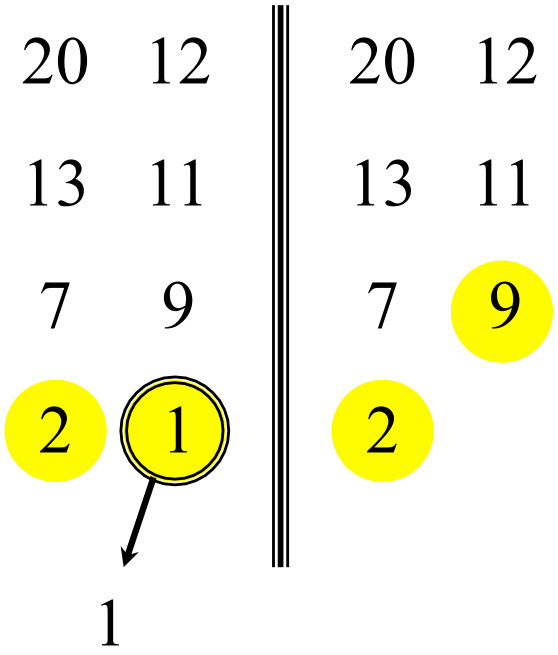


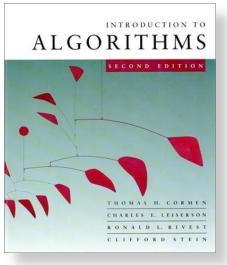
Merging two sorted arrays



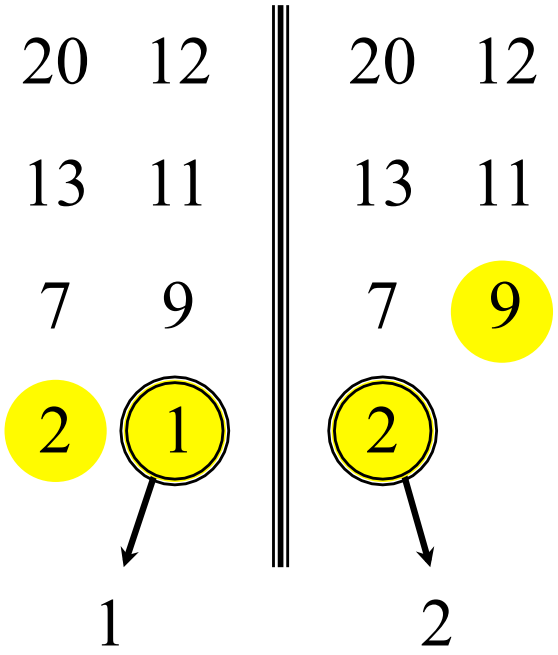


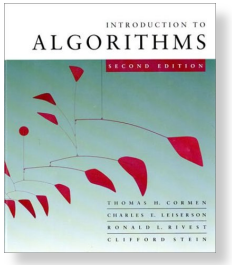
Merging two sorted arrays



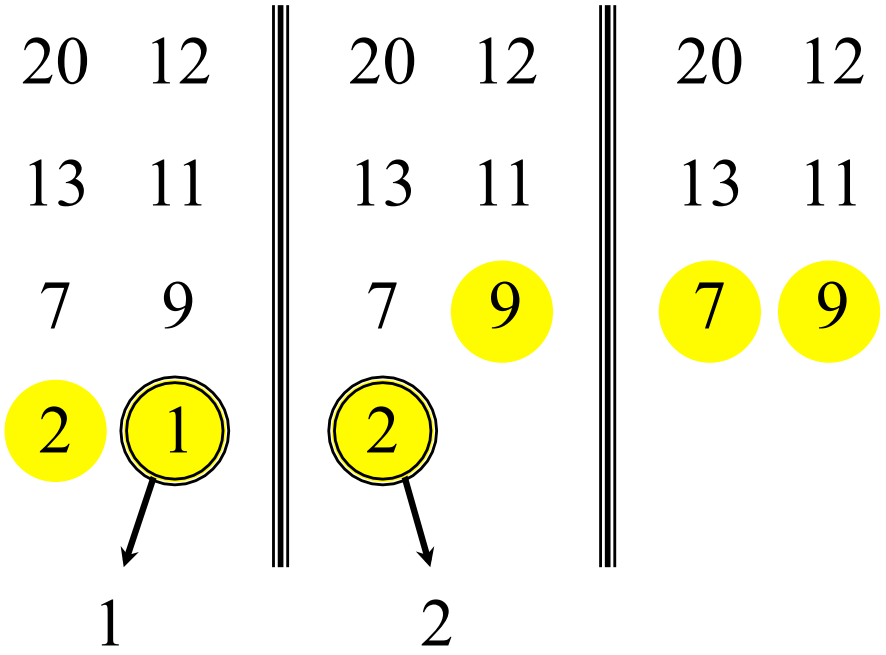


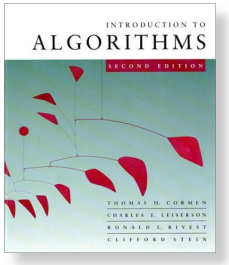
Merging two sorted arrays



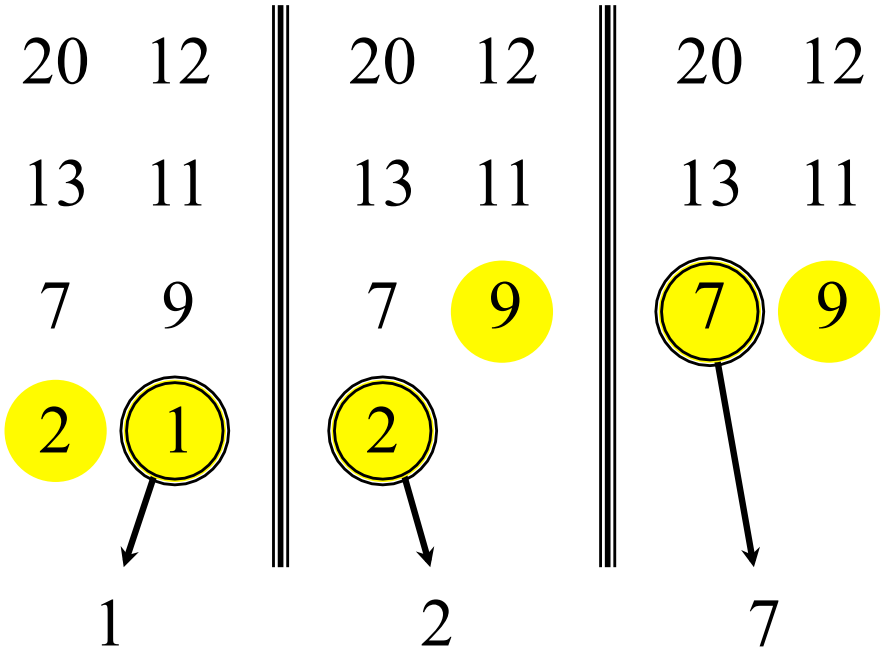


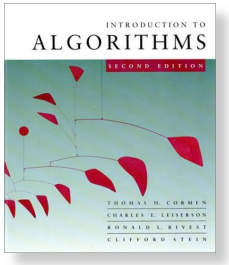
Merging two sorted arrays



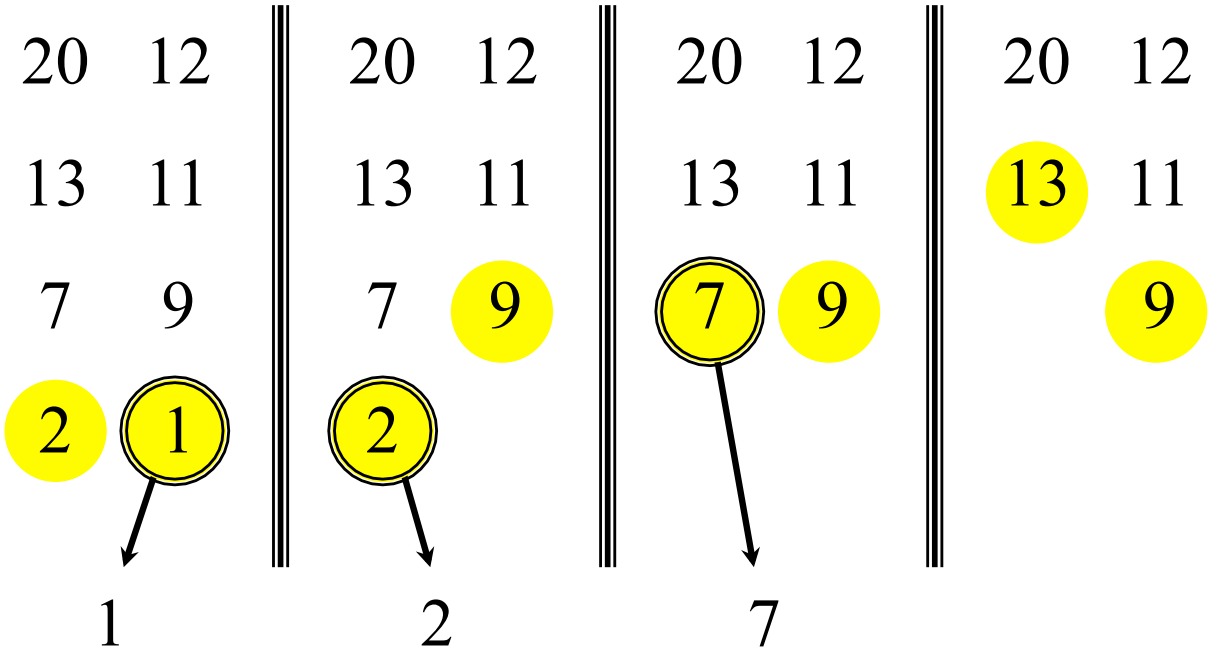


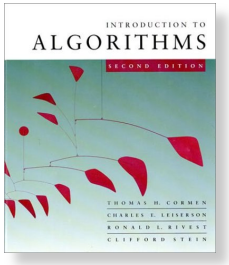
Merging two sorted arrays



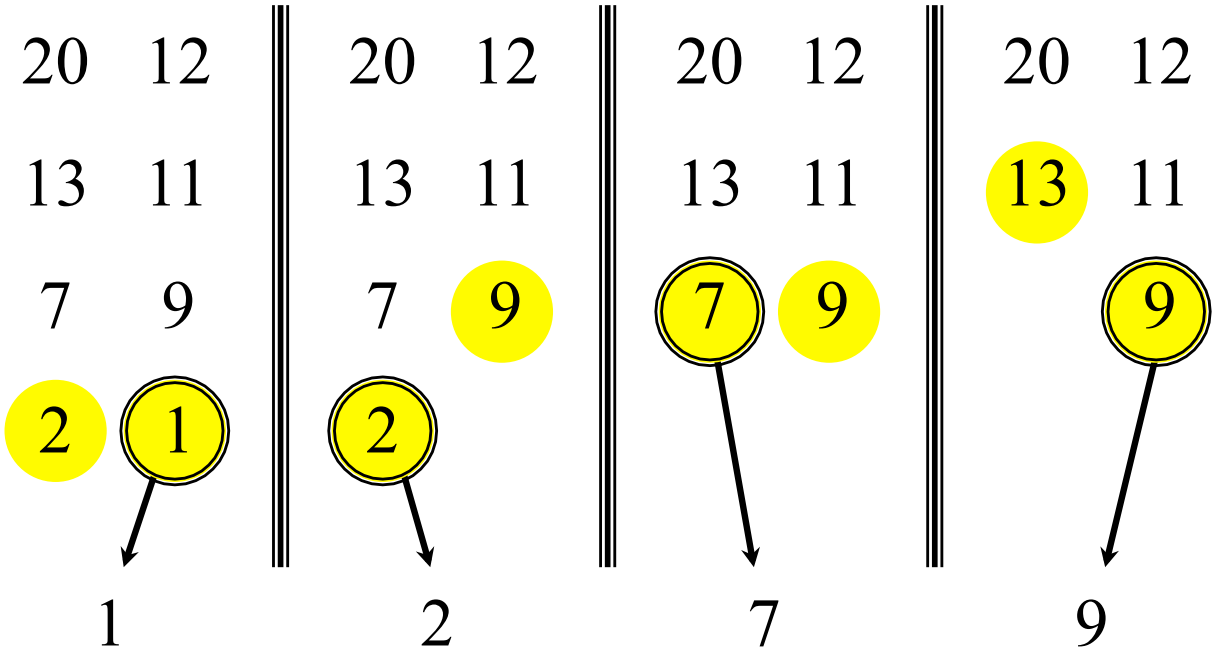


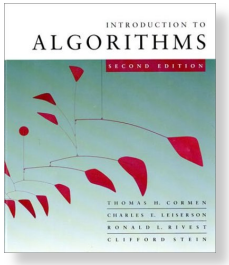
Merging two sorted arrays



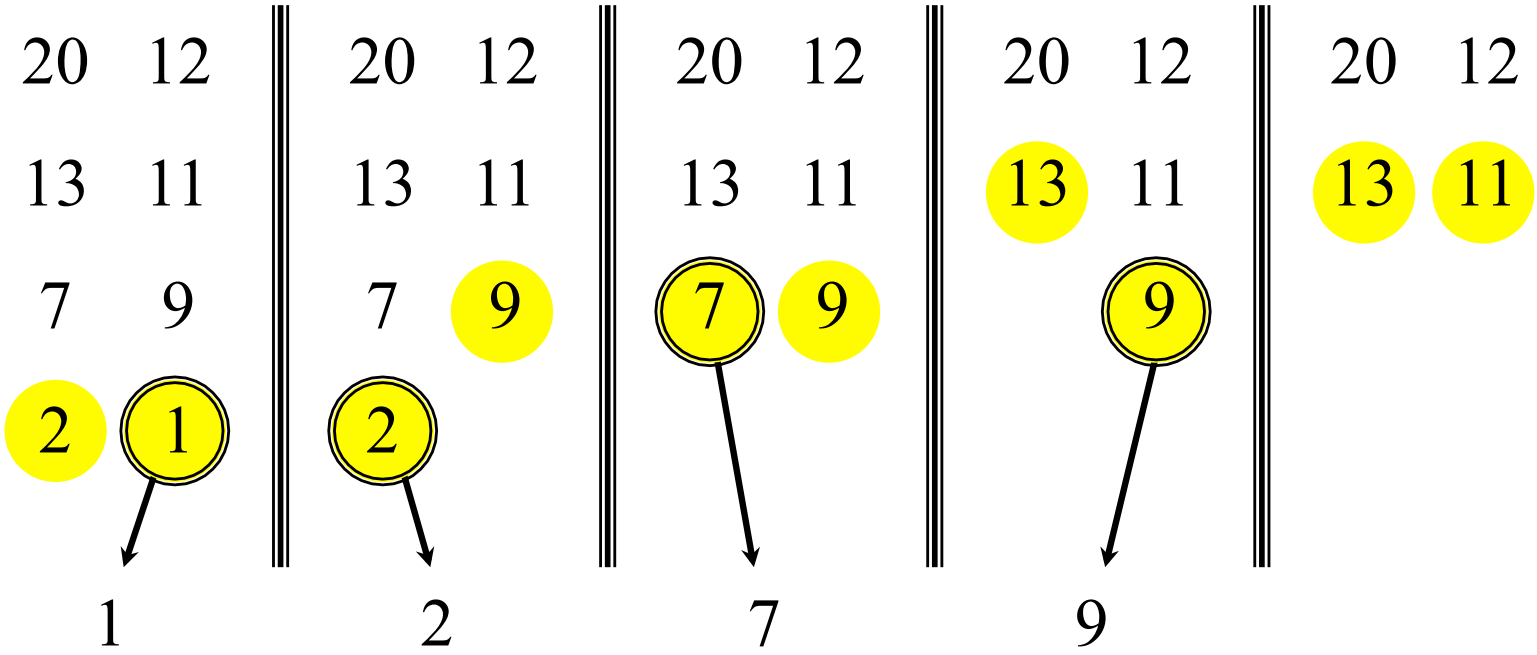


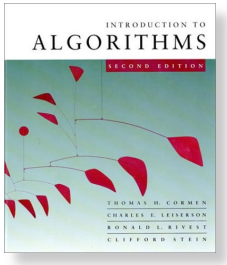
Merging two sorted arrays



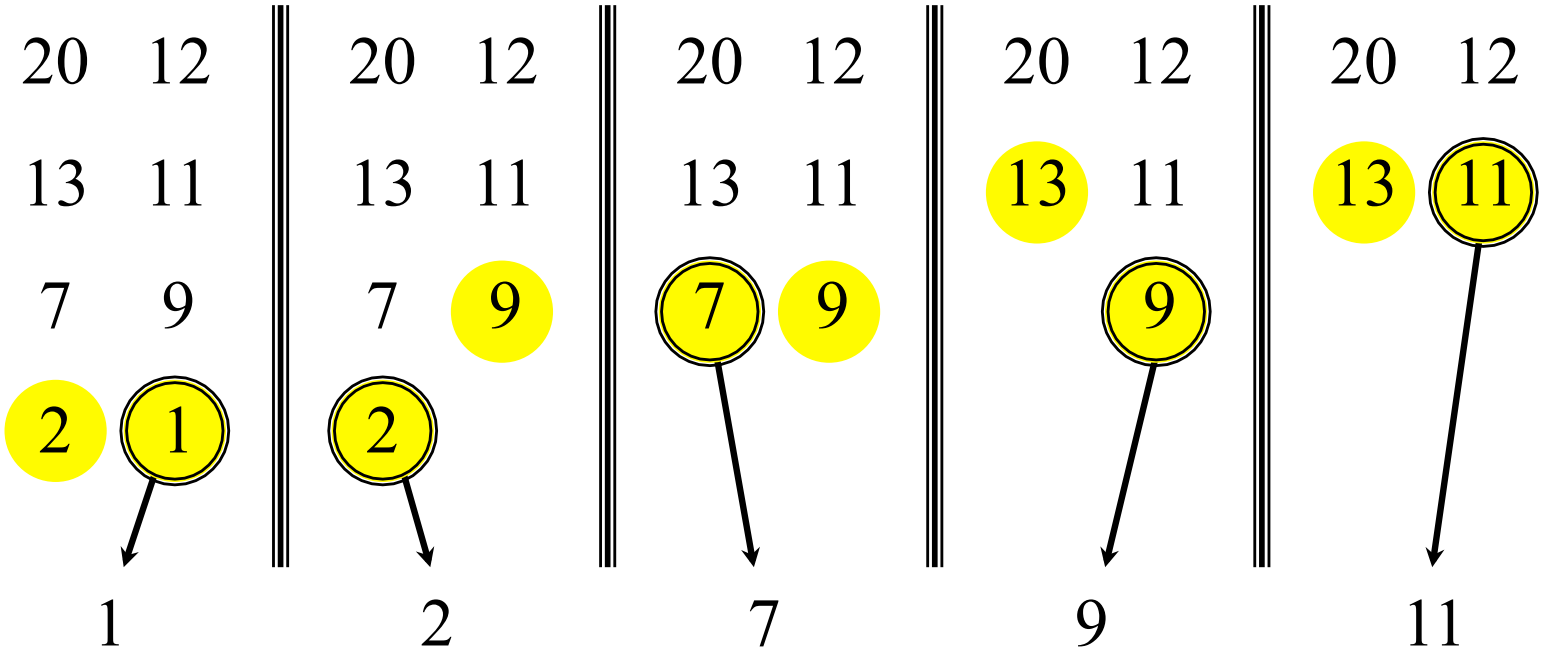


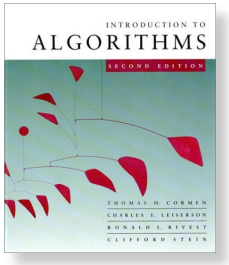
Merging two sorted arrays



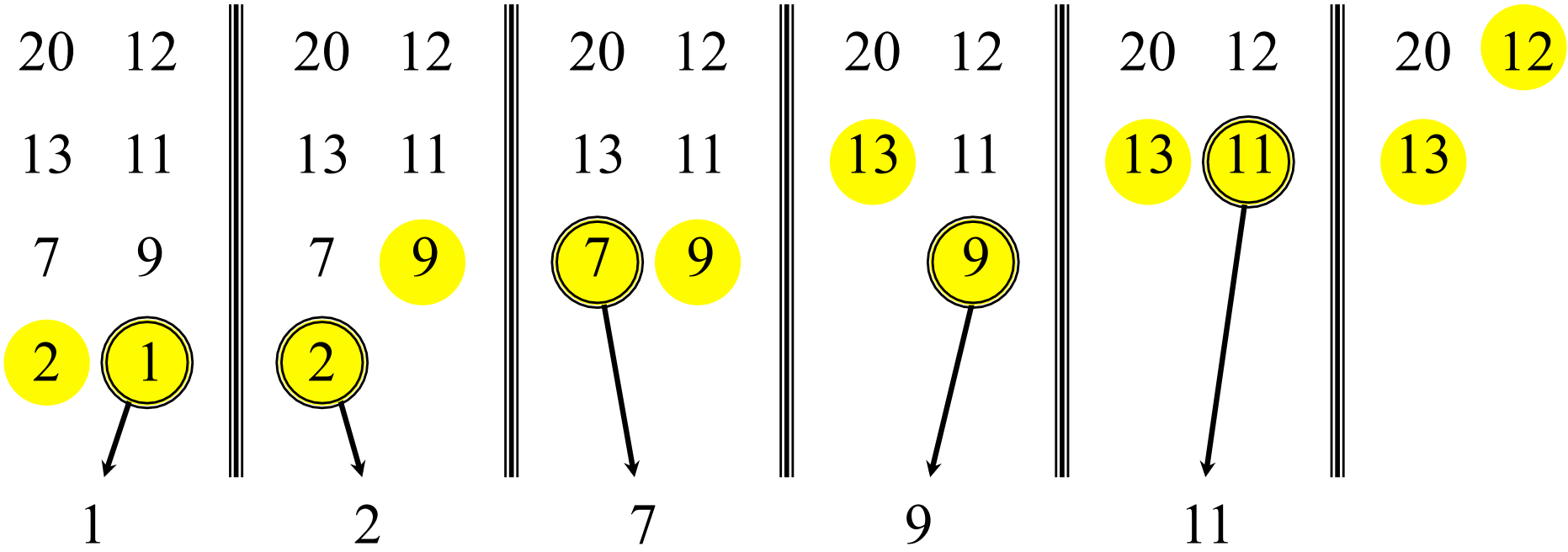


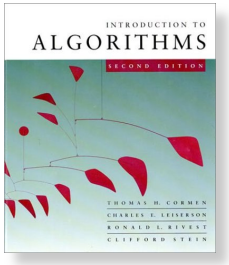
Merging two sorted arrays



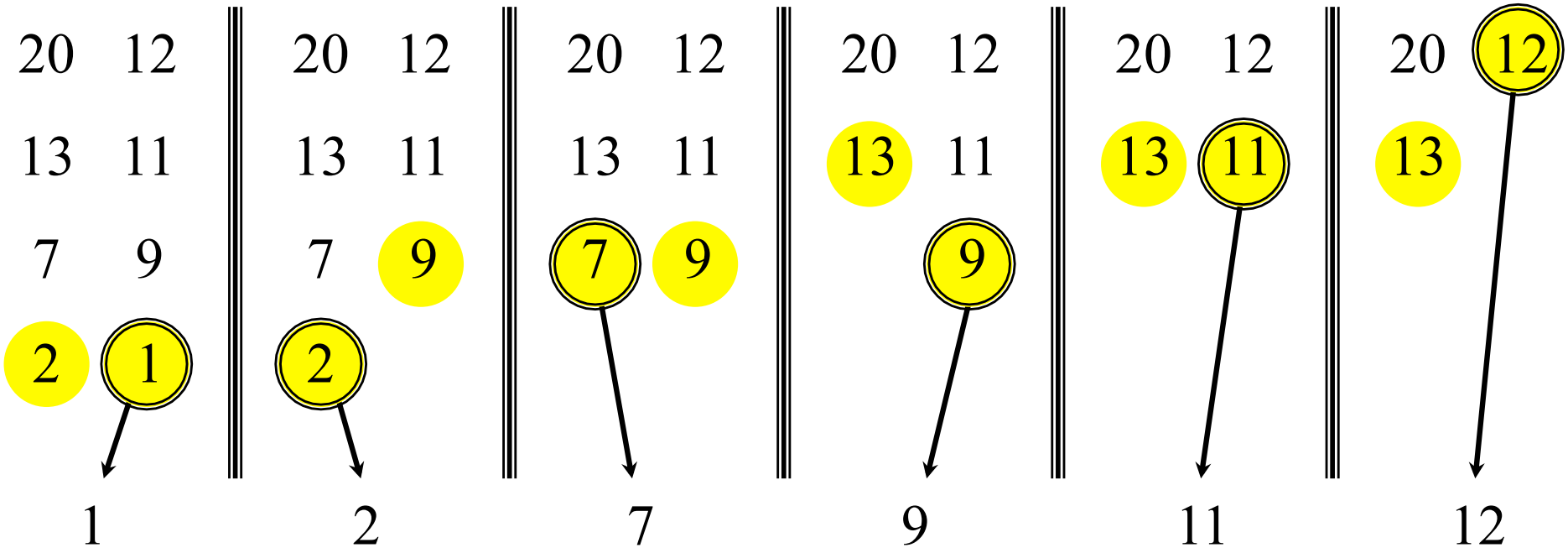


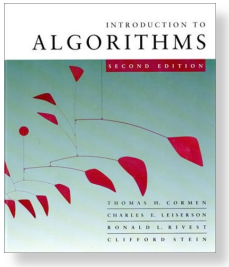
Merging two sorted arrays



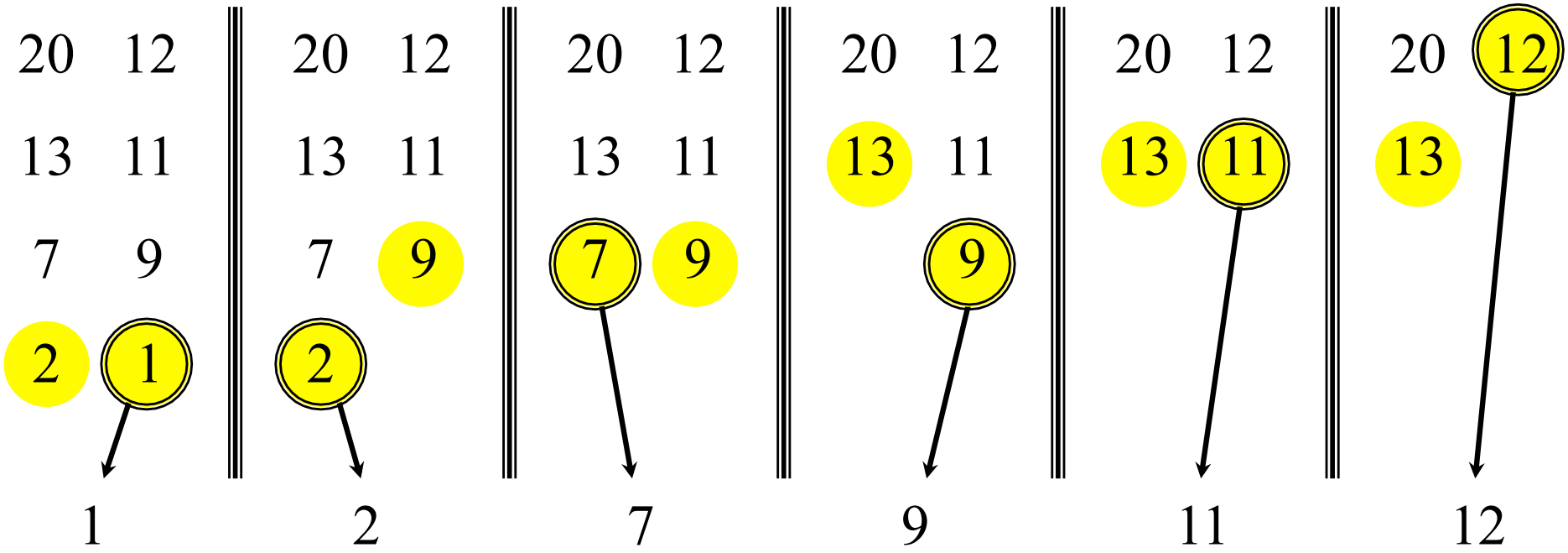


Merging two sorted arrays

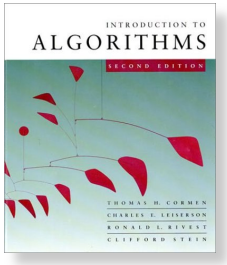




Merging two sorted arrays



Time = $\Theta(n)$ to merge a total of n elements (linear time).



Analyzing merge sort

$T(n)$

$\Theta(1)$

$2T(n/2)$

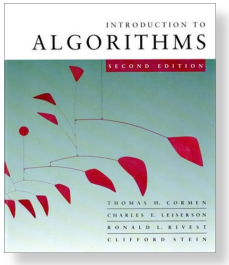
Abuse

$\Theta(n)$

MERGE-SORT $A[1 \ldots n]$

1. If $n = 1$, done.
2. Recursively sort $A[\,1 \ldots \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \ldots n\,]$.
3. ***"Merge"*** the 2 sorted lists

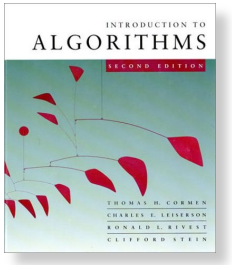
Sloppiness: Should be $T(\,\lceil n/2 \rceil\,) + T(\,\lfloor n/2 \rfloor\,)$, but it turns out not to matter asymptotically.



Recurrence for merge sort

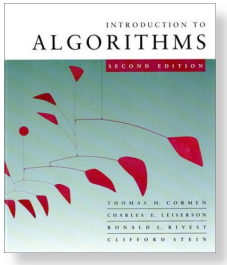
$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n, but only when it has no effect on the asymptotic solution to the recurrence.



Solving recurrences

- The analysis of merge sort required us to solve a recurrence.
- Recurrences are like solving integrals, differential equations, etc.
  - Learn a few tricks.
- Applications of recurrences to divide-and-conquer algorithms.



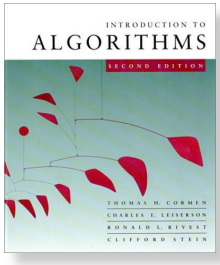
Substitution method

Recursion Tree method
basis of the Master Theorem

The most general method:

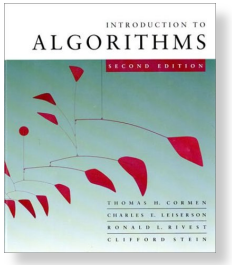
1. ***Guess*** the form of the solution.
2. ***Verify*** by induction.
3. ***Solve*** for constants.

Not covered in CS3230; those interested see example and details in CLRS



Recursion tree

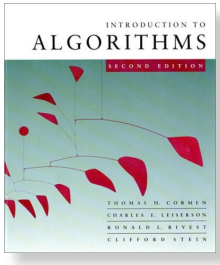
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recursion tree

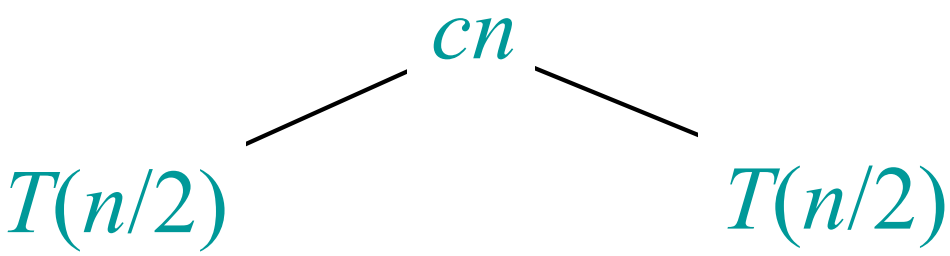
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

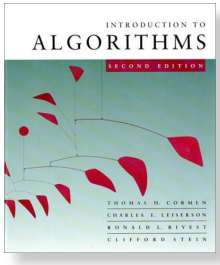
$T(n)$



Recursion tree

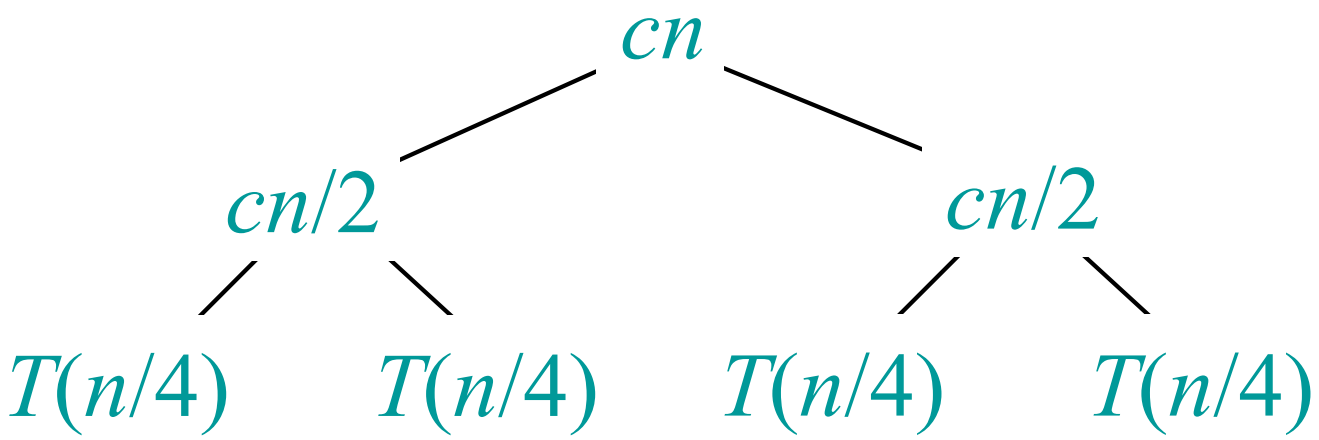
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

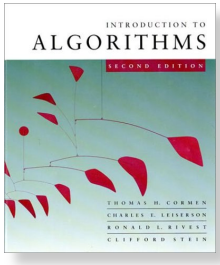




Recursion tree

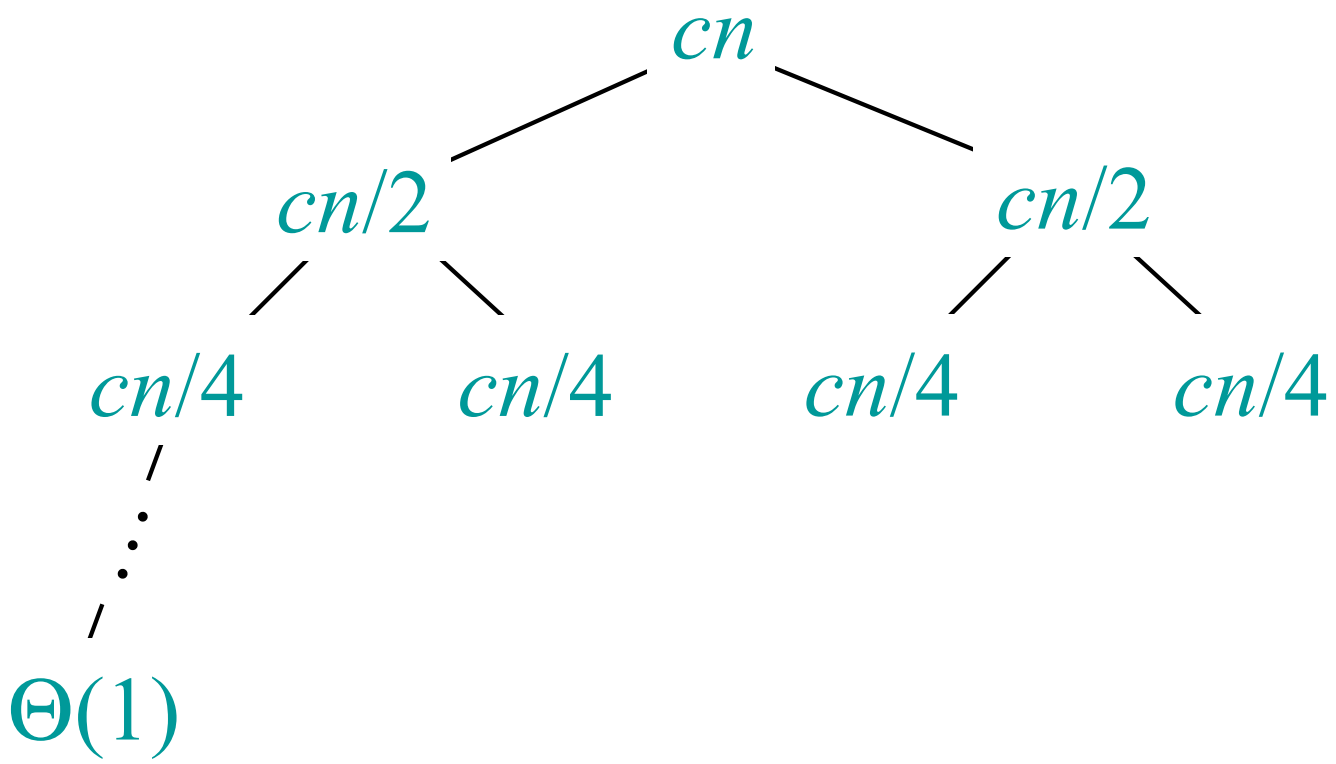
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

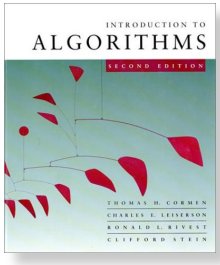




Recursion tree

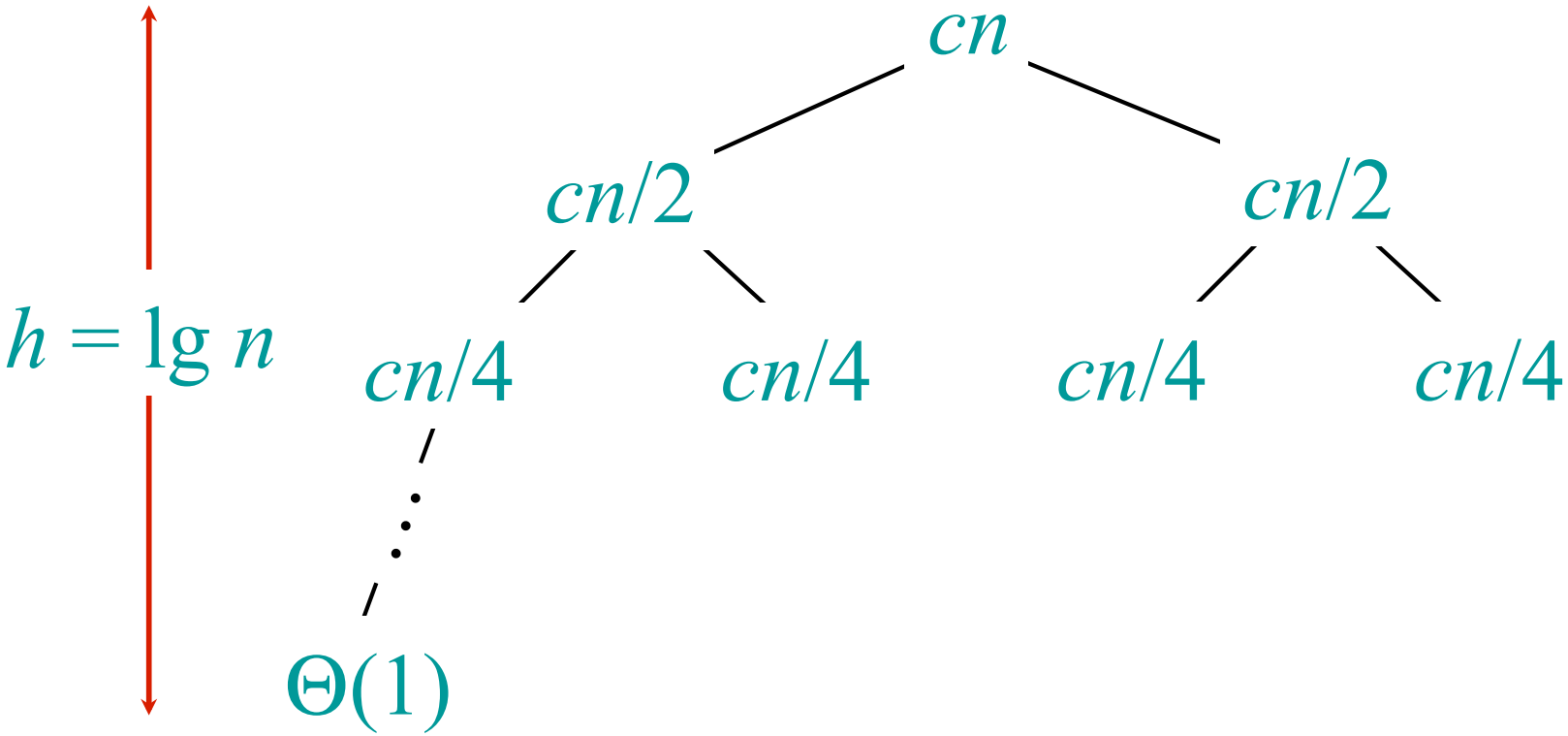
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

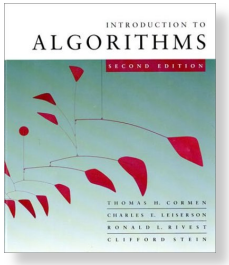




Recursion tree

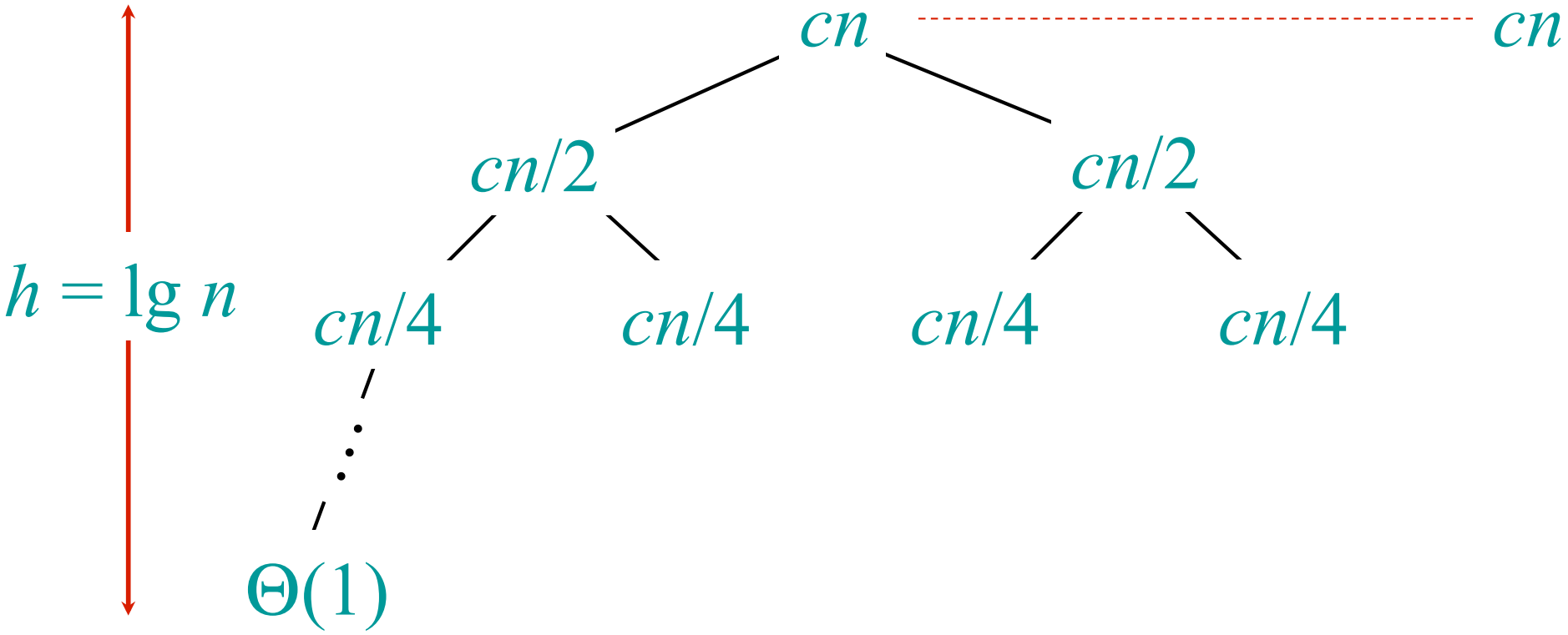
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

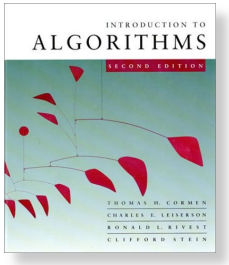




Recursion tree

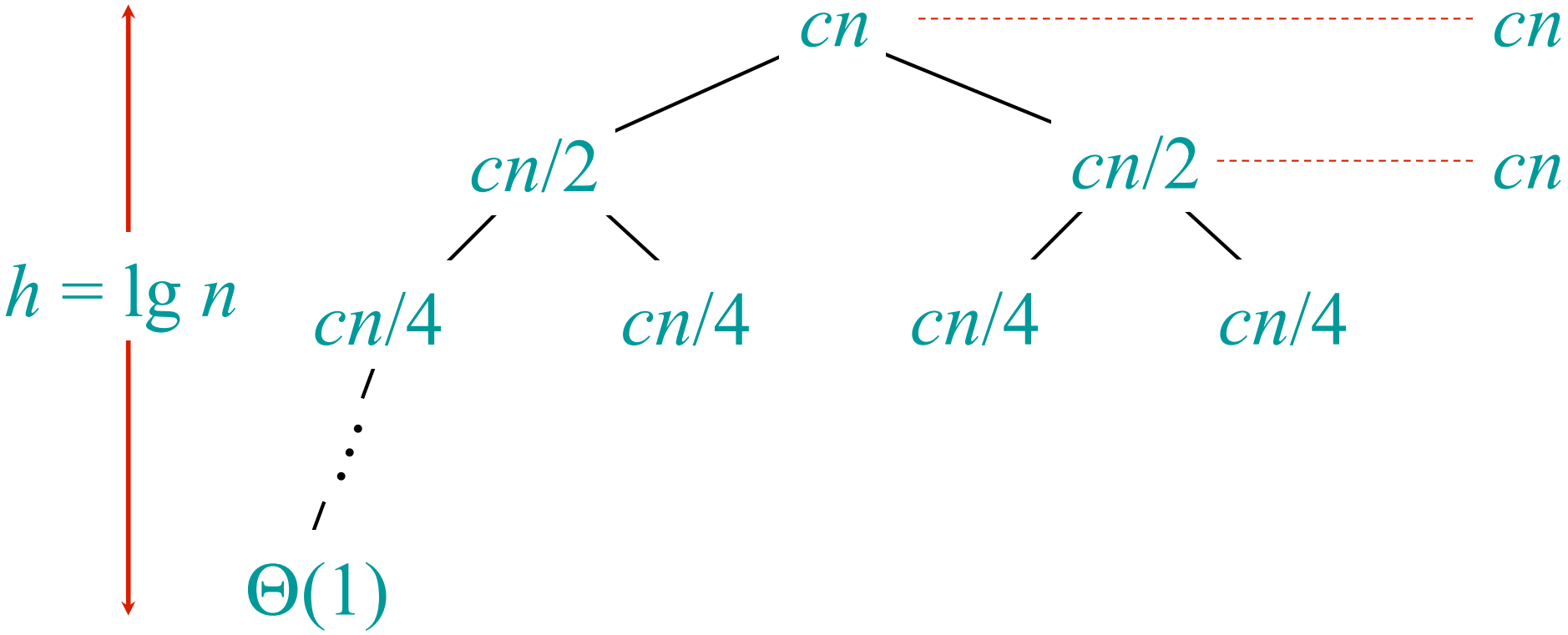
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

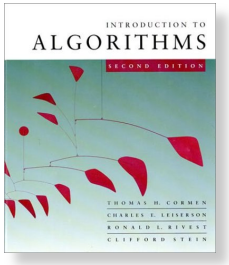




Recursion tree

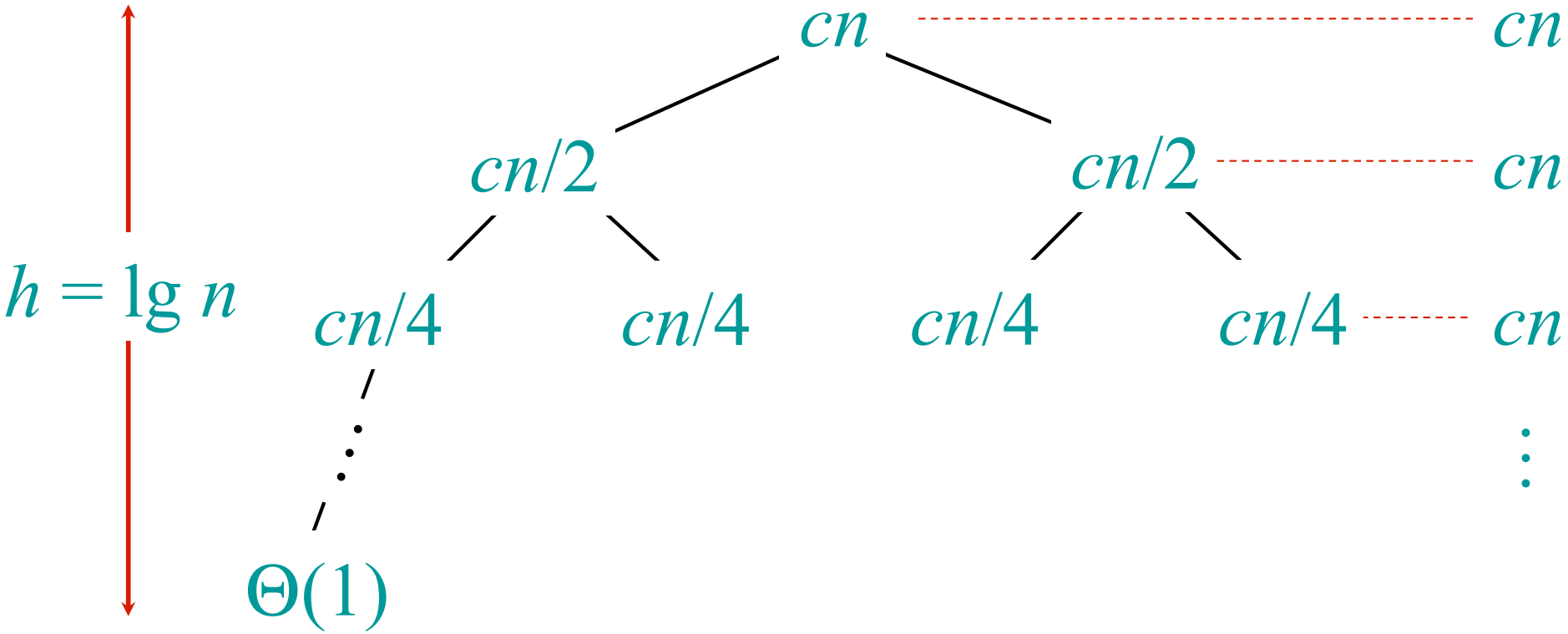
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

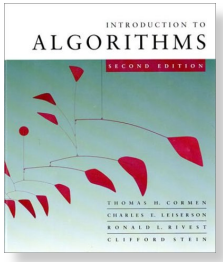




Recursion tree

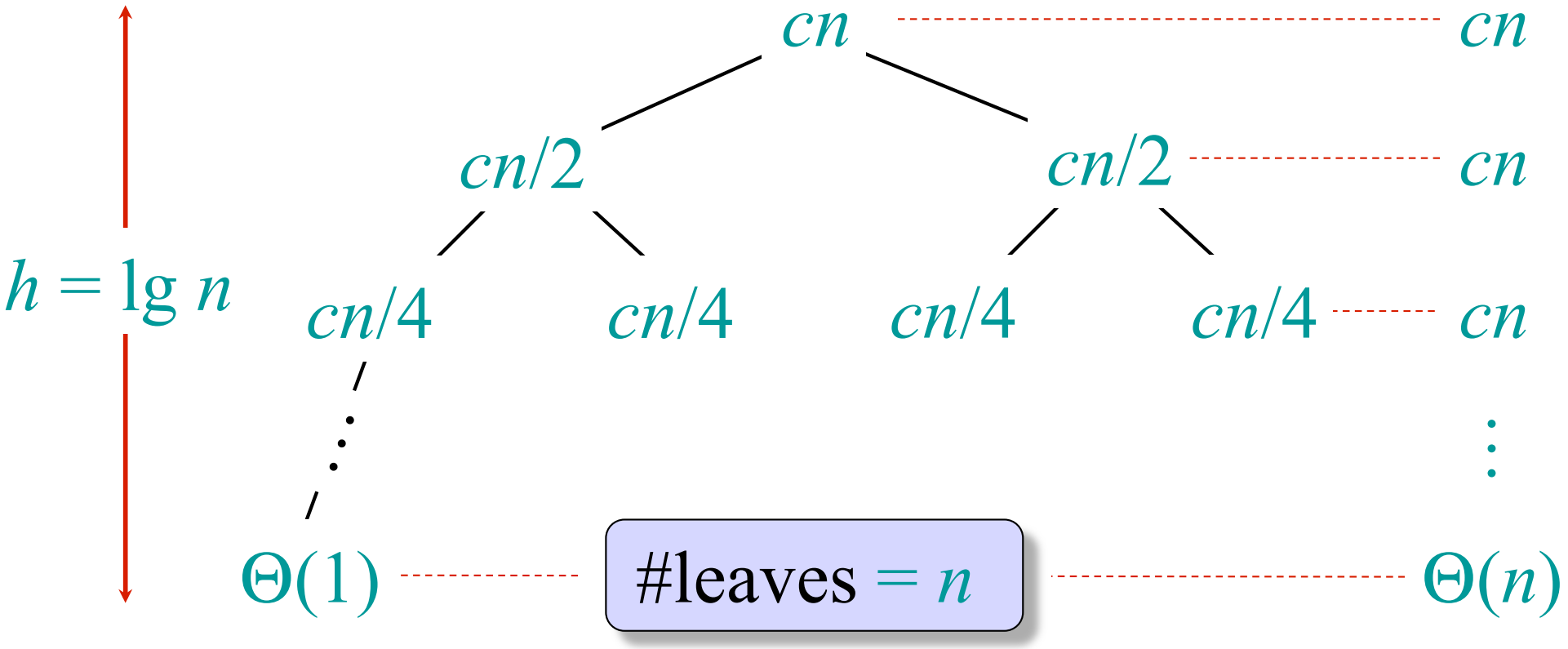
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

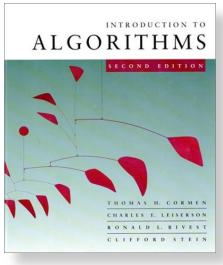




Recursion tree

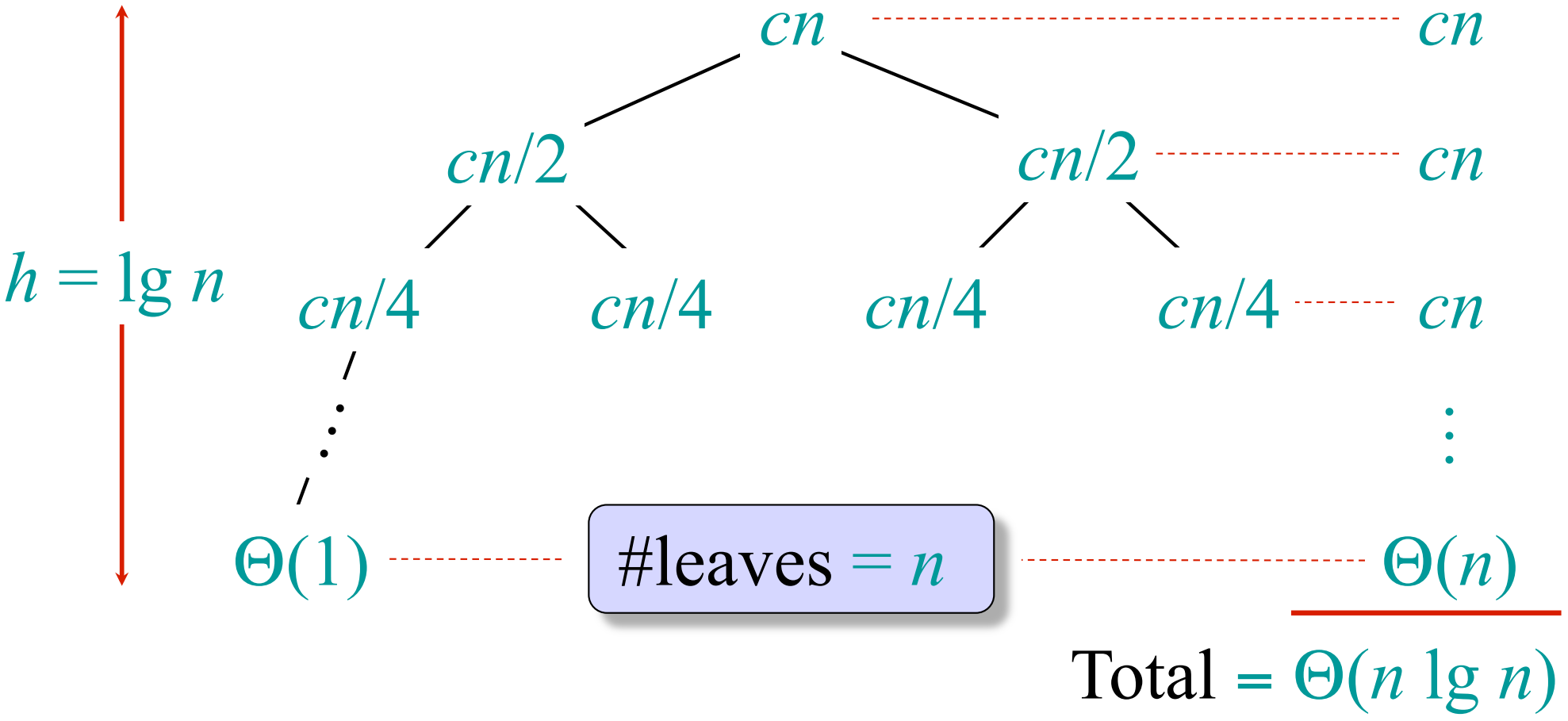
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

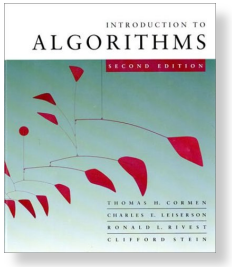




Recursion tree

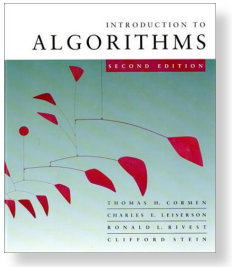
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.





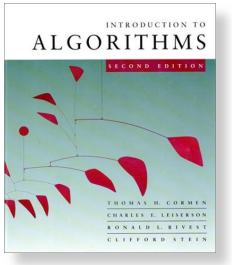
Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (…).
- The recursion-tree method promotes intuition, however.
- The recursion tree method is good for generating guesses for the substitution method.



Example of recursion tree

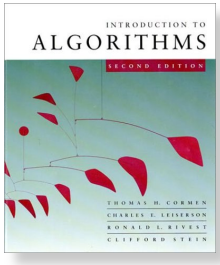
Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Example of recursion tree

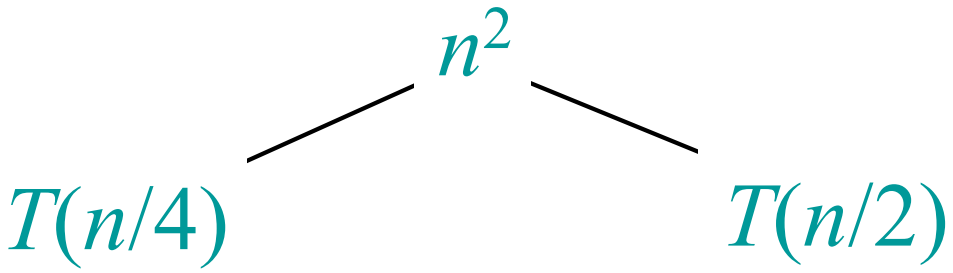
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

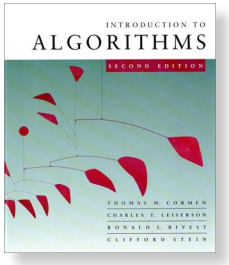
$T(n)$



Example of recursion tree

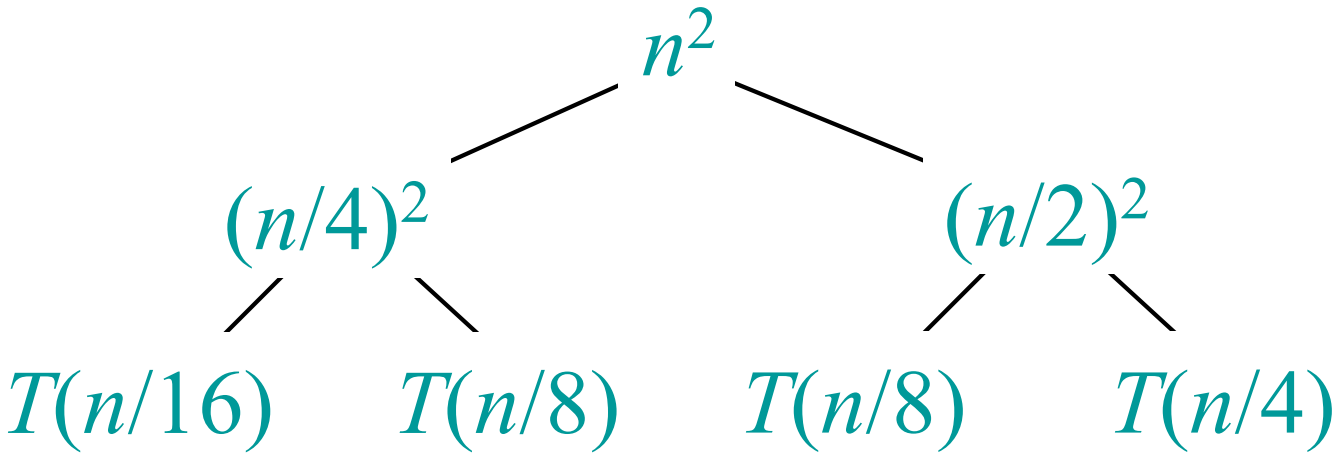
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

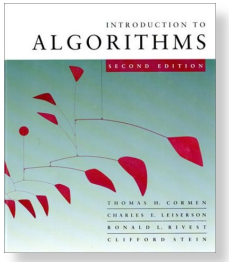




Example of recursion tree

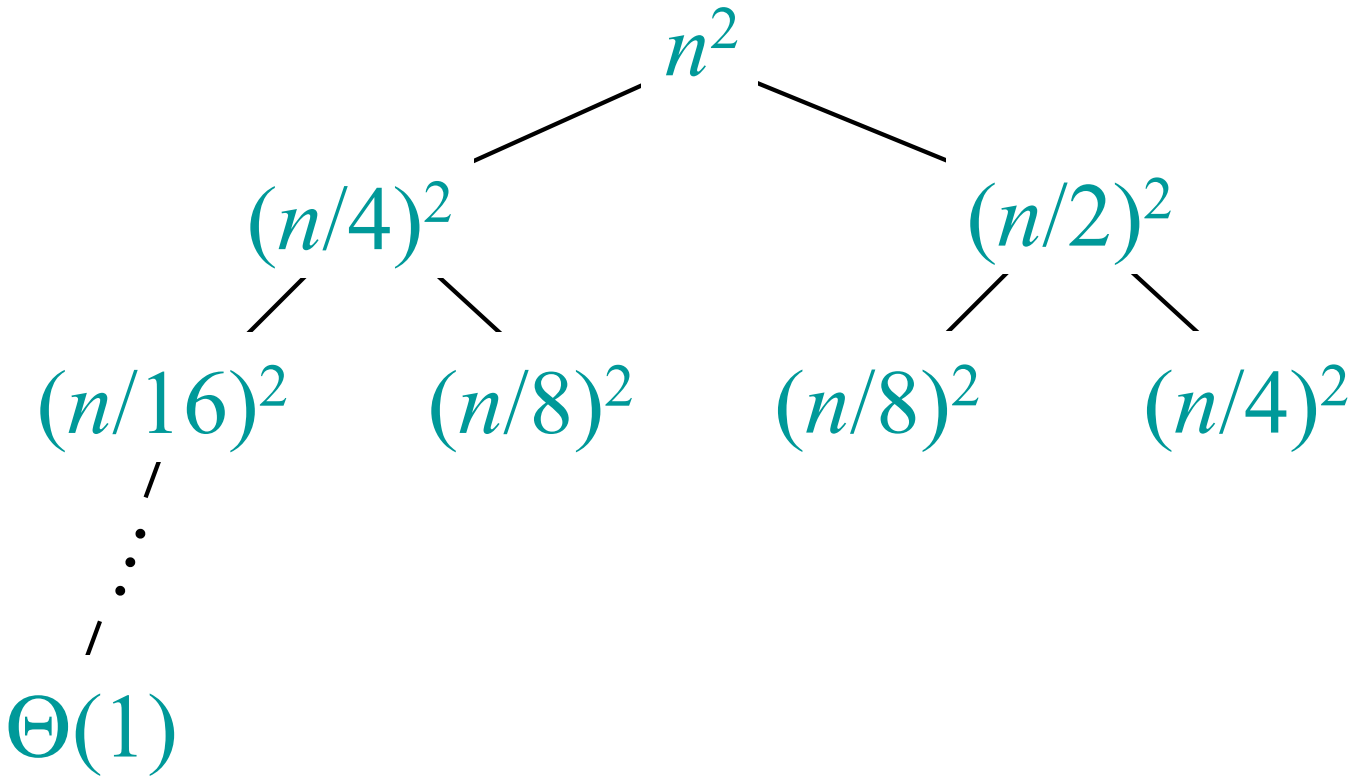
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

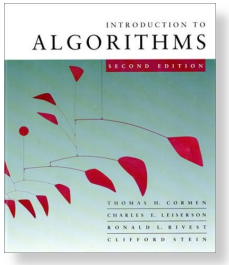




Example of recursion tree

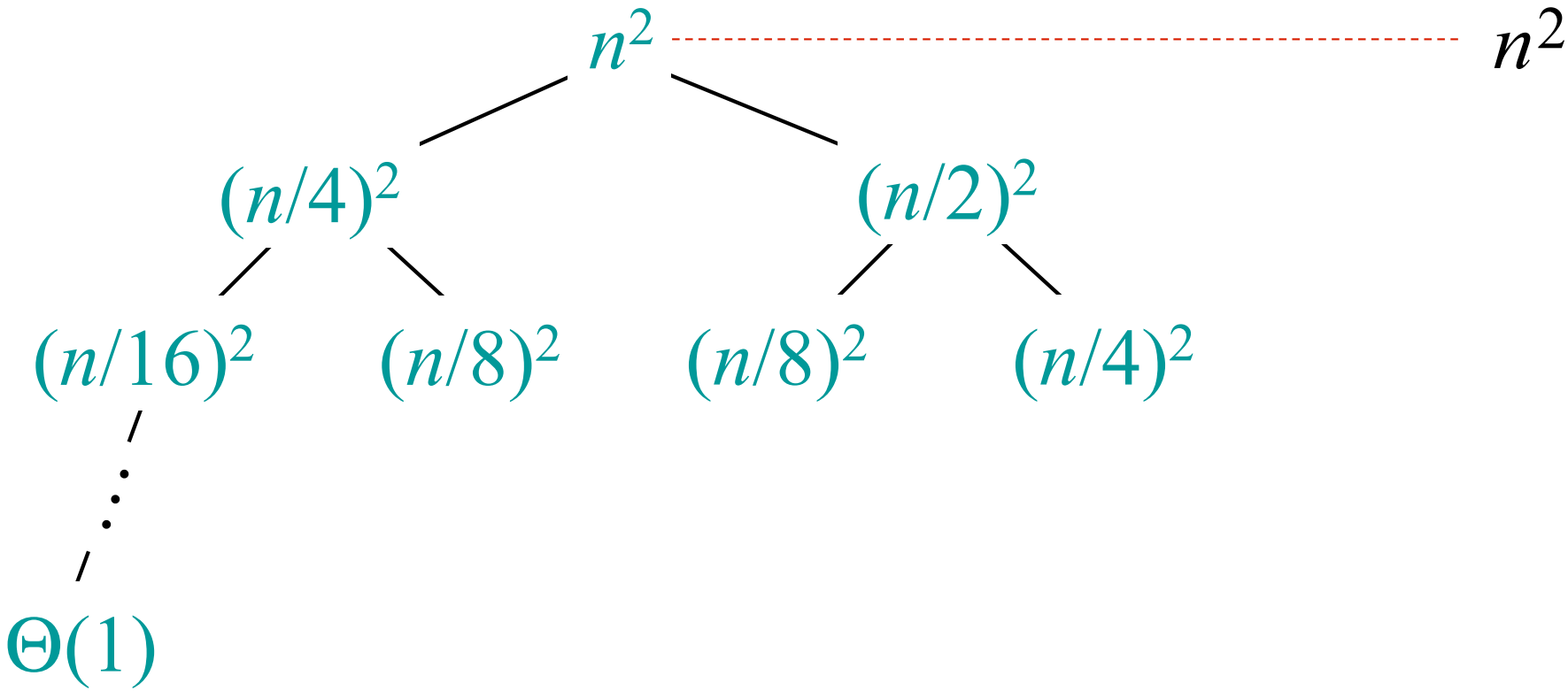
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

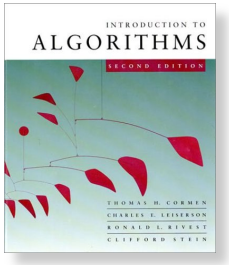




Example of recursion tree

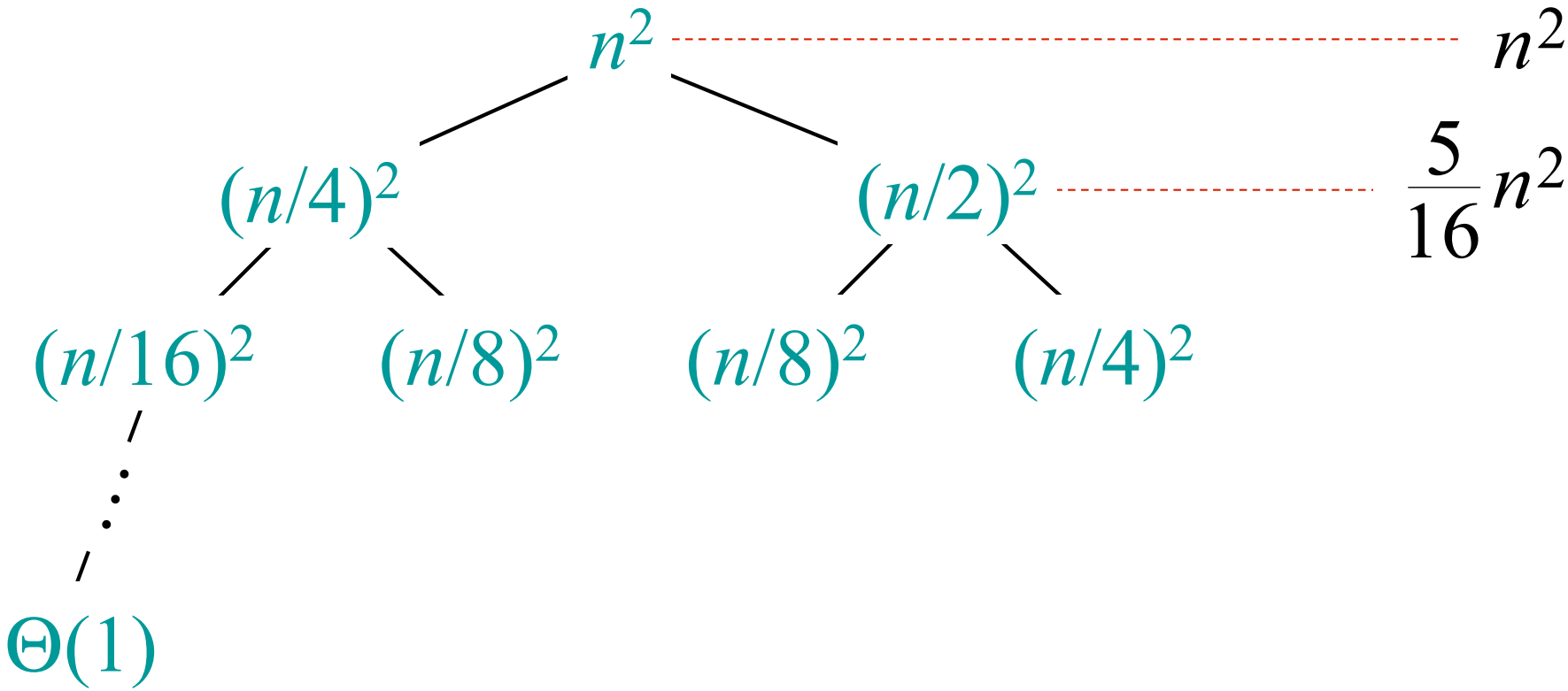
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

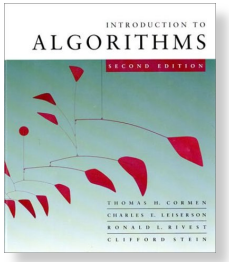




Example of recursion tree

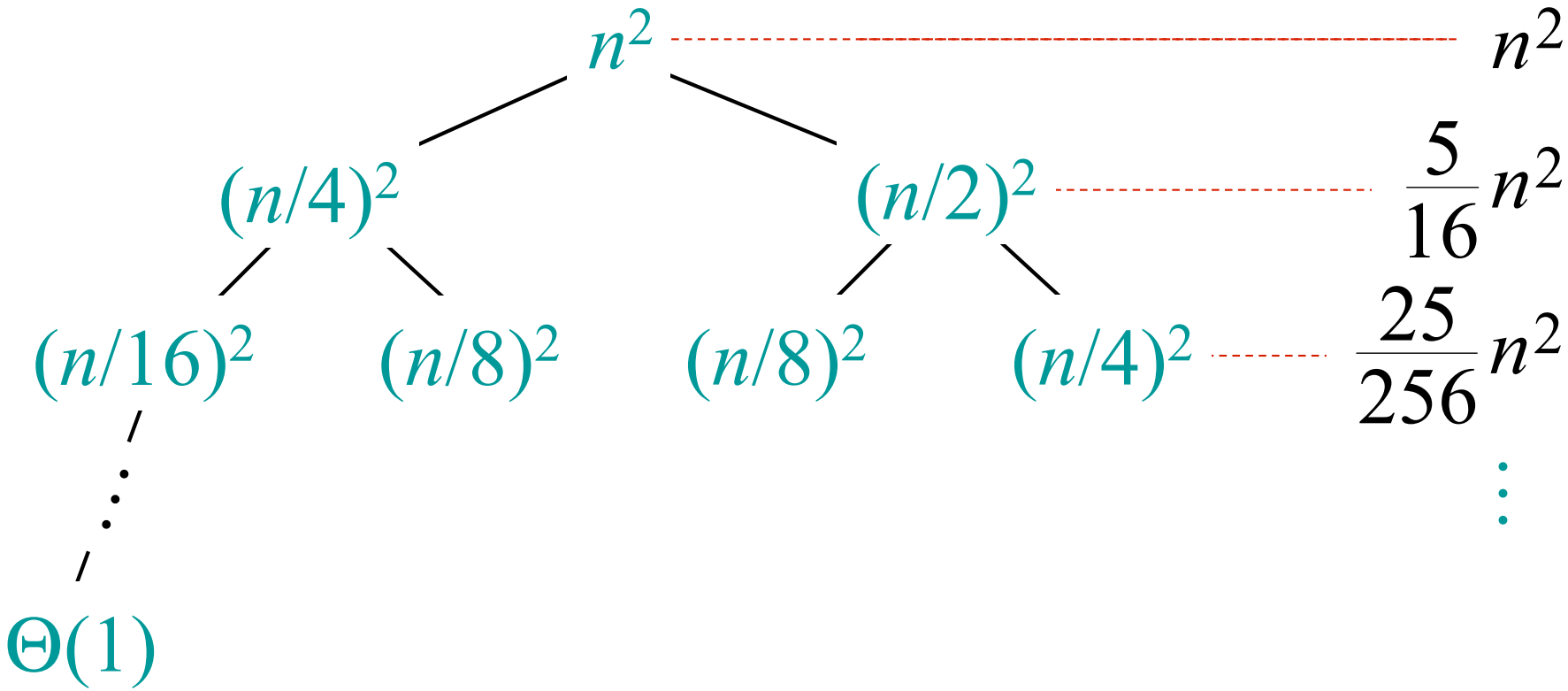
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

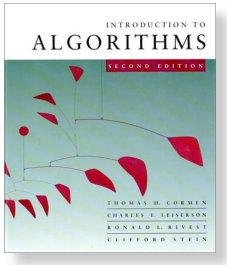




Example of recursion tree

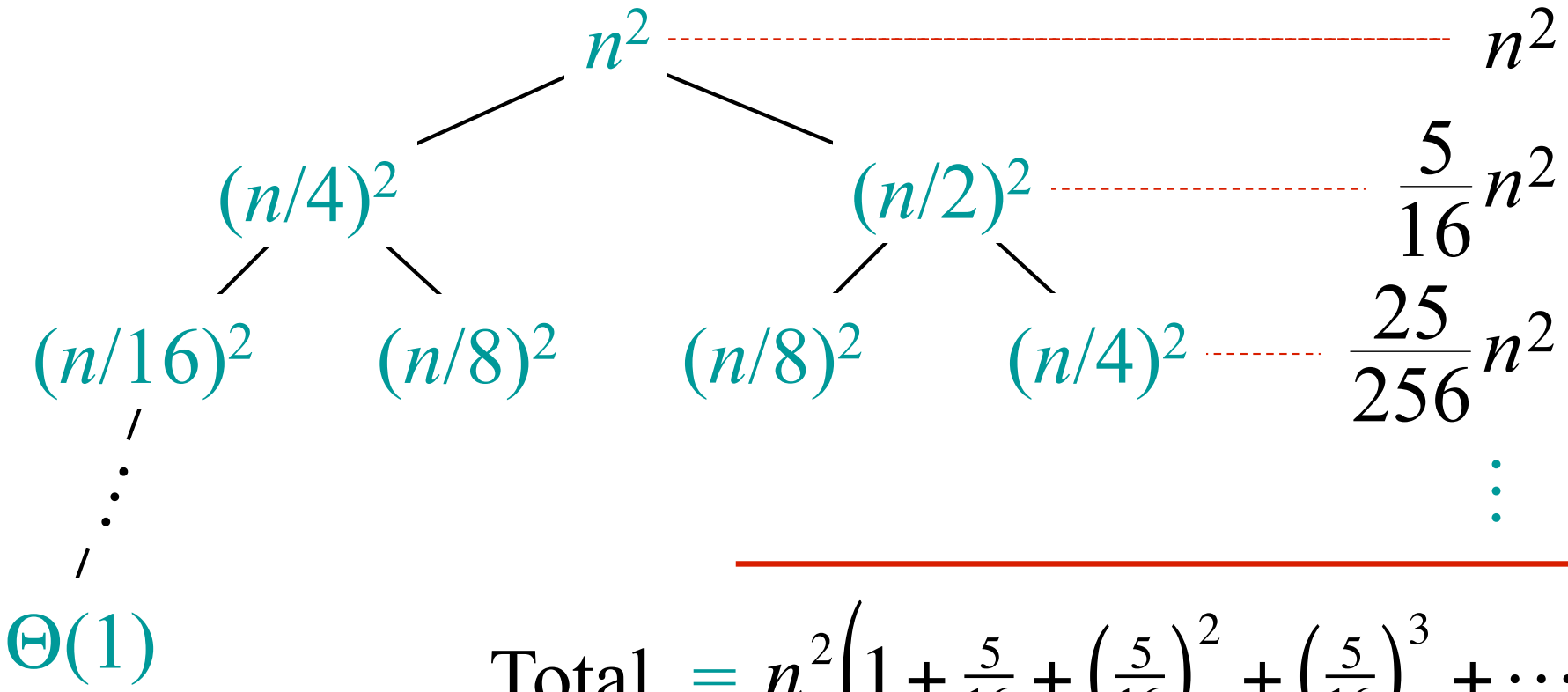
Solve $T(n) = T(n/4) + T(n/2) + n^2$:





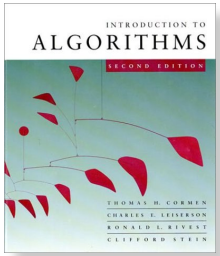
Example of recursion tree

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

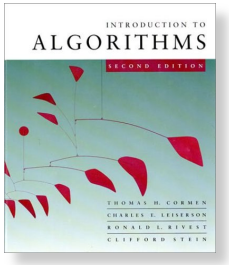


$$\text{Total} = n^2\left(1 + \tfrac{5}{16} + \left(\tfrac{5}{16}\right)^2 + \left(\tfrac{5}{16}\right)^3 + \cdots\right)$$

$$= \Theta(n^2) \qquad \textit{geometric series}$$



The Master Theorem

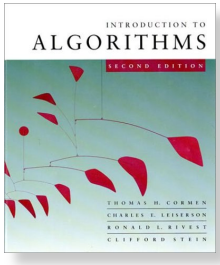


The master method

The master method applies to recurrences of the form

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

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

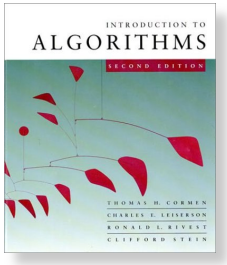


Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

   Solution: $T(n) = \Theta(n^{\log_b a})$.



Three common cases

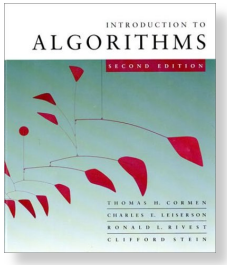
Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

   Solution: $T(n) = \Theta(n^{\log_b a})$.

2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

   Solution: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.



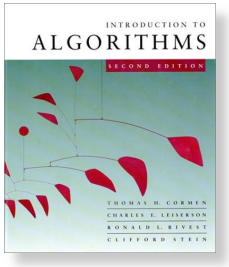
Three common cases (cont.)

Compare $f(n)$ with $n^{\log_b a}$:

3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor),

   and $f(n)$ satisfies the ***regularity condition*** that $a f(n/b) \le c f(n)$ for some constant $c < 1$.

   Solution: $T(n) = \Theta(f(n))$.



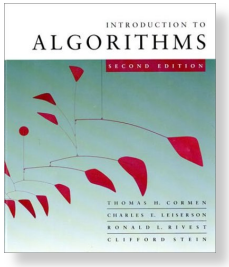
Examples

Ex. $T(n) = 4T(n/2) + n$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n.$

CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.

$\therefore\ T(n) = \Theta(n^2).$



Examples

Ex. $T(n) = 4T(n/2) + n$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n.$

Case 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.

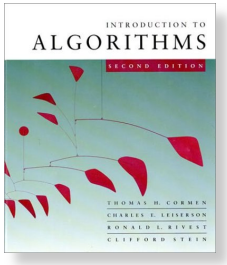
$\therefore\ T(n) = \Theta(n^2).$

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

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2.$

Case 2: $f(n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.

$\therefore\ T(n) = \Theta(n^2 \lg n).$



Examples

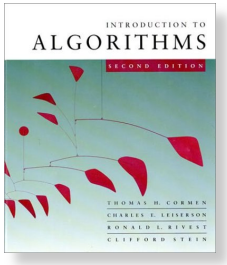
Ex. $T(n) = 4T(n/2) + n^3$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^3.$

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

and $4(n/2)^3 \le cn^3$ (reg. cond.) for $c = 1/2$.

$\therefore\ T(n) = \Theta(n^3)$.



Examples

Ex. $T(n) = 4T(n/2) + n^3$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^3.$

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

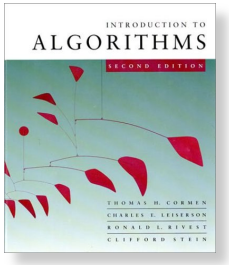
and $4(n/2)^3 \le cn^3$ (reg. cond.) for $c = 1/2$.

$\therefore\ T(n) = \Theta(n^3)$.

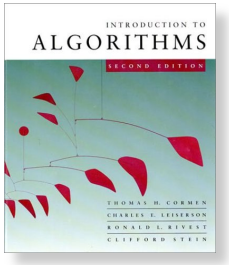
Ex. $T(n) = 4T(n/2) + n^2/\lg n$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2/\lg n.$

Master method does not apply. In particular, for every constant $\varepsilon > 0$, we have $n^\varepsilon = \omega(\lg n)$.

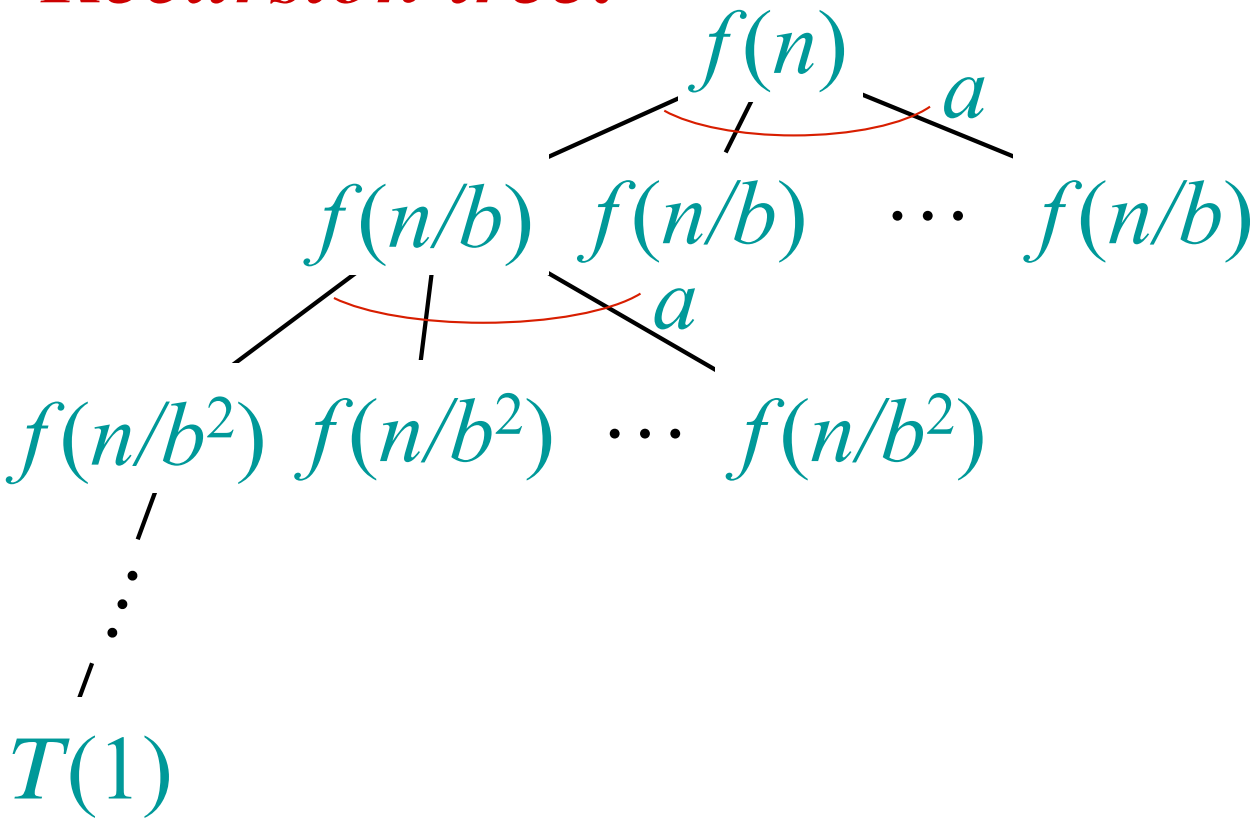


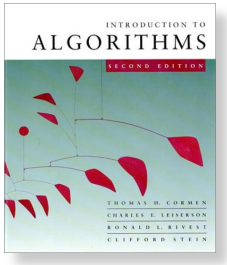
Idea behind the proof of the Master Theorem



Idea of master theorem

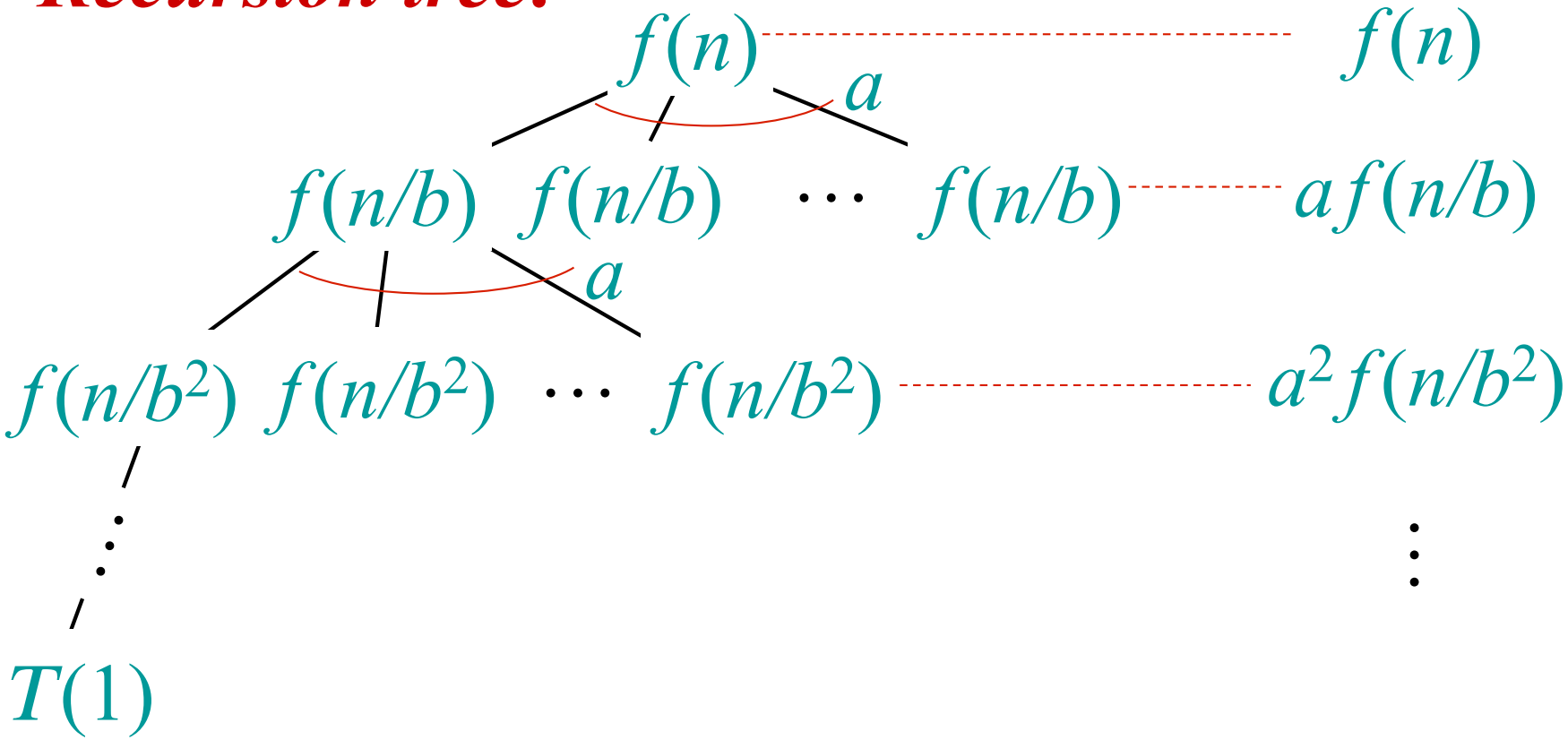
Recursion tree:

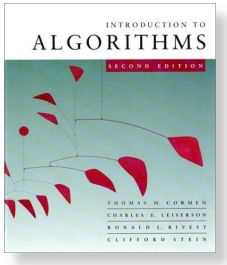




Idea of master theorem

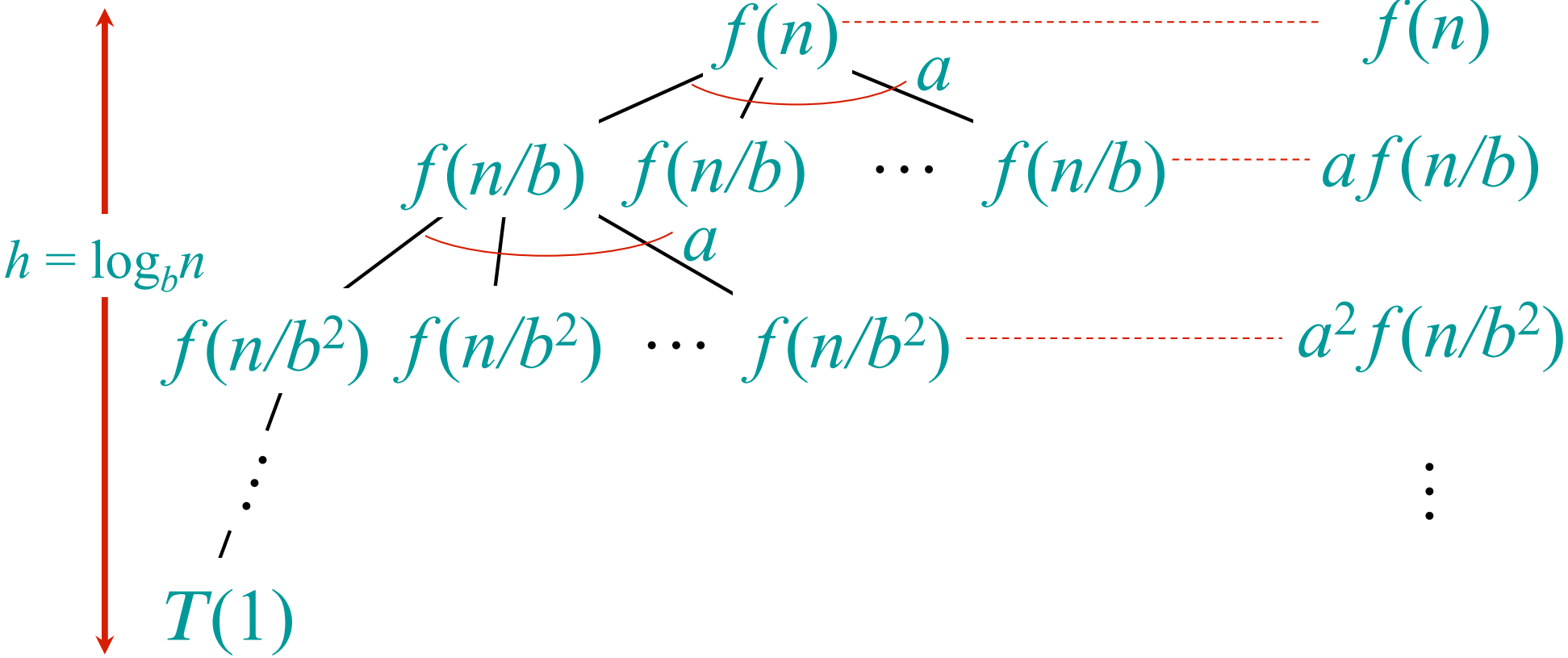
Recursion tree:

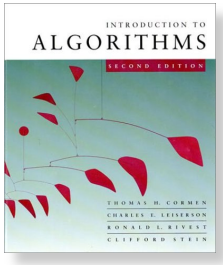




Idea of master theorem

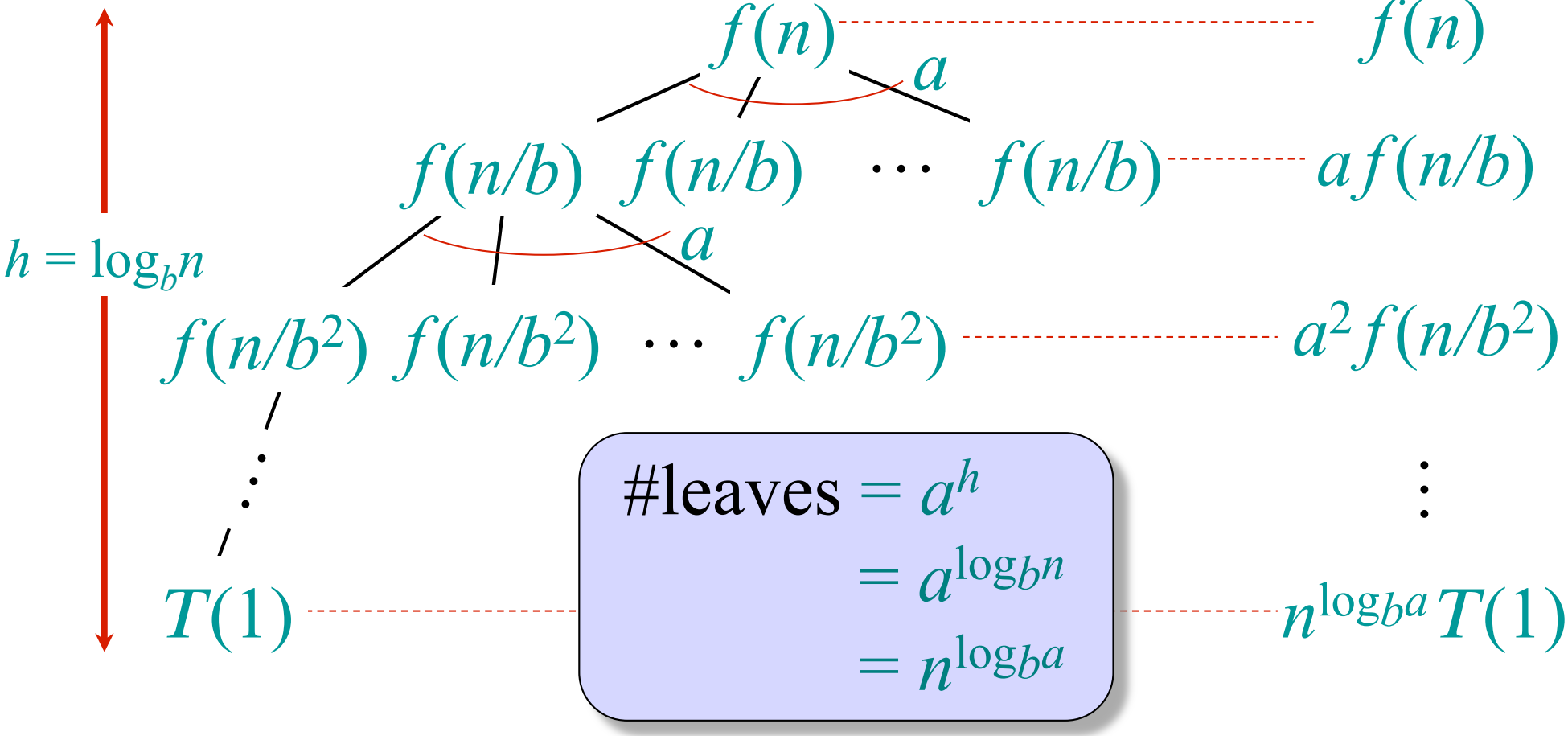
Recursion tree:

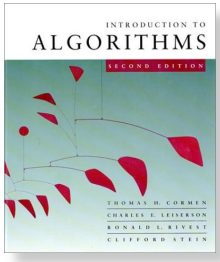




Idea of master theorem

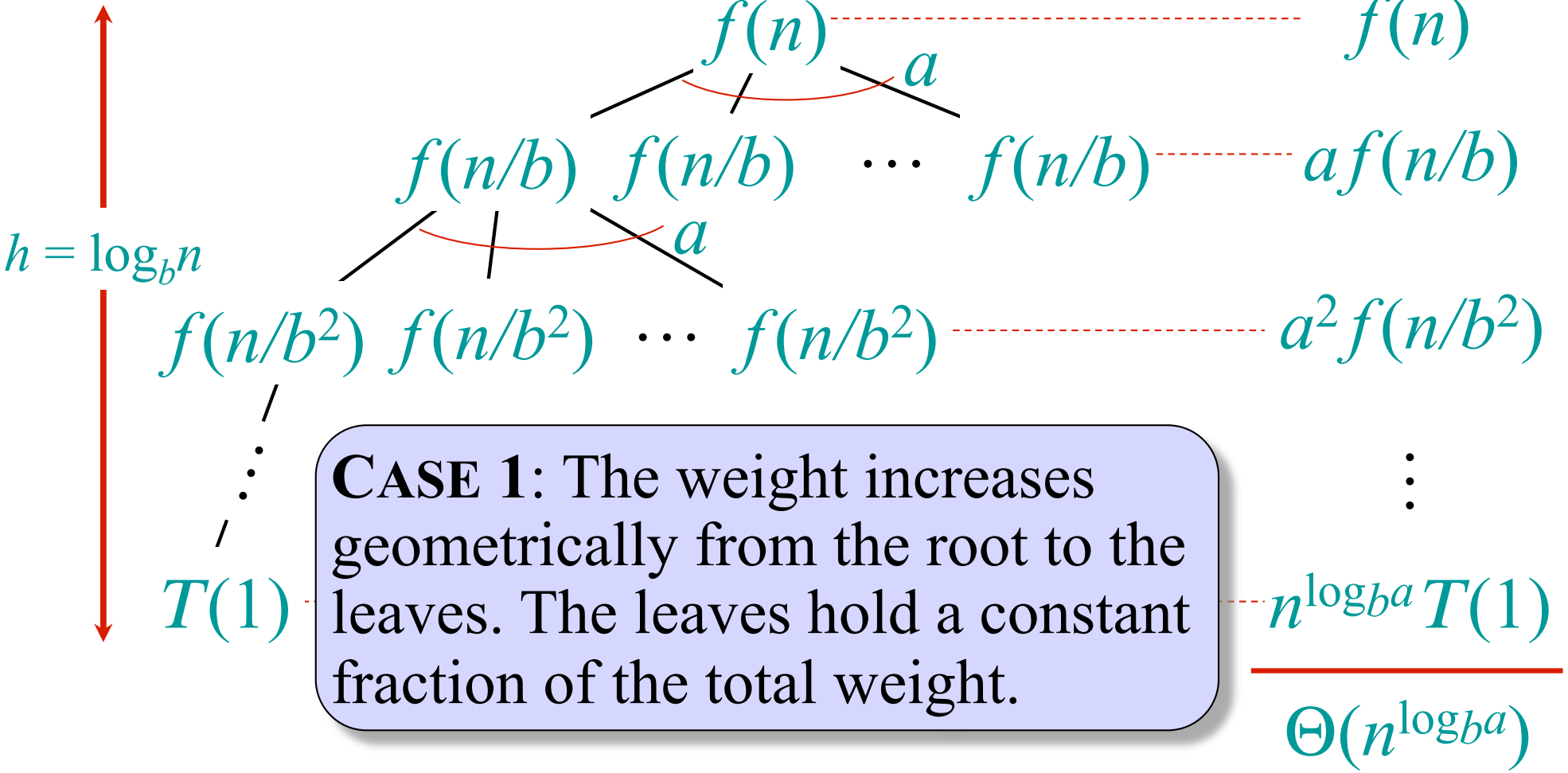
Recursion tree:



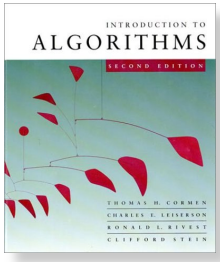


Idea of master theorem

Recursion tree:

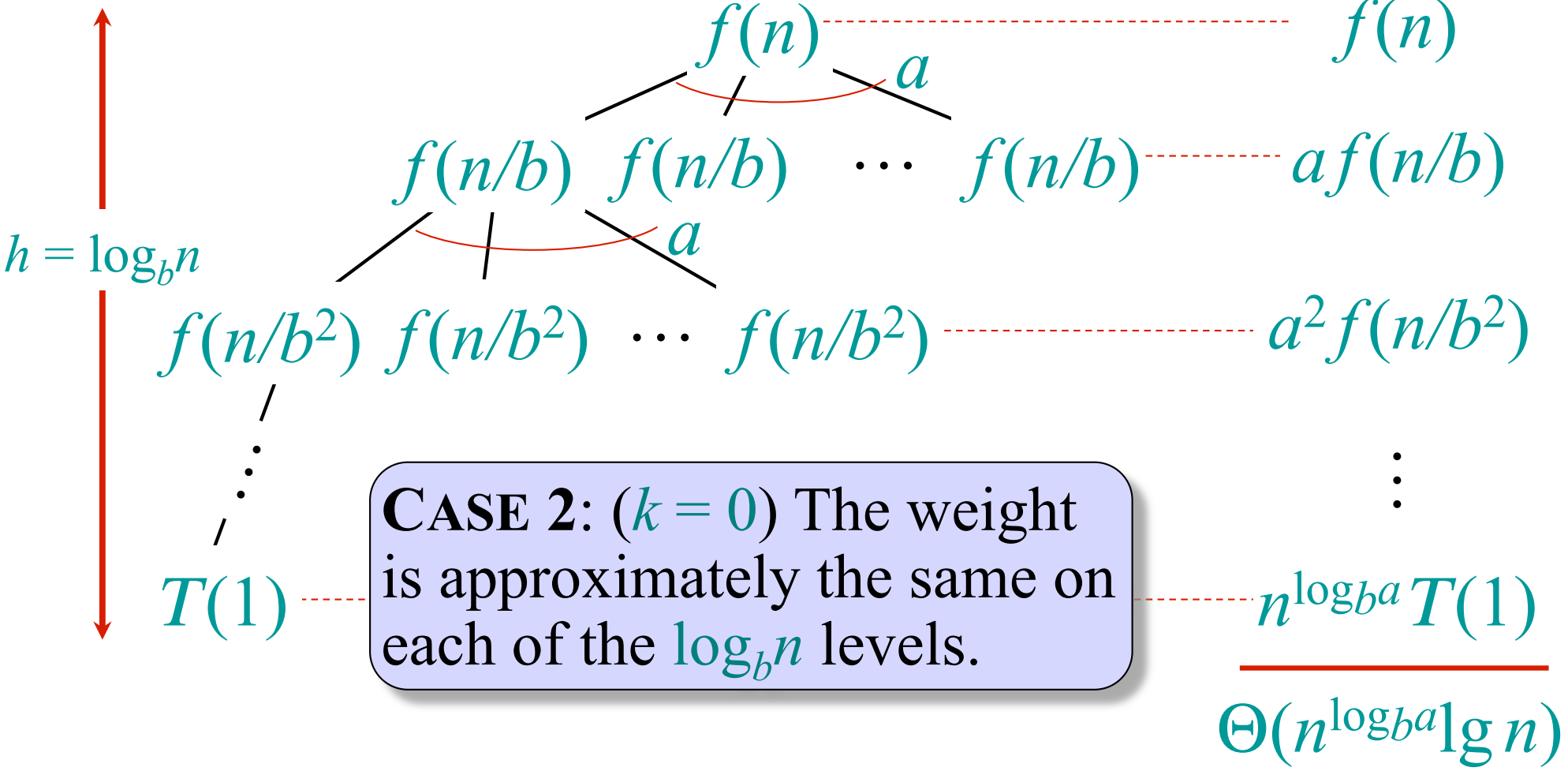


CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

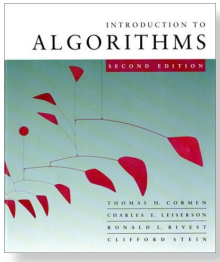


Idea of master theorem

Recursion tree:

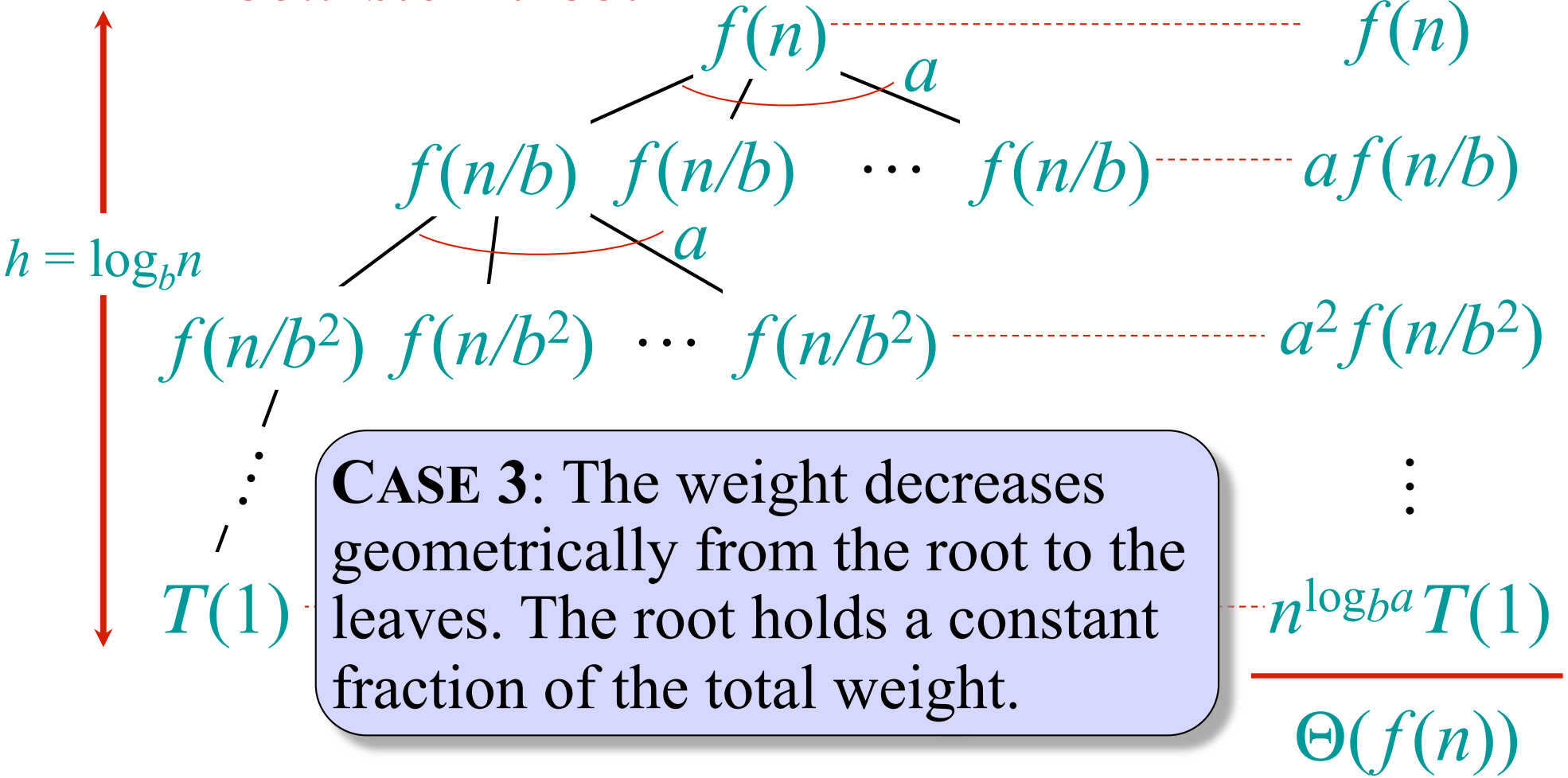


CASE 2: ($k = 0$) The weight is approximately the same on each of the $\log_b n$ levels.

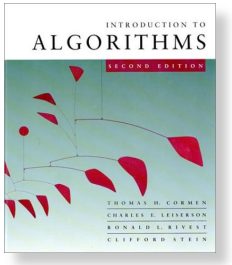


Idea of master theorem

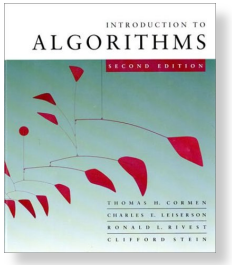
Recursion tree:



CASE 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

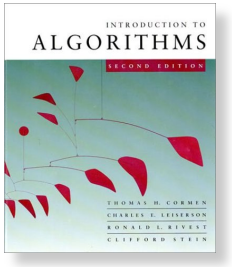


Revise many Divide and Conquer Algorithms



The divide-and-conquer design paradigm

1. ***Divide*** the problem (instance) into subproblems.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.

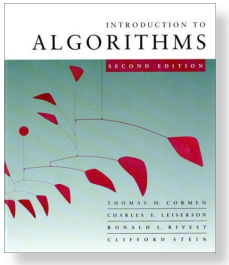


Merge sort

1. Divide: Trivial.

2. Conquer: Recursively sort 2 subarrays.

3. Combine: Linear-time merge.



Merge sort

1. Divide: Trivial.

2. Conquer: Recursively sort 2 subarrays.

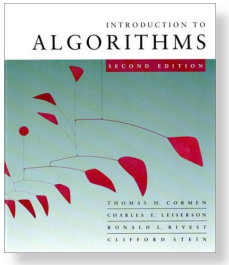
3. Combine: Linear-time merge.

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

 # subproblems

subproblem size

work dividing and combining



Master theorem (Mergesort)

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

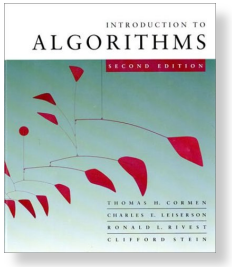
 # subproblems

subproblem size

work dividing and combining

Merge sort: $a = 2,\ b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$

$\Rightarrow$ CASE 2 $(k = 0) \Rightarrow T(n) = \Theta(n \lg n)$.



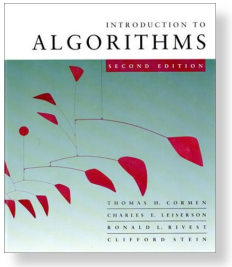
Binary search

Find an element in a sorted array:

1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.



Binary search

Find an element in a sorted array:

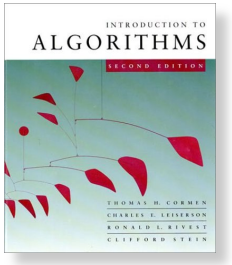
1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Example: Find 9

3	5	7	8	9	12	15



Binary search

Find an element in a sorted array:

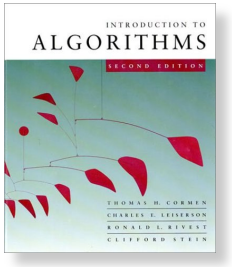
1.Divide: Check middle element.

2.Conquer: Recursively search 1 subarray.

3.Combine: Trivial.

Example: Find 9

3	5	7	8	9	12	15



Binary search

Find an element in a sorted array:

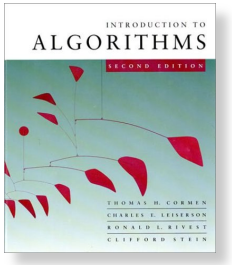
1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Example: Find 9

3 5 7 8 9 12 15



Binary search

Find an element in a sorted array:

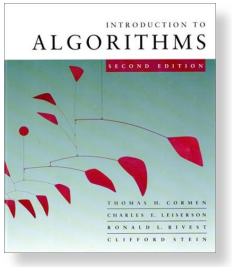
1.Divide: Check middle element.

2.Conquer: Recursively search 1 subarray.

3.Combine: Trivial.

Example: Find 9

3 5 7 8 9 12 15



Binary search

Find an element in a sorted array:

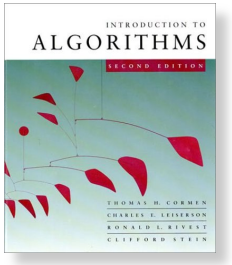
1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Example: Find 9

3 5 7 8 **9** 12 15



Binary search

Find an element in a sorted array:

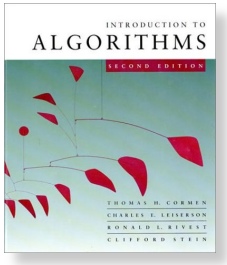
1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Example: Find 9

3 5 7 8 **9** 12 15



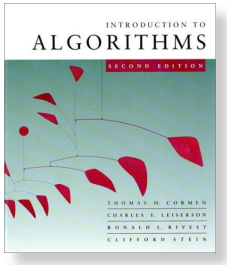
Recurrence for binary search

$$T(n) = 1\,T(n/2) + \Theta(1)$$

 # subproblems

subproblem size

work dividing and combining



Recurrence for binary search

$$T(n) = 1\,T(n/2) + \Theta(1)$$

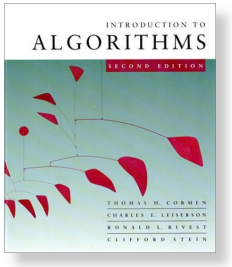
 # subproblems

subproblem size

work dividing and combining

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0)$$

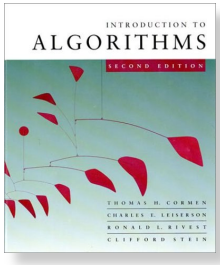
$$\Rightarrow T(n) = \Theta(\lg n) .$$



Powering a number

Problem: Compute a^n, where $n \in \mathbf{N}$.

Naive algorithm: $\Theta(n)$.



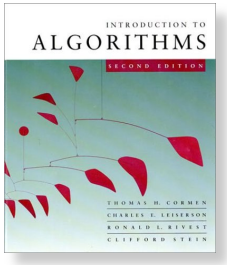
Powering a number

Problem: Compute a^n, where $n \in \mathbf{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$



Powering a number

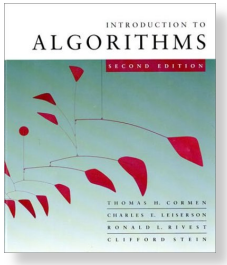
Problem: Compute a^n, where $n \in \mathbf{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n) .$$

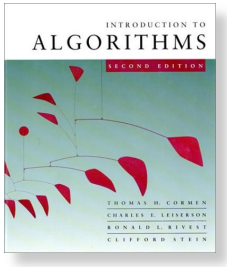


Matrix multiplication

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{11} = (a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} + \ldots + a_{1n} \cdot b_{n1})$$

$$c_{ij} = (a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + a_{i3} \cdot b_{3j} + \ldots + a_{in} \cdot b_{nj})$$

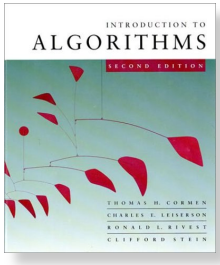


Matrix multiplication

Input: $A = [a_{ij}], B = [b_{ij}].$
Output: $C = [c_{ij}] = A \cdot B.$
$\Big\}$ $i, j = 1, 2, \ldots, n.$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

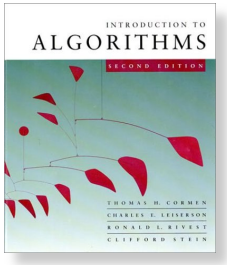
$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$



Standard algorithm

for $i \leftarrow 1$ **to** n
  do for $j \leftarrow 1$ **to** n
    do $c_{ij} \leftarrow 0$
      for $k \leftarrow 1$ **to** n
        do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Running time $= \Theta(n^3)$



Divide-and-conquer algorithm

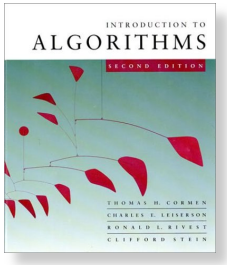
IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\} \begin{array}{l} 8 \text{ mults of } (n/2) \times (n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2) \times (n/2) \text{ submatrices} \end{array}$$



Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

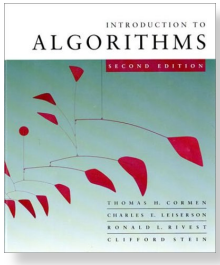
$$C = A \cdot B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dh \\ u &= cf + dg \end{aligned}\right\}$$

recursive

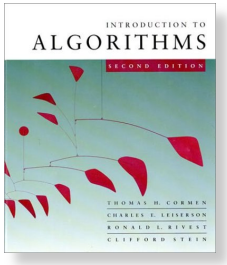
8 mults of $(n/2) \times (n/2)$ submatrices

4 adds of $(n/2) \times (n/2)$ submatrices



Standard algorithm

for $i \leftarrow 1$ **to** n
  do for $j \leftarrow 1$ **to** n
    do $c_{ij} \leftarrow 0$
      for $k \leftarrow 1$ **to** n
        do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$



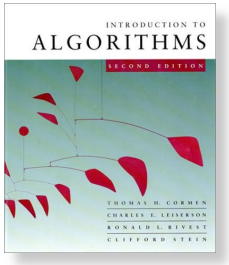
Analysis of D&C algorithm

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

submatrix size

work adding submatrices



Analysis of D&C algorithm

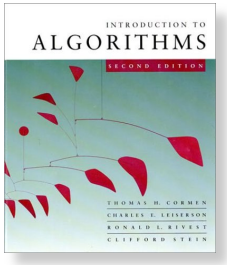
$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

submatrix size

work adding submatrices

$$n^{\log_b a} = n^{\log_2 8} = n^3 \implies \text{CASE 1} \implies T(n) = \Theta(n^3).$$



Analysis of D&C algorithm

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

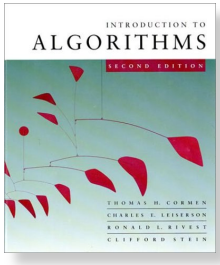
 # submatrices

submatrix size

work adding submatrices

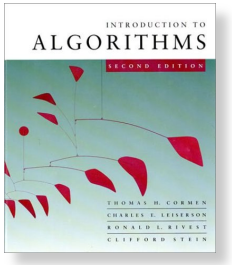
$$n^{\log_b a} = n^{\log_2 8} = n^3 \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^3).$$

No better than the ordinary algorithm.



Strassen's idea

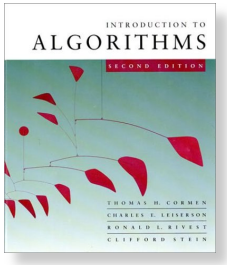
- Multiply 2×2 matrices with only 7 recursive mults.



Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

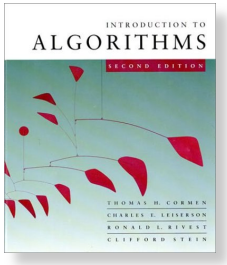


Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$



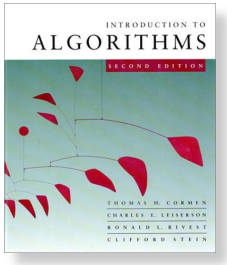
Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

7 mults, 18 adds/subs.
Note: No reliance on commutativity of mult!

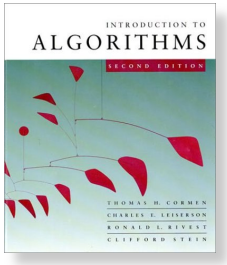


Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

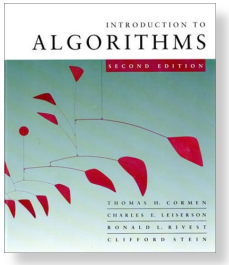
$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}
$$



Strassen's algorithm

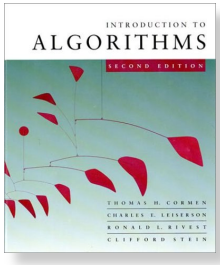
1. ***Divide:*** Partition A and B into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
2. ***Conquer:*** Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.
3. ***Combine:*** Form C using $+$ and $-$ on $(n/2)\times(n/2)$ submatrices.



Strassen's algorithm

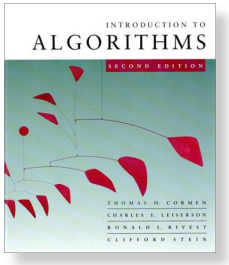
1. ***Divide:*** Partition A and B into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
2. ***Conquer:*** Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.
3. ***Combine:*** Form C using $+$ and $-$ on $(n/2)\times(n/2)$ submatrices.

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



Analysis of Strassen

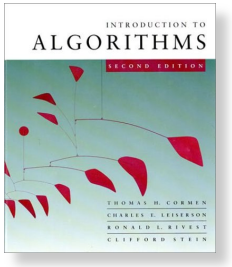
$$T(n) = 7\,T(n/2) + \Theta(n^2)$$



Analysis of Strassen

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

$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow$ CASE 1 $\Rightarrow T(n) = \Theta(n^{\lg 7})$.

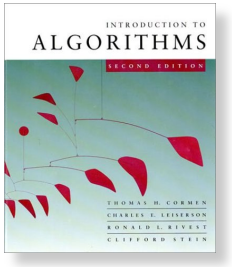


Analysis of Strassen

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

$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow$ CASE 1 $\Rightarrow T(n) = \Theta(n^{\lg 7})$.

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.



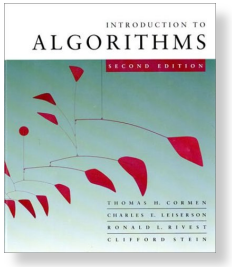
Analysis of Strassen

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

$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow$ CASE 1 $\Rightarrow T(n) = \Theta(n^{\lg 7})$.

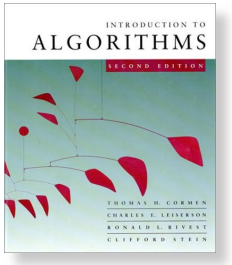
The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.

Best to date (of theoretical interest only): $\Theta(n^{2.376\cdots})$.



Conclusion

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method (so practice this math).
- The divide-and-conquer strategy often leads to efficient algorithms.



Why study algorithms and performance?

- Algorithms help us to understand ***scalability***.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a ***language*** for talking about program behavior.
- Performance is the ***currency*** of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

Thank you.

Q & A