CSE326: Analysis and Design of Algorithms

Lecture 1

Course Outline (Tentative)

- Basics: Asymptotic notation and recurrences
- Divide-and-Conquer
- Dynamic programming
- Greedy algorithms
- Amortized analysis
- NP-completeness
- Graph algorithms

Course Logistics

- Course updates will be posted on MS teams.
  - Team code posted to your group.

- Grade distribution (tentative):
  - Coursework and labs: 25%
  - Midterm: 25%
  - Final: 50%

- Academic integrity:
  - You must complete the sheets and assignments independently. Avoid any violation of academic integrity.

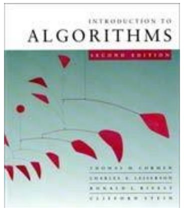
Textbook

- Main textbook: *Introduction to Algorithms*, 3rd Edition by Cormen, Leiserson, Rivest, & Stein (CLRS).

- Other useful references will be posted when needed.

Lecture 1: Asymptotic Notation and Recurrences

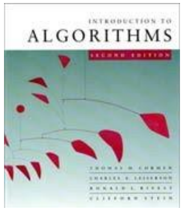
The following slides are an edited version of slides by professors *Erik D. Demaine* and *Charles E. Leiserson* provided by MIT OCW (CC license).



Analysis of algorithms

The theoretical study of computer-program performance and resource usage.

In practice, are there other aspects that we care about?



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!

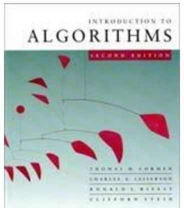
Analysis of Algorithms

- Goal: Predicting the resources that an algorithm requires
- Efficiency metrics:
  - **Running time**
  - Memory (space) used
  - Communication overhead, e.g., packets sent over a network in a distributed algorithm

- The efficiency is measured with respect to the input size.
  - Note: The input size is defined based on the problem.
    - For a sorting problem, it is usually the number of items.
    - For a graph problem, it is usually expressed in two numbers: number of edges and number of vertices.
    - For integer multiplication, it is the number of bits of the input numbers.

Analysis of Algorithms

- In most cases, assumed model of computation: the RAM model
- The RAM model contains instructions commonly found in real computers:
  - Arithmetic (add, subtract, multiply, divide, remainder, floor, ceiling),
  - Data movement (load, store, copy)
  - Control (conditional and unconditional branch, subroutine call and return).
- Each basic instruction is assumed to take a constant amount of time.
- Data types: integer and floating-point types
  - Each word of data has a limited size.



The problem of sorting

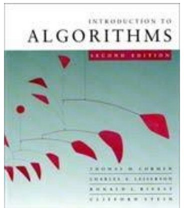
Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

Example:

Input: 8 2 4 9 3 6

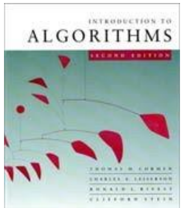
Output: 2 3 4 6 8 9



Insertion sort

"pseudocode"

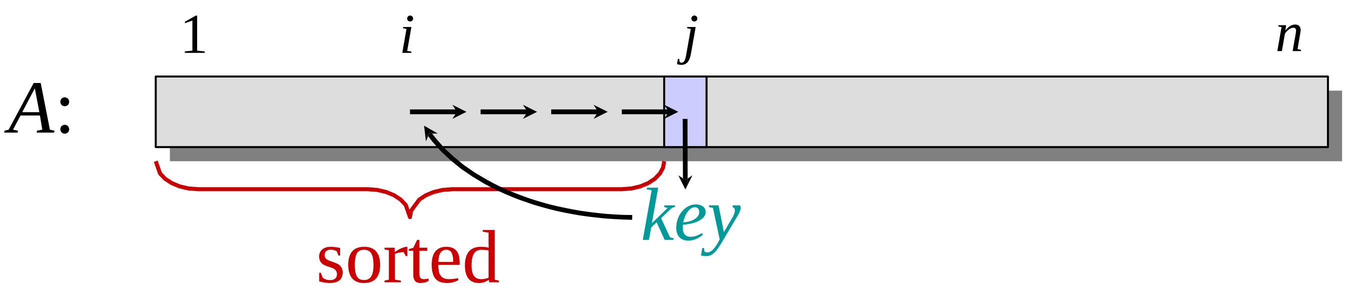
```
INSERTION-SORT (A, n)        ⊳ A[1 . . n]
    for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
           A[i+1] = key
```

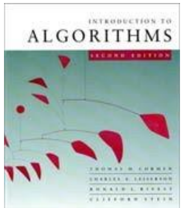


Insertion sort

“pseudocode”

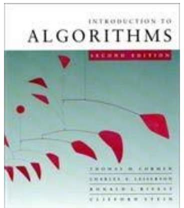
```
INSERTION-SORT (A, n)      ⊳ A[1 . . n]
    for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
           A[i+1] = key
```





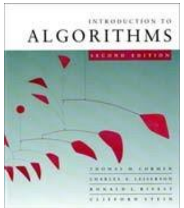
Example of insertion sort

8 2 4 9 3 6



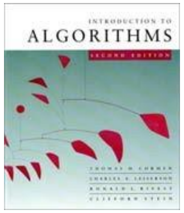
Example of insertion sort

8 2 4 9 3 6



Example of insertion sort

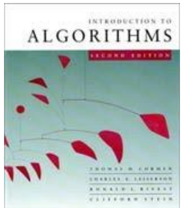
8	2	4	9	3	6
2	8	4	9	3	6



Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6



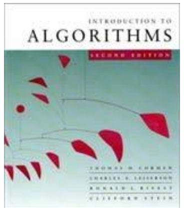
Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

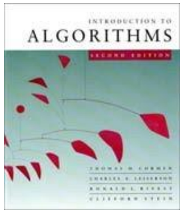
2 4 8 9 3 6

Introduction to Algorithms



Example of insertion sort

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6



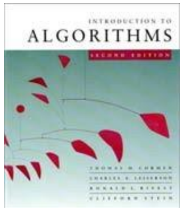
Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

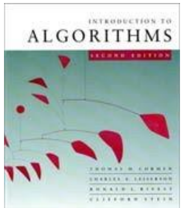
2 4 8 9 3 6

2 4 8 9 3 6



Example of insertion sort

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6



Example of insertion sort

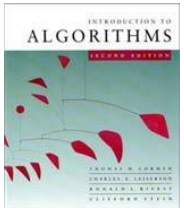
8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6



Example of insertion sort

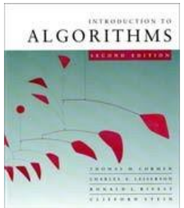
8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

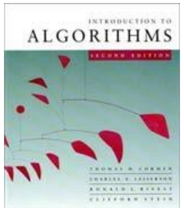
2 4 8 9 3 6

2 3 4 8 9 6



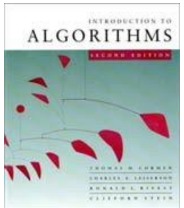
Example of insertion sort

8	2	4	9	3	6	
2	8	4	9	3	6	
2	4	8	9	3	6	
2	4	8	9	3	6	
2	3	4	8	9	6	
2	3	4	6	8	9	*done*



Running time of insertion sort

- The running time depends on the input: an already sorted sequence is easier to sort.
- Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.



Kinds of analyses

Worst-case: (usually)

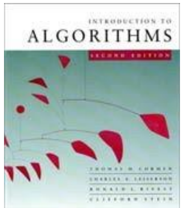
- $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes)

- $T(n)$ = expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs.

Best-case: (bogus)

- Cheat with a slow algorithm that works fast on *some* input.



Machine-independent time

What is insertion sort's worst-case time?

- It depends on the speed of our computer:
  - relative speed (on the same machine),
  - absolute speed (on different machines).

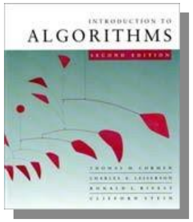
BIG IDEA:

- Ignore machine-dependent constants.
- Look at ***growth*** of $T(n)$ as $n \to \infty$.

"Asymptotic Analysis"

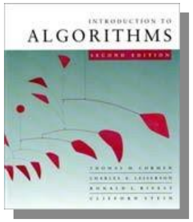
Next

- **Asymptotic Notation**
  - O-, Ω-, and Θ-notation
- **Recurrences**
  - Substitution method
  - Recursion tree
  - Master method



O-notation (upper bounds)

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.



O-notation (upper bounds)

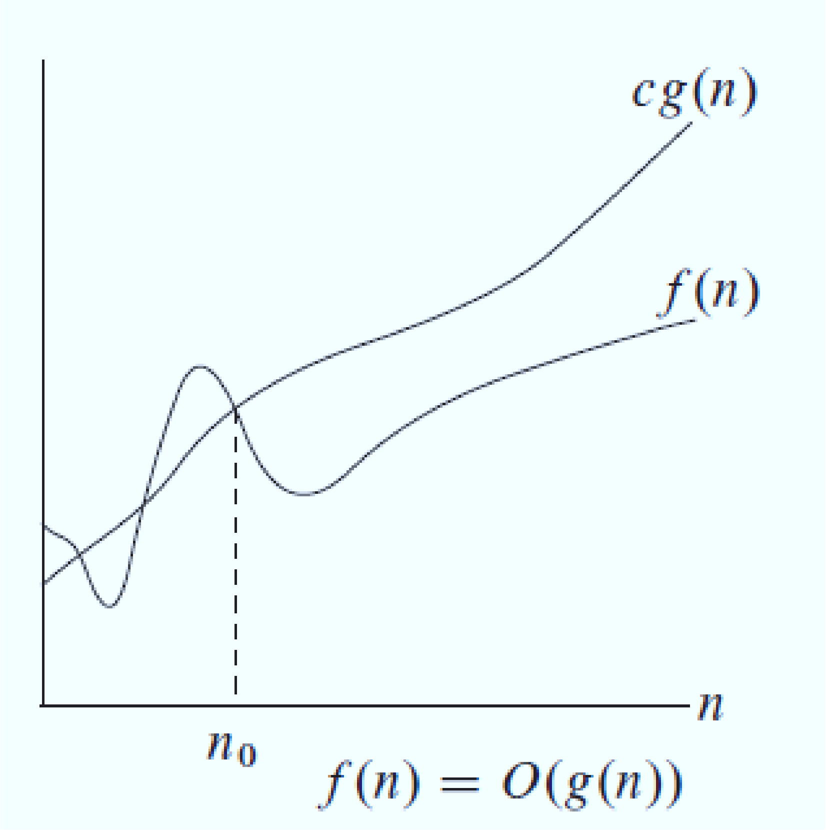
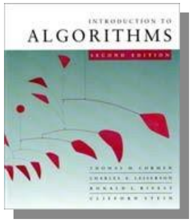


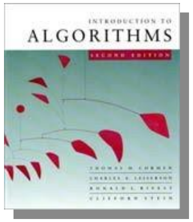
Figure 3.1(b) from CLRS



O-notation (upper bounds)

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

EXAMPLE: $2n^2 = O(n^3)$ $\quad$ $(c = 1, n_0 = 2)$

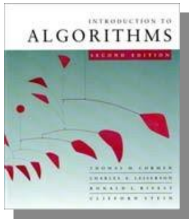


O-notation (upper bounds)

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

EXAMPLE: $2n^2 = O(n^3)$ $\quad (c = 1, n_0 = 2)$

functions, not values



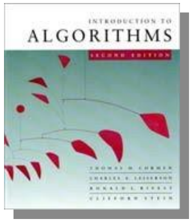
O-notation (upper bounds)

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

EXAMPLE: $2n^2 = O(n^3)$ $\quad (c = 1, n_0 = 2)$

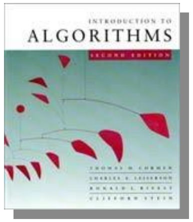
functions, not values

funny, "one-way" equality



Set definition of O-notation

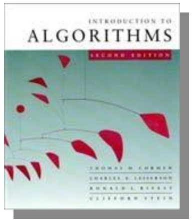
$O(g(n)) = \{ f(n)$: there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$ $\}$



Set definition of O-notation

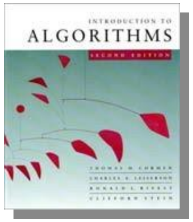
$$O(g(n)) = \{ f(n) : \text{there exist constants } c > 0,\ n_0 > 0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \}$$

EXAMPLE: $2n^2 \in O(n^3)$



Macro substitution

Convention: A set in a formula represents an anonymous function in the set.



Macro substitution

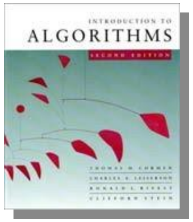
Convention: A set in a formula represents an anonymous function in the set.

EXAMPLE: $f(n) = n^3 + O(n^2)$

means

$f(n) = n^3 + h(n)$

for some $h(n) \in O(n^2)$.



Macro substitution

Convention: A set in a formula represents an anonymous function in the set.

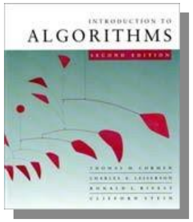
EXAMPLE: $n^2 + O(n) = O(n^2)$

means

for any $f(n) \in O(n)$:

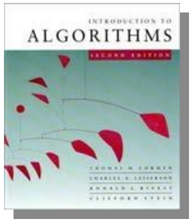
$$n^2 + f(n) = h(n)$$

for some $h(n) \in O(n^2)$.



Ω-notation (lower bounds)

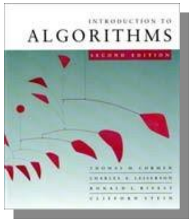
O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.



Ω-notation (lower bounds)

O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$. We use Ω to indicate lower bounds.

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



Ω-notation (lower bounds)

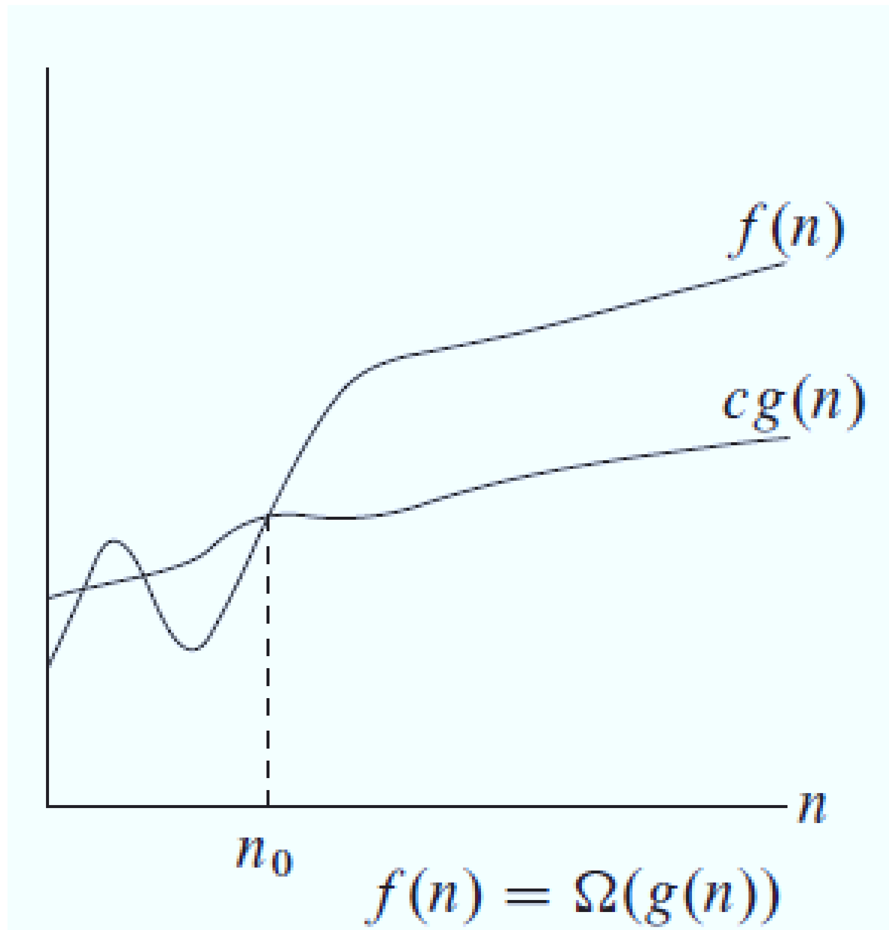
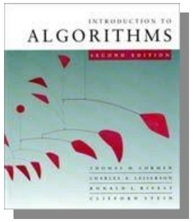


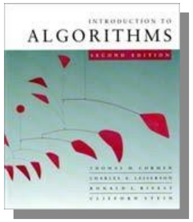
Figure 3.1(c) from CLRS



Ω-notation (lower bounds)

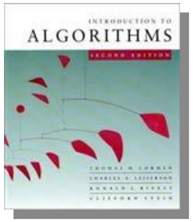
$$\Omega(g(n)) = \{ f(n) : \text{there exist constants } c > 0,\ n_0 > 0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0 \}$$

EXAMPLE: $\sqrt{n} = \Omega(\lg n) \quad (c = 1,\ n_0 = 16)$



Θ-notation (tight bounds)

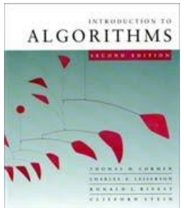
$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$



Θ-notation (tight bounds)

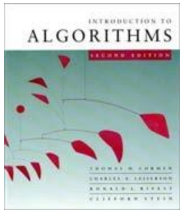
$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

EXAMPLE: $\frac{1}{2}n^2 - 2n = \Theta(n^2)$



Θ-notation (tight bounds)

$\Theta(g(n)) = \{ f(n) :$ there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$ $\}$



Asymptotic Notation

- Θ-, O-, and Ω -notation

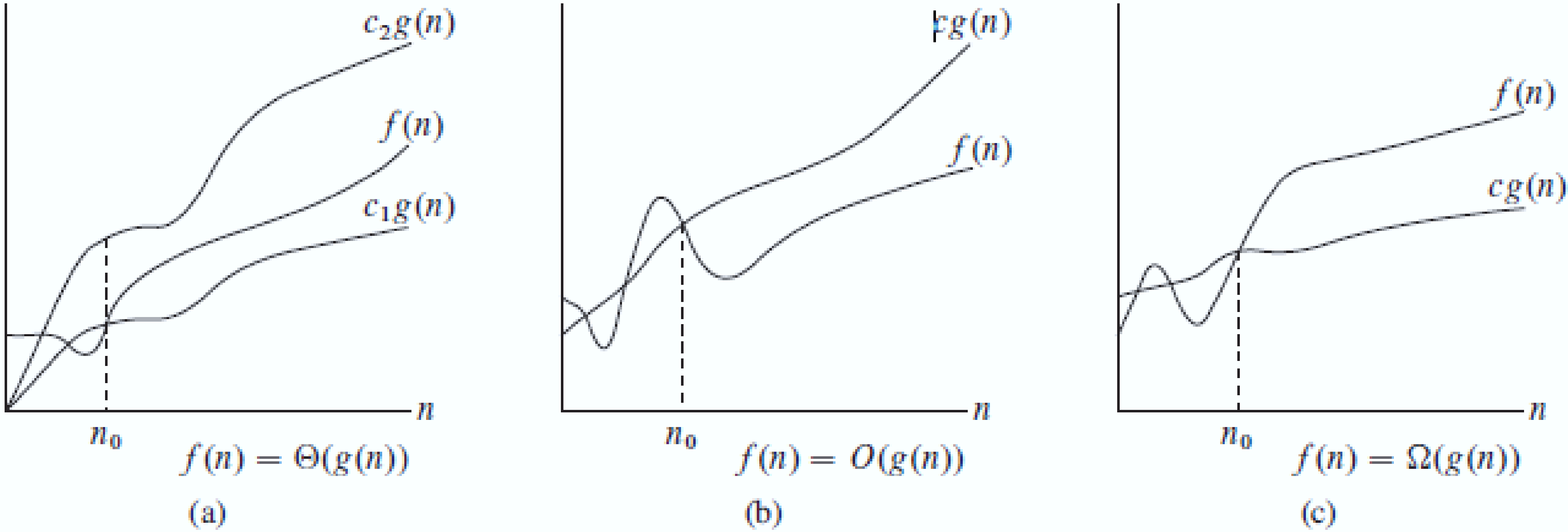
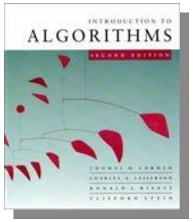


Figure 3.1 from CLRS



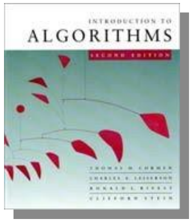
o-notation and ω-notation

O-notation and Ω-notation are like $\leq$ and $\geq$.
o-notation and ω-notation are like $<$ and $>$.

$o(g(n)) = \{ f(n)$: for any constant $c > 0$,
there is a constant $n_0 > 0$
such that $0 \leq f(n) < cg(n)$
for all $n \geq n_0$ $\}$

EXAMPLE: $2n^2 = o(n^3)$

$2n^2 < cn^3$ for any $c > 0$ and all $n \geq n_0$ where $n_0 > 2/c$



o-notation and ω-notation

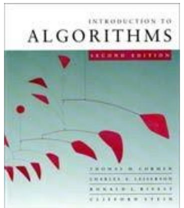
O-notation and Ω-notation are like $\leq$ and $\geq$.
o-notation and ω-notation are like $<$ and $>$.

$$\omega(g(n)) = \{ f(n) : \text{for any constant } c > 0, \text{ there is a constant } n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0 \}$$

EXAMPLE: $\sqrt{n} = \omega(\lg n)$

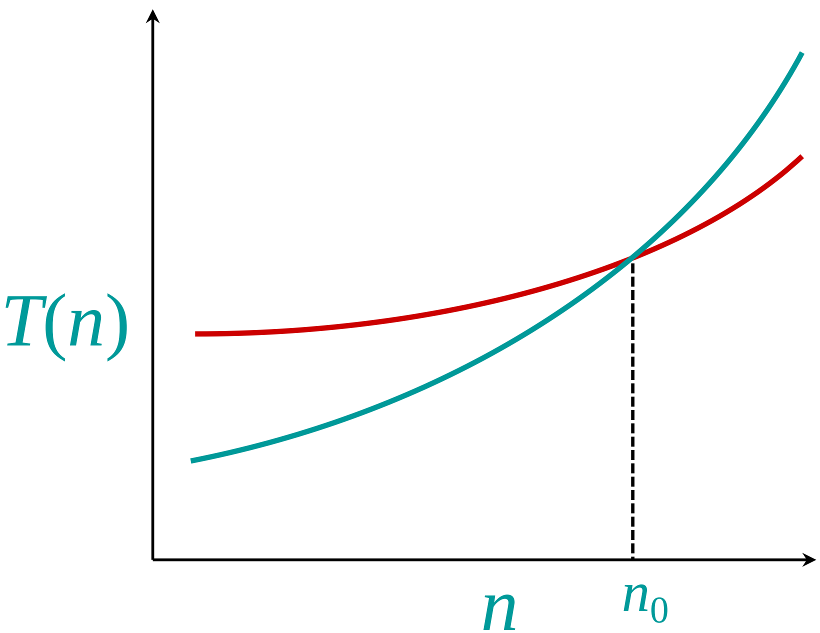
o-notation and ω-notation

	Limit Definition
$f(n) = o(g(n))$	$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$
$f(n) = \omega(g(n))$	$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$



Asymptotic performance

When n gets large enough, a $\Theta(n^2)$ algorithm *always* beats a $\Theta(n^3)$ algorithm.



- We shouldn’t ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.

Exercise

- Sort the following functions by the order of their growth:
  - $5n^2$
  - $n \lg n$
  - $\log_3 n^2$
  - $\lg n$
  - $(\lg n)^2$
  - $n!$
  - $(n+1)!$
  - 2^n
  - $3^{\lg n}$

Note: Section 3.2 in CLRS has a review of standard mathematical functions that can help with this problem.

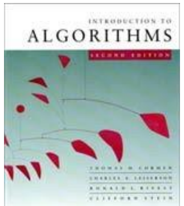
Solution

- From lowest to highest:

  Functions written on the same line are in the same group.

  f(n) and g(n) are in the same group if and only if $f(n) = \theta(g(n))$

  - $\lg n$, $\log_3 n^2$
  - $(\lg n)^2$
  - $n \lg n$
  - $3^{\lg n}$
  - $5n^2$
  - 2^n
  - $n!$
  - $(n+1)!$



Insertion sort analysis

Worst case: Input reverse sorted.

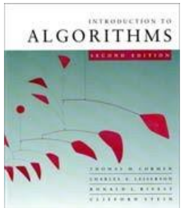
$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2) \qquad \text{[arithmetic series]}$$

Average case: All permutations equally likely.

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta(n^2)$$

Is insertion sort a fast sorting algorithm?

- Moderately so, for small n.
- Not at all, for large n.

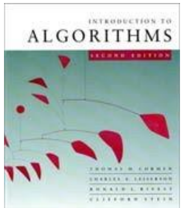


Merge sort

MERGE-SORT $A[1 \,.\,.\, n]$

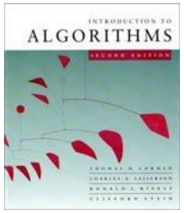
1. If $n = 1$, done.
2. Recursively sort $A[\, 1 \,.\,.\, \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \,.\,.\, n\,]$.
3. "***Merge***" the 2 sorted lists.

Key subroutine: **MERGE**



Merging two sorted arrays

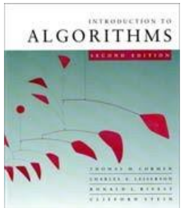
20	12
13	11
7	9
2	1



Merging two sorted arrays

20	12
13	11
7	9
2	1

1

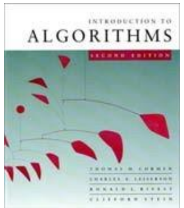


Merging two sorted arrays

20	12		20	12
13	11		13	11
7	9		7	9
2	1		2	

1

Introduction to Algorithms

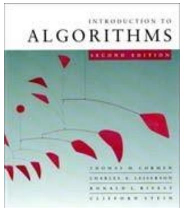


Merging two sorted arrays

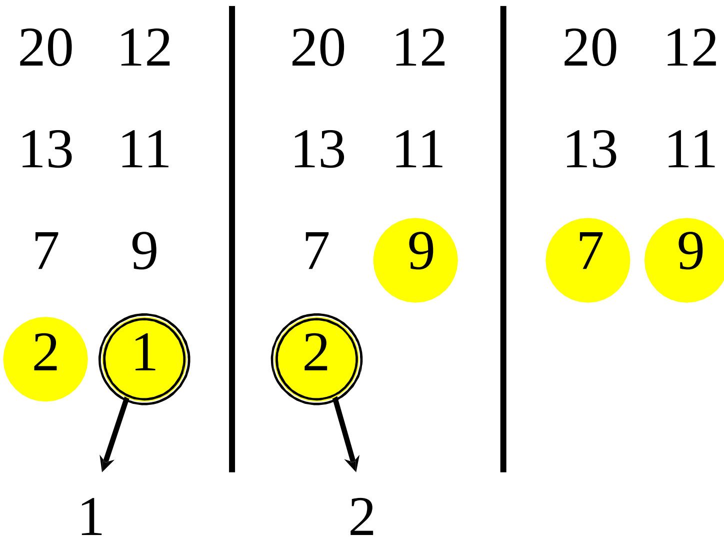
20	12		20	12
13	11		13	11
7	9		7	9
2	1		2	

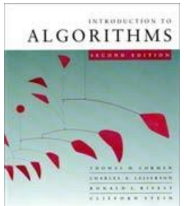
1 2

Introduction to Algorithms

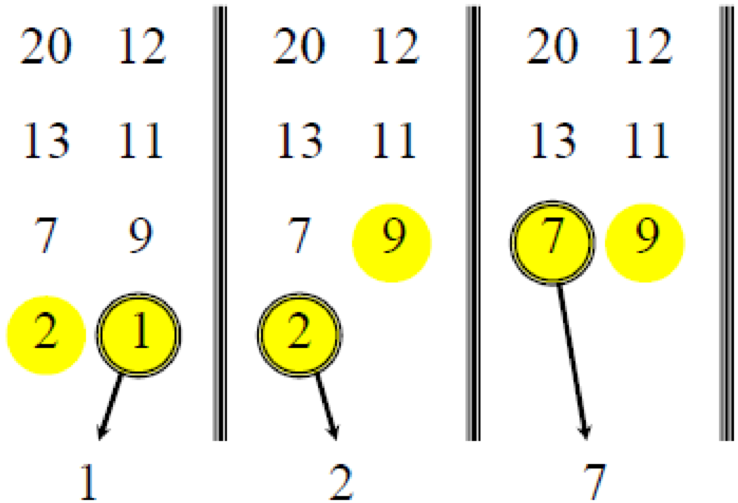


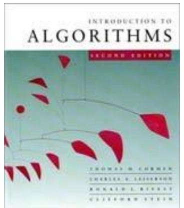
Merging two sorted arrays



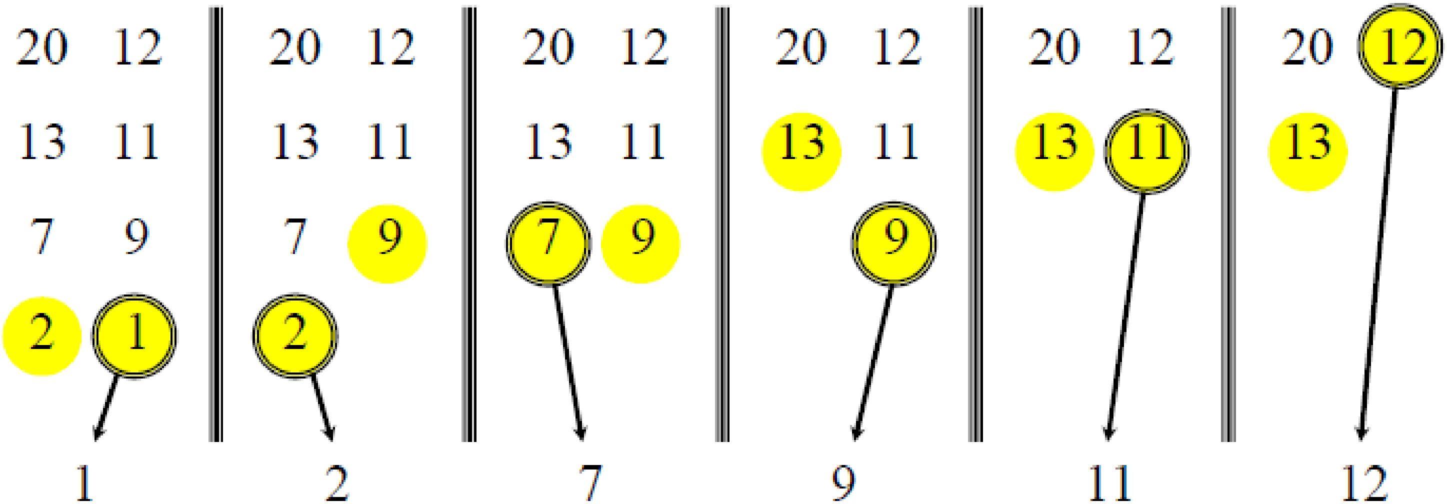


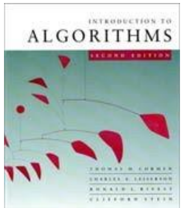
Merging two sorted arrays



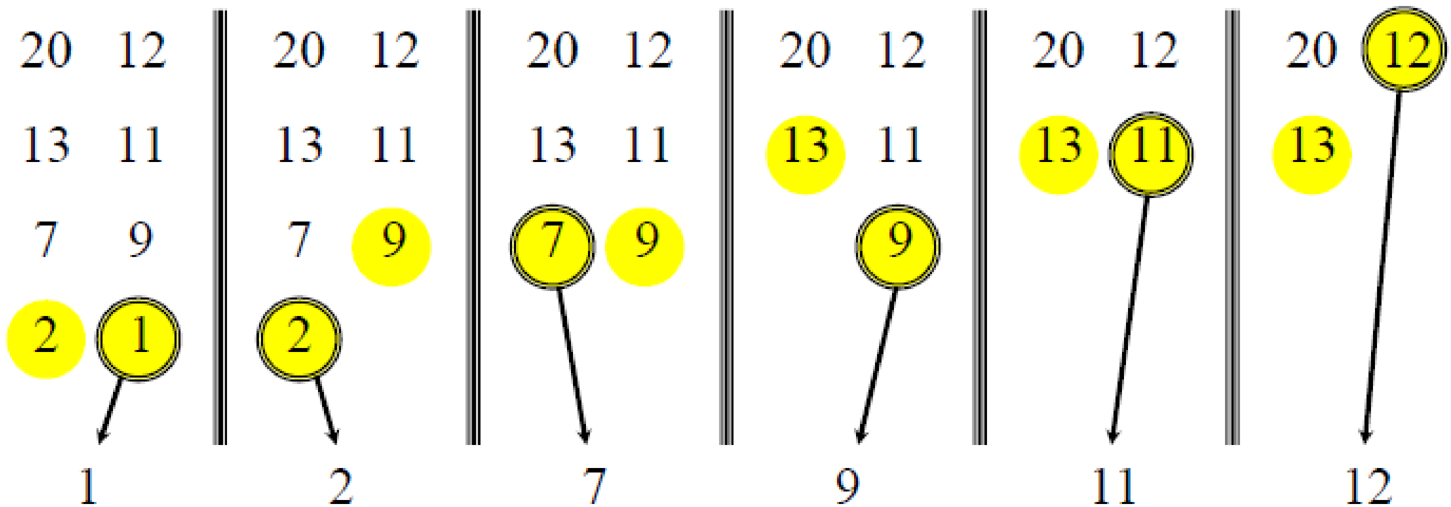


Merging two sorted arrays

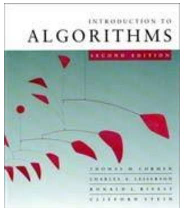




Merging two sorted arrays



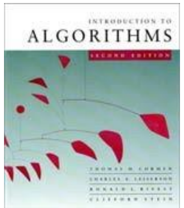
Time = $\Theta(n)$ to merge a total of n elements (linear time).



Analyzing merge sort

$T(n)$	**MERGE-SORT** $A[1\,.\,.\,n]$
$\Theta(1)$	1. If $n = 1$, done.
$2T(n/2)$	2. Recursively sort $A[\,1\,.\,.\,\lceil n/2\rceil]$ and $A[\lceil n/2\rceil +1\,.\,.\,n\,]$.
$\Theta(n)$	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.



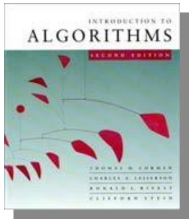
Recurrence for merge sort

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

- 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.
- Next, we will discuss several ways to find a good upper bound on $T(n)$.

Solving Recurrences

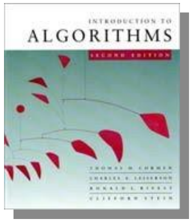
- Iterating the recurrence
- **Substitution method**
- **Recursion tree**
- **Master method**



Substitution method

The most general method:

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



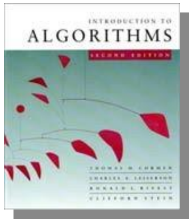
Substitution method

The most general method:

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

EXAMPLE: $T(n) = 4T(n/2) + n$

- [Assume that $T(1) = \Theta(1)$.]
- Guess $O(n^3)$. (Prove O and Ω separately.)
- Assume that $T(k) \leq ck^3$ for $k < n$.
- Prove $T(n) \leq cn^3$ by induction.

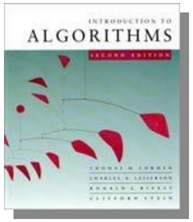


Example of substitution

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\leq 4c(n/2)^3 + n \\
&= (c/2)n^3 + n \\
&= cn^3 - ((c/2)n^3 - n) \longleftarrow \textit{desired} - \textit{residual} \\
&\leq cn^3 \longleftarrow \textit{desired}
\end{aligned}
$$

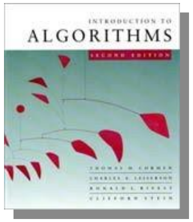
whenever $(c/2)n^3 - n \geq 0$, for example, if $c \geq 2$ and $n \geq 1$.

residual



Example (continued)

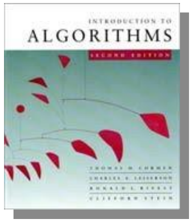
- We must also handle the initial conditions, that is, ground the induction with base cases.
- ***Base:*** $T(n) = \Theta(1)$ for all $n < n_0$, where n_0 is a suitable constant.
- For $1 \leq n < n_0$, we have "$\Theta(1)$" $\leq cn^3$, if we pick c big enough.



Example (continued)

- We must also handle the initial conditions, that is, ground the induction with base cases.
- ***Base:*** $T(n) = \Theta(1)$ for all $n < n_0$, where n_0 is a suitable constant.
- For $1 \leq n < n_0$, we have "$\Theta(1)$" $\leq cn^3$, if we pick c big enough.

This bound is not tight!



A tighter upper bound?

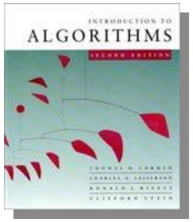
We shall prove that $T(n) = O(n^2)$.

A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4c(n/2)^2 + n \\ &= cn^2 + n \\ &= O(n^2) \end{aligned}$$



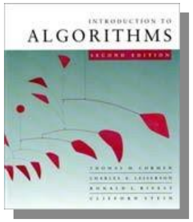
A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4c(n/2)^2 + n \\
&= cn^2 + n \\
&= \cancel{O(n^2)}
\end{aligned}
$$

Wrong! We must prove the I.H.

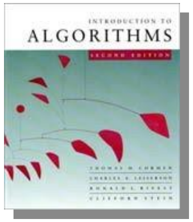


A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4c(n/2)^2 + n \\ &= cn^2 + n \\ &= \cancel{O(n^2)} \quad \textit{\textbf{Wrong!}} \text{ We must prove the I.H.} \\ &= cn^2 - (-n) \quad [\text{ desired} - \text{residual }] \\ &\le cn^2 \quad \text{for } \textit{\textbf{no}} \text{ choice of } c > 0. \text{ Lose!} \end{aligned}$$

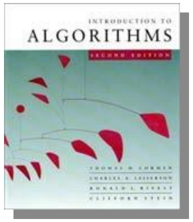


A tighter upper bound!

IDEA: Strengthen the inductive hypothesis.

• ***Subtract*** a low-order term.

Inductive hypothesis: $T(k) \le c_1 k^2 - c_2 k$ for $k < n$.



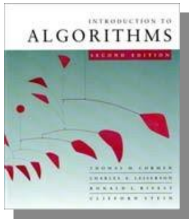
A tighter upper bound!

IDEA: Strengthen the inductive hypothesis.

- ***Subtract*** a low-order term.

Inductive hypothesis: $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$.

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&= 4(c_1(n/2)^2 - c_2(n/2)) + n \\
&= c_1 n^2 - 2c_2 n + n \\
&= c_1 n^2 - c_2 n - (c_2 n - n) \\
&\leq c_1 n^2 - c_2 n \quad \text{if } c_2 \geq 1.
\end{aligned}
$$



A tighter upper bound!

IDEA: Strengthen the inductive hypothesis.

- ***Subtract*** a low-order term.

Inductive hypothesis: $T(k) \le c_1k^2 - c_2k$ for $k < n$.

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&= 4(c_1(n/2)^2 - c_2(n/2)) + n \\
&= c_1n^2 - 2c_2n + n \\
&= c_1n^2 - c_2n - (c_2n - n) \\
&\le c_1n^2 - c_2n \quad \text{if } c_2 \ge 1.
\end{aligned}
$$

Pick c_1 big enough to handle the initial conditions.

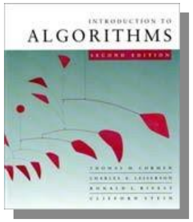
Exercises

- Solve the following recurrence using the substitution method.

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

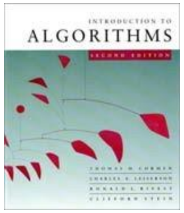
- Solve this recurrence:

$$T(n) = 2T\left(\lfloor \sqrt{n} \rfloor\right) + \lg n\ ,$$



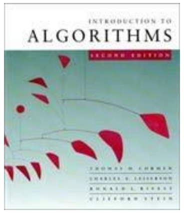
Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion tree method is good for generating guesses for the substitution method.
- When using a recursion tree to generate a good guess, a small amount of "sloppiness," is tolerated, since the guess will be verified later on.



Recursion tree – Example 1

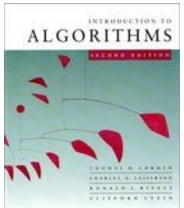
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recursion tree – Example 1

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

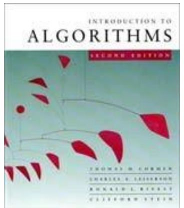
$$T(n)$$



Recursion tree – Example 1

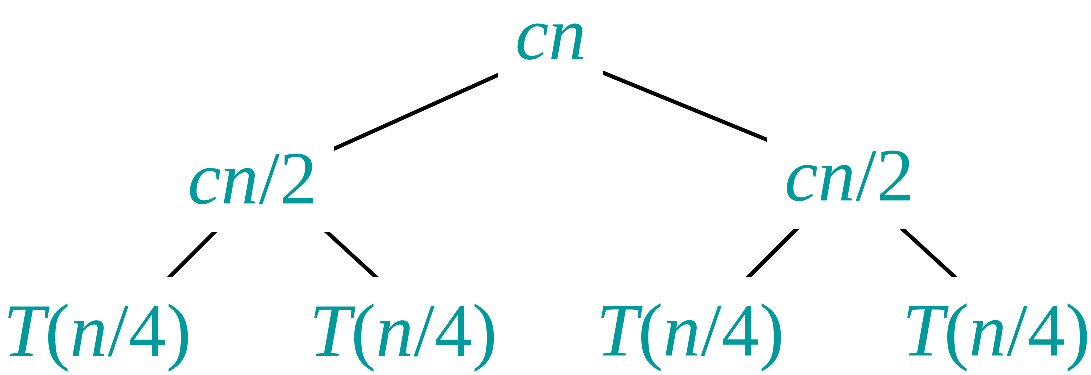
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

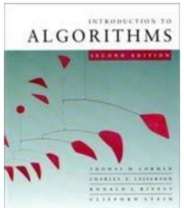
$$
\begin{array}{ccc}
 & cn & \\
\diagup & & \diagdown \\
T(n/2) & & T(n/2)
\end{array}
$$



Recursion tree – Example 1

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





Recursion tree – Example 1

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

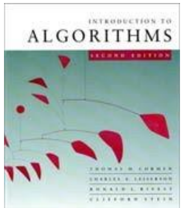
cn

$cn/2$ $cn/2$

$cn/4$ $cn/4$ $cn/4$ $cn/4$

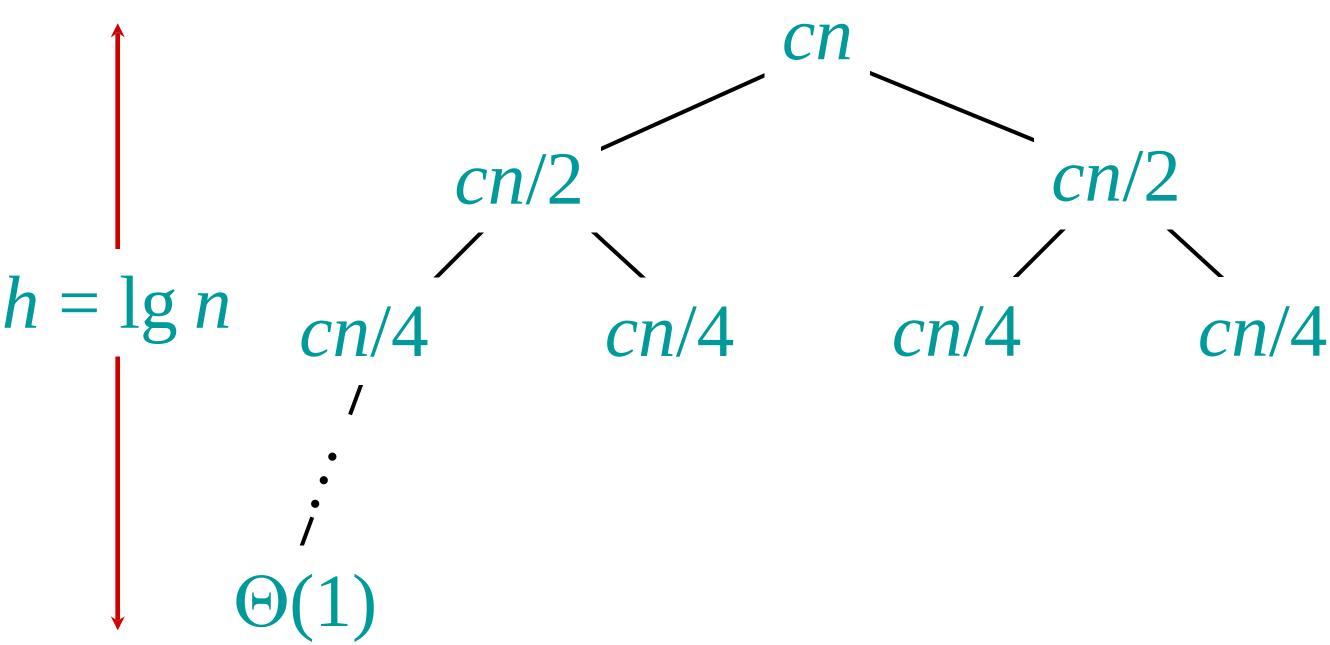
$\vdots$

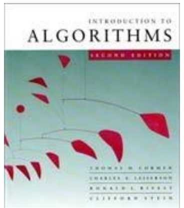
$\Theta(1)$



Recursion tree – Example 1

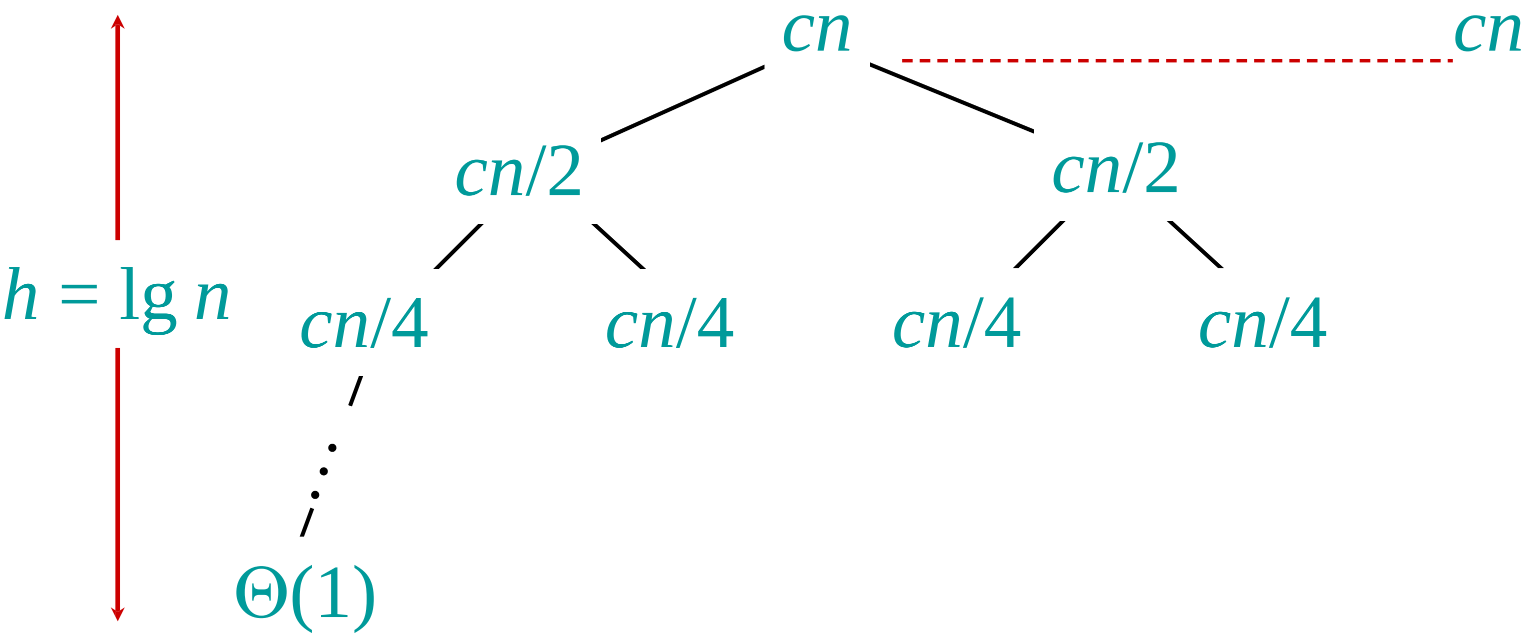
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

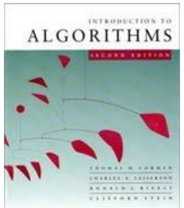




Recursion tree – Example 1

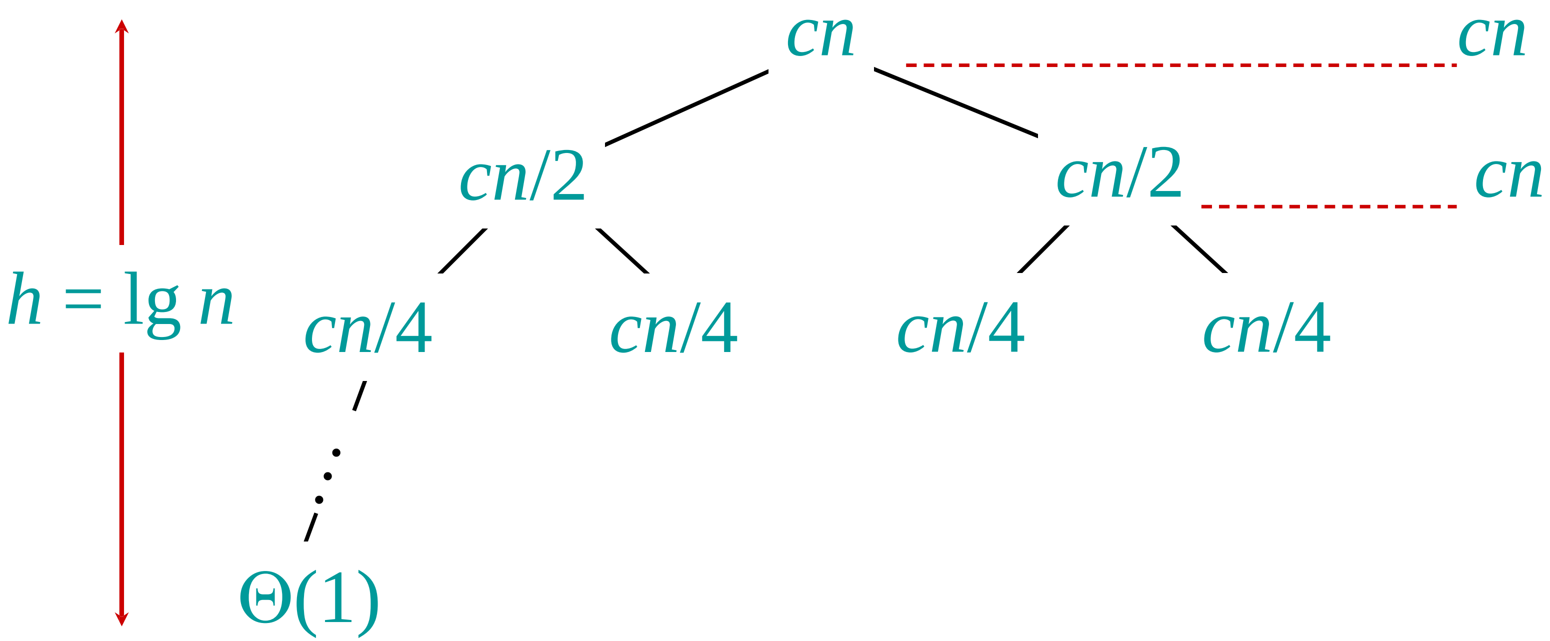
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

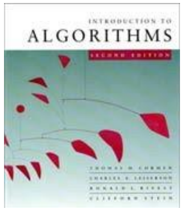




Recursion tree – Example 1

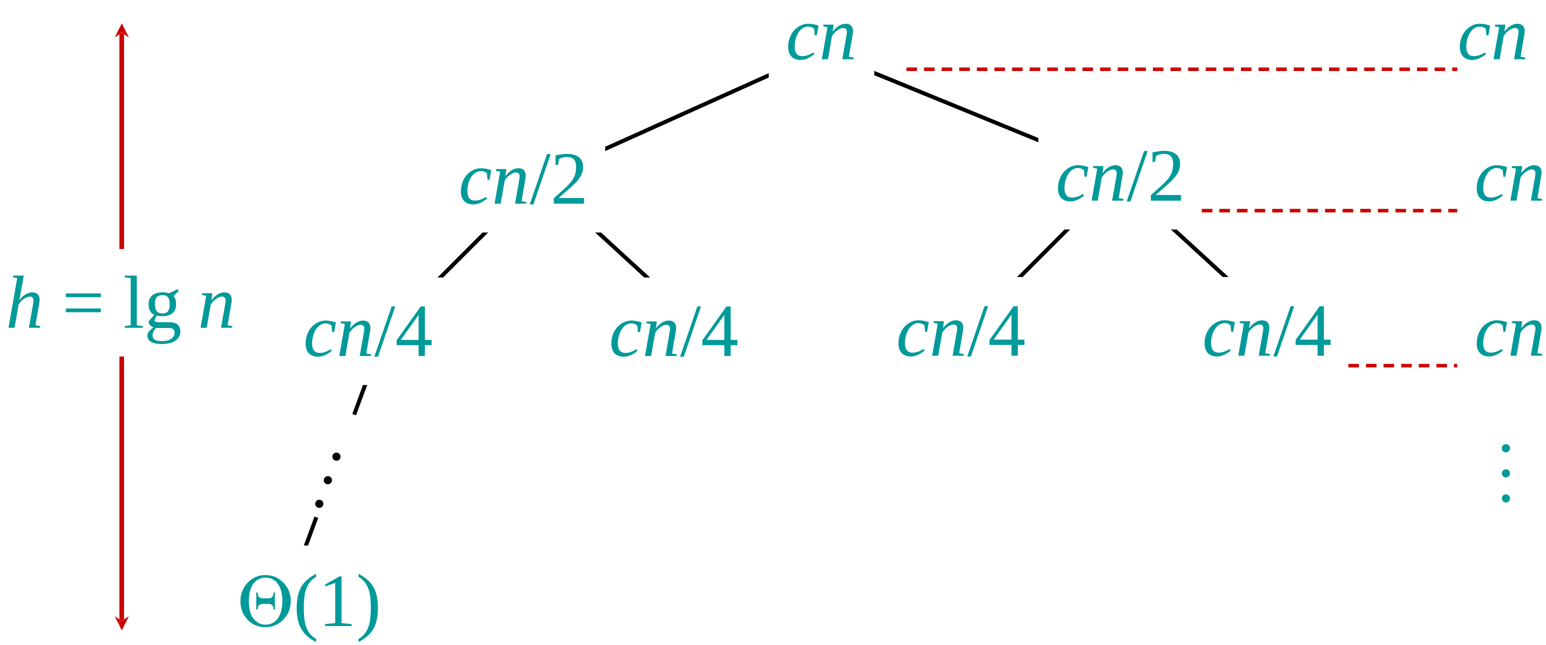
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

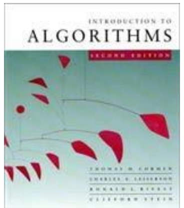




Recursion tree – Example 1

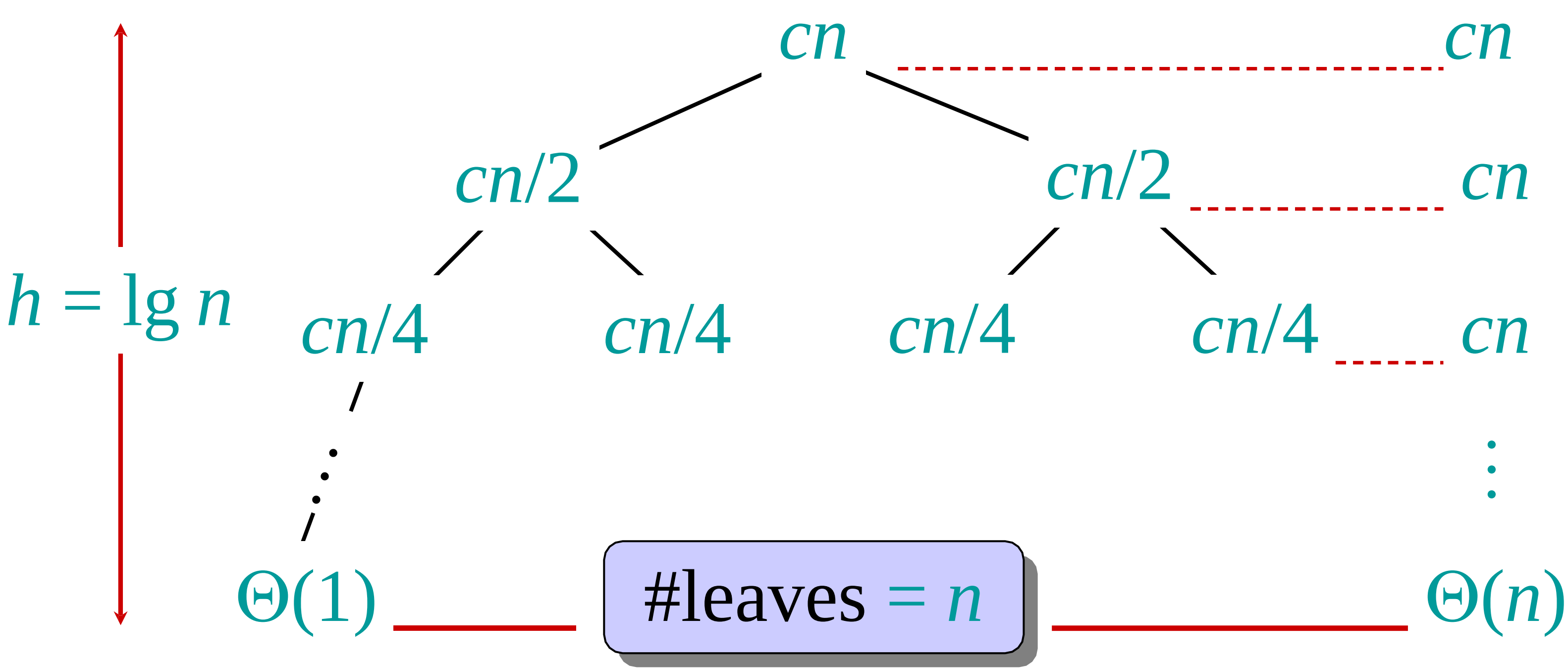
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

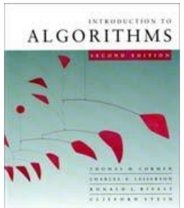




Recursion tree – Example 1

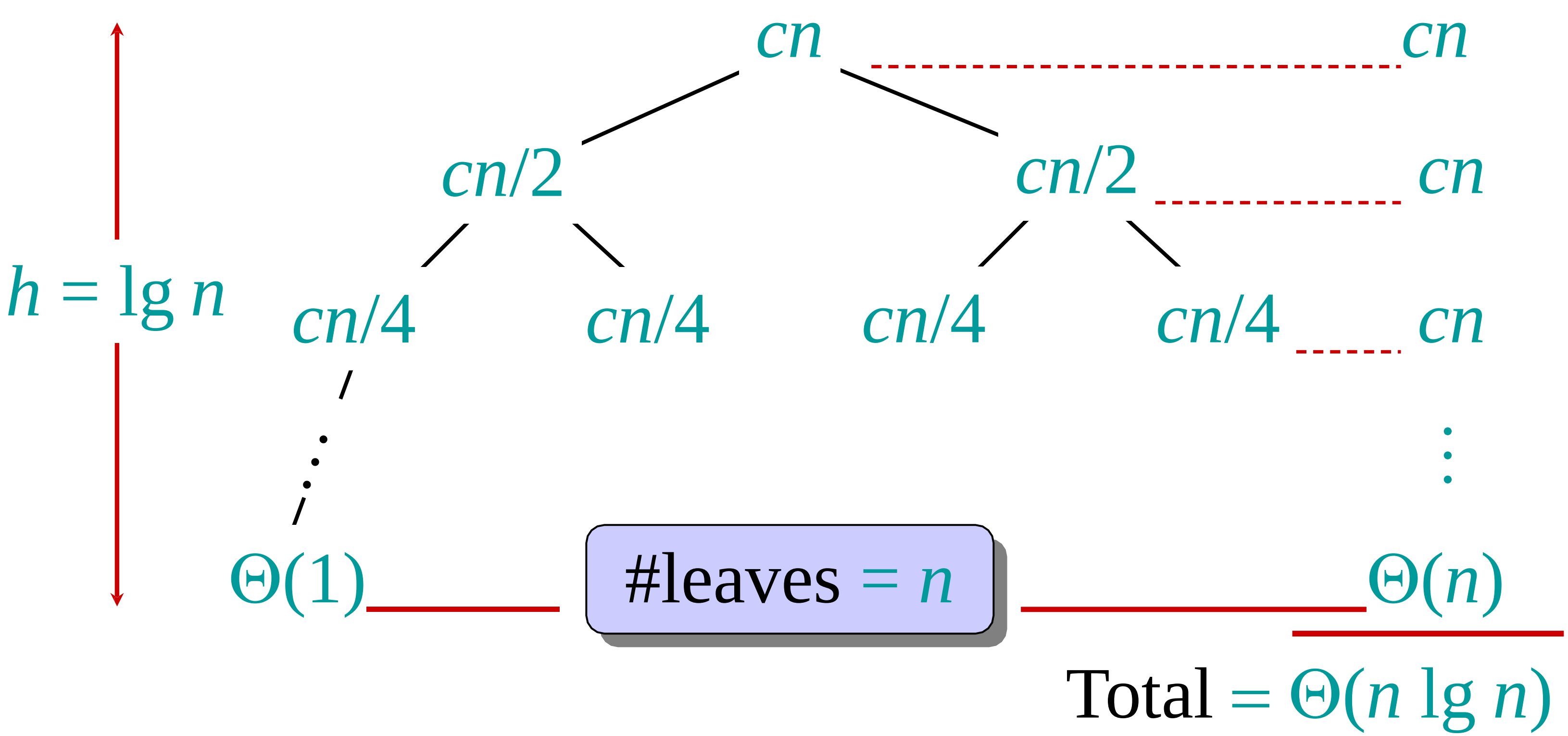
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

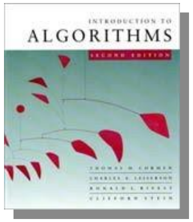




Recursion tree – Example 1

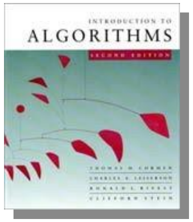
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.





Recursion tree – Example 2

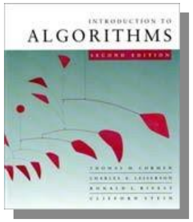
Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Recursion tree – Example 2

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

$$T(n)$$

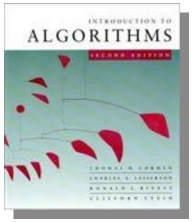


Recursion tree – Example 2

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

$$n^2$$

$$T(n/4) \qquad\qquad T(n/2)$$



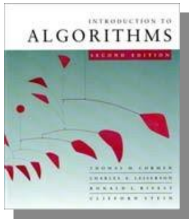
Recursion tree – Example 2

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

$$n^2$$

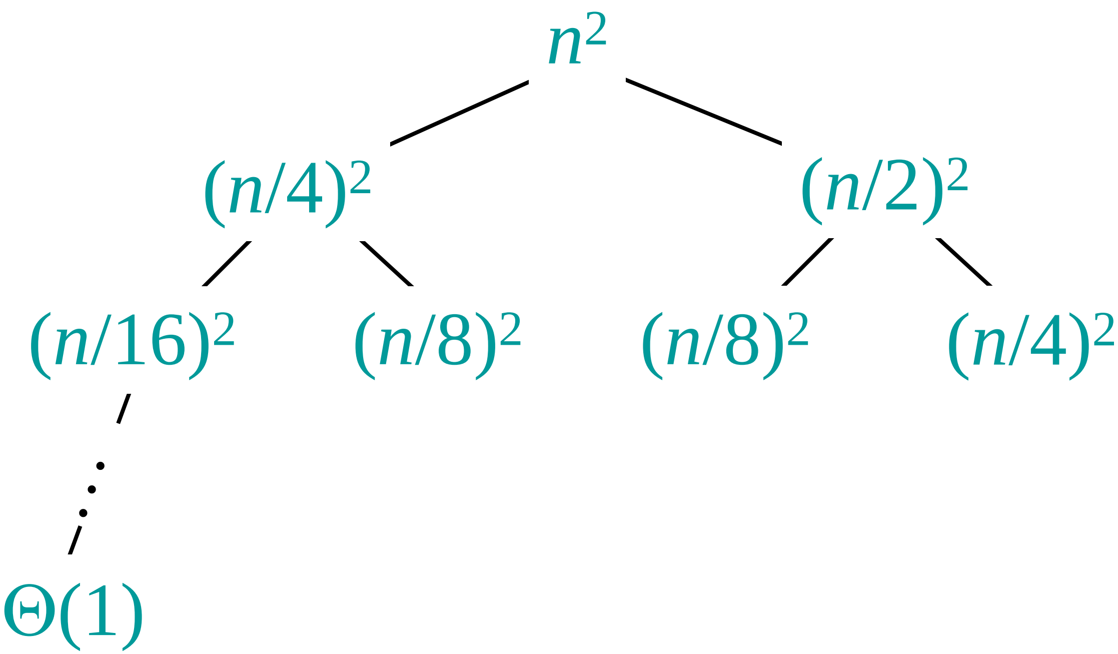
$$(n/4)^2 \qquad\qquad (n/2)^2$$

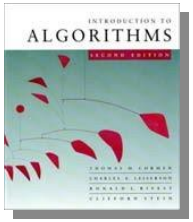
$$T(n/16) \quad T(n/8) \quad T(n/8) \quad T(n/4)$$



Recursion tree – Example 2

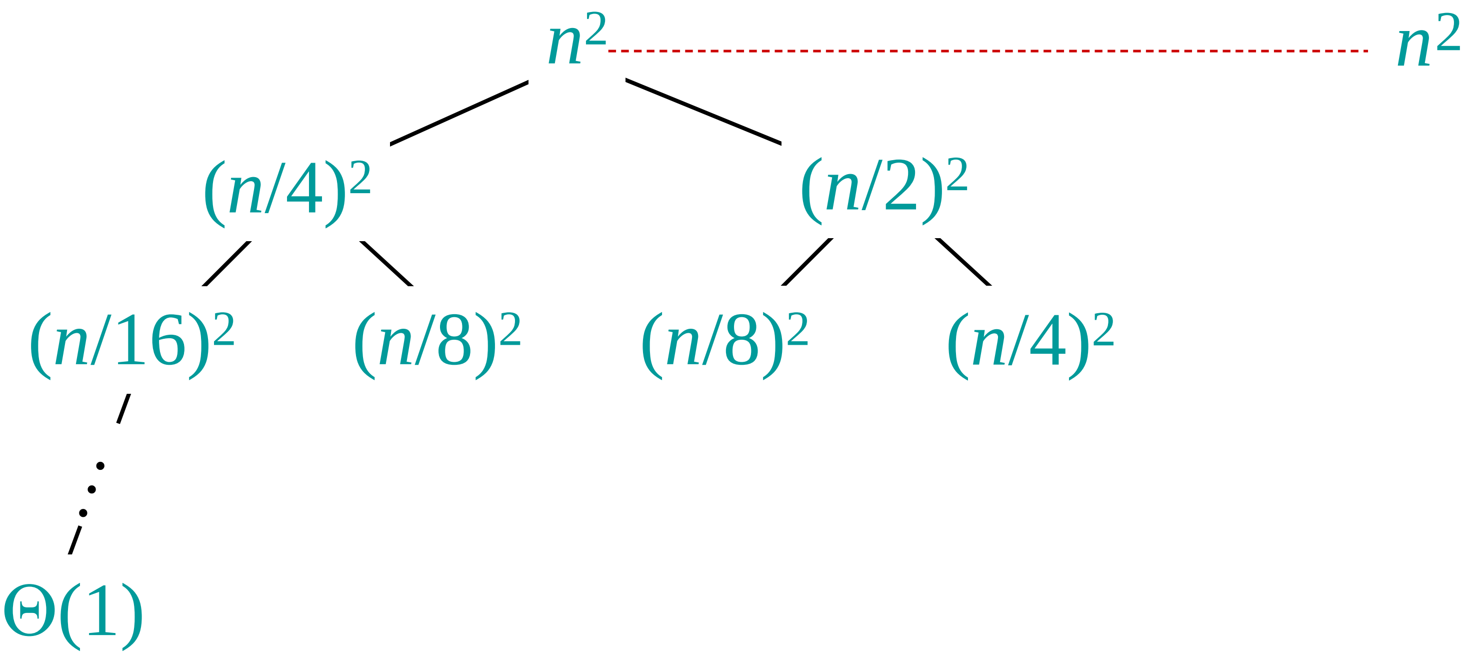
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

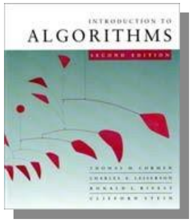




Recursion tree – Example 2

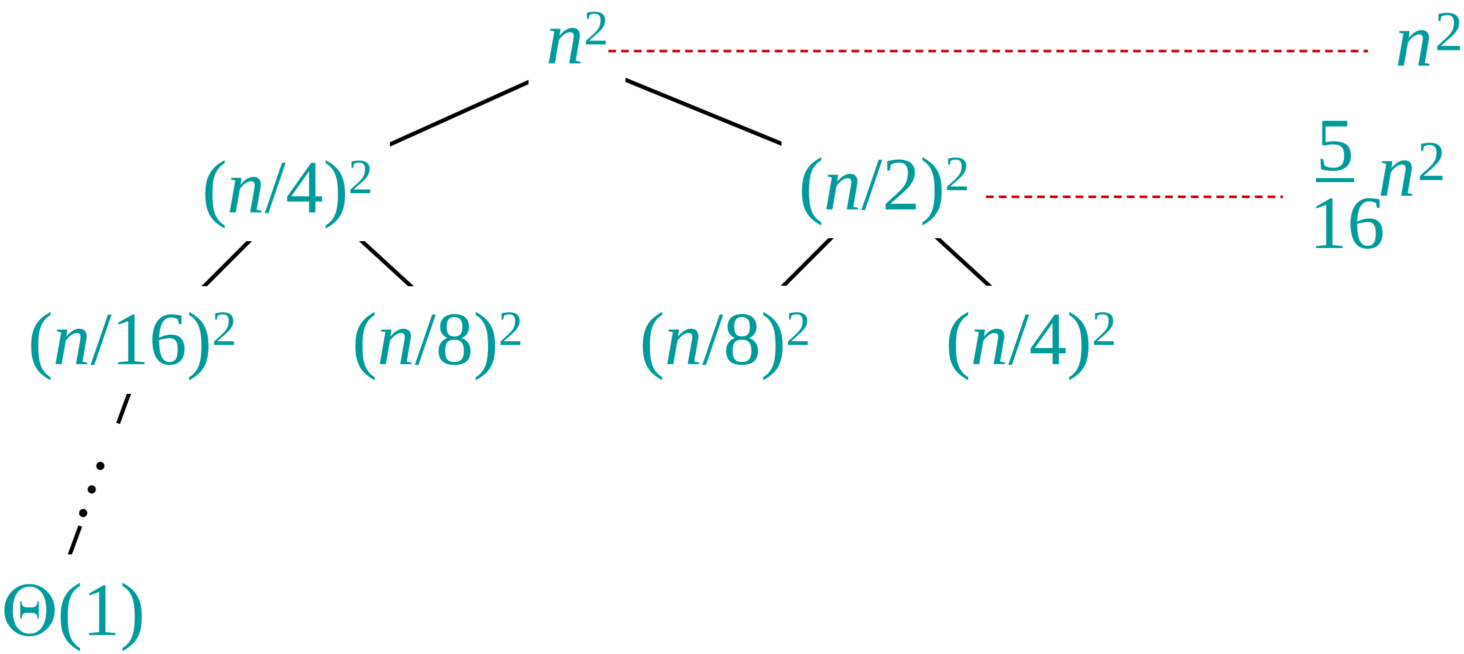
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

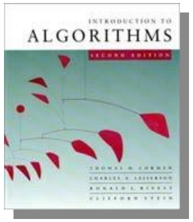




Recursion tree – Example 2

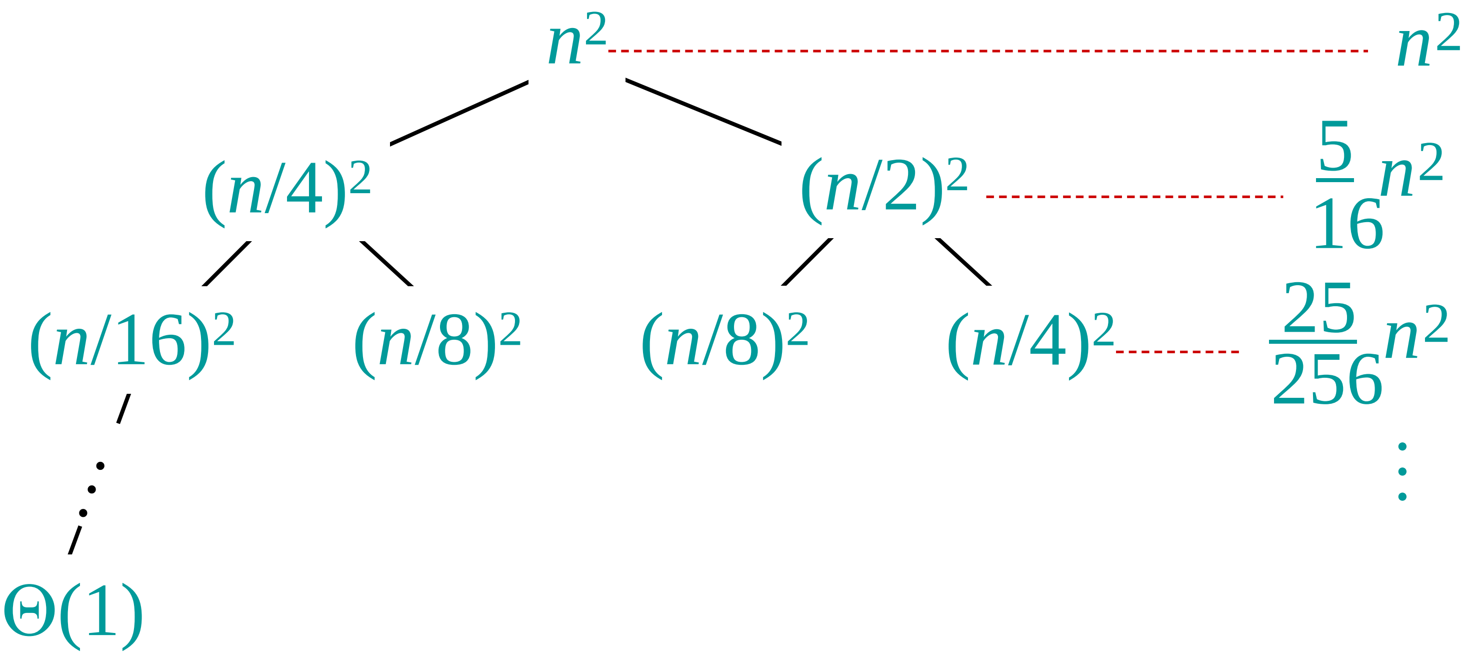
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

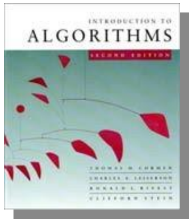




Recursion tree – Example 2

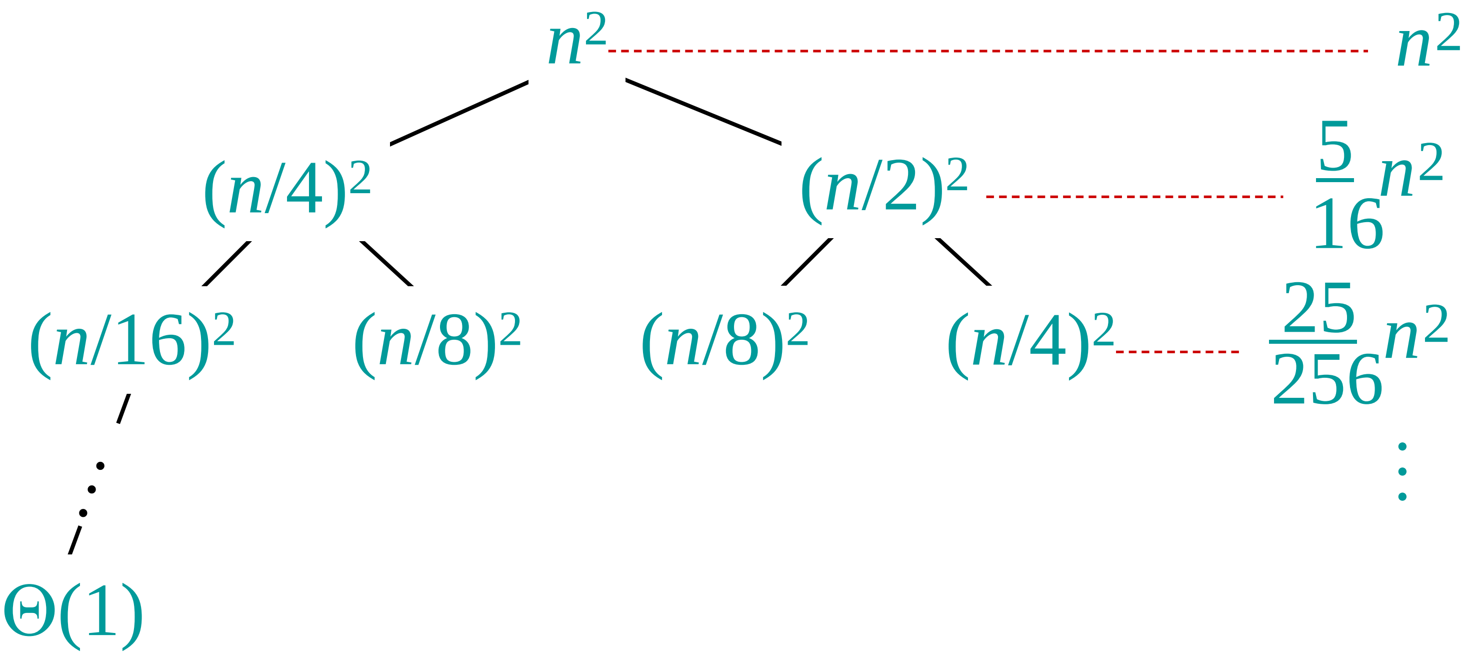
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

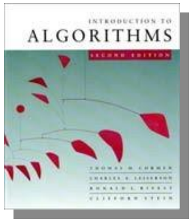




Recursion tree – Example 2

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





Recursion tree – Example 2

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

n^2 .... n^2

$(n/4)^2$ $(n/2)^2$ .... $\frac{5}{16}n^2$

$(n/16)^2$ $(n/8)^2$ $(n/8)^2$ $(n/4)^2$ .... $\frac{25}{256}n^2$

$\vdots$

$\Theta(1)$

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

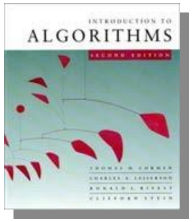
geometric series

Note: This computes a bound, not the exact cost.

$\therefore T(n) = O(n^2)$

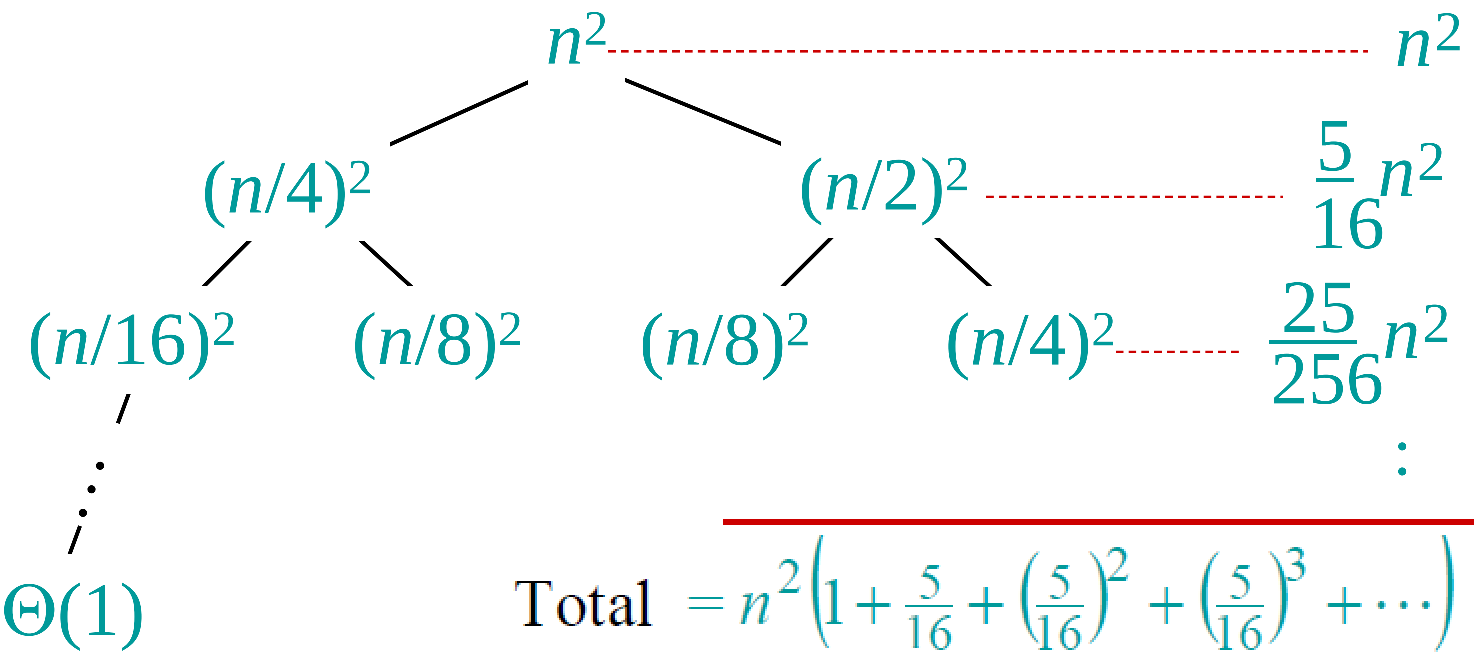
Also, note that $T(n) = \Omega(n^2)$

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



Recursion tree – Example 2

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



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

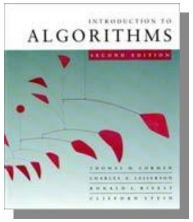
geometric series

$\therefore T(n) = O(n^2)$

Also, note that $T(n) = \Omega(n^2)$

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

Q. Why was the cost of the leaf nodes ignored?



Recursion tree – Example 2

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

$$n^2 \quad \cdots\cdots \quad n^2$$

$$(n/4)^2 \qquad (n/2)^2 \quad \cdots\cdots \quad \frac{5}{16}n^2$$

$$(n/16)^2 \qquad (n/8)^2 \qquad (n/8)^2 \qquad (n/4)^2 \quad \cdots\cdots \quad \frac{25}{256}n^2$$

$$\vdots$$

$$\Theta(1)$$

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

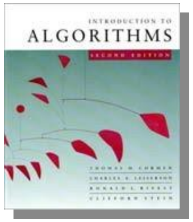
geometric series

$\therefore T(n) = O(n^2)$

Also, note that $T(n) = \Omega(n^2)$

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

Q. Why was the cost of the leaf nodes ignored? The number of leaf nodes is O(n).

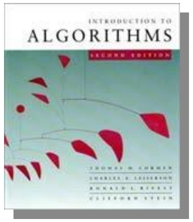


The master method

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.

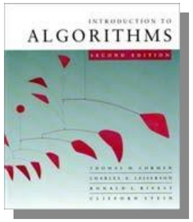


Three common cases

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

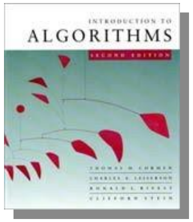
Compare $f(n)$ with $n^{\log_b a}$:

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



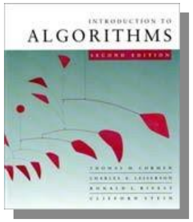
Three common cases (cont.)

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



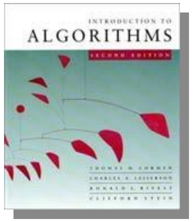
Examples

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 $\varepsilon = 1$.

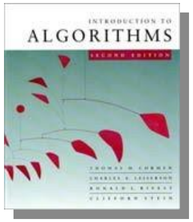
$\therefore\ T(n) = \Theta(n^2).$

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

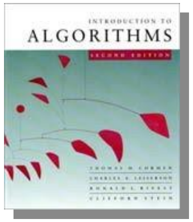
Ex. $T(n) = 4T(n/2) + n^3$

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

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

and $4(n/2)^3 \leq cn^3$ (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 $n^\varepsilon = \omega(\lg n)$.