Data Structures and Algorithm

Xiaoqing Zheng
zhengxq@fudan.edu.cn

Problem

x: *A G C A T G C*

y: *G C A G A C T*

Are we similar?

Problem

x: *A G C A T G C* **5/7**

y: *G C A G A C T*

z: *G C T A G A T* **6/7**

Are we similar?

What are algorithms?

- *A sequence of computational steps that transform the input into the output*

Sorting problem:

Input: A sequence of n numbers $<a_1, a_2, \ldots, a_n>$

Output: A permutation (reordering) $<a'_1, a'_2, \ldots, a'_n>$ such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Instance of sorting problem:

Input: <32, 45, 64, 28, 45, 58>

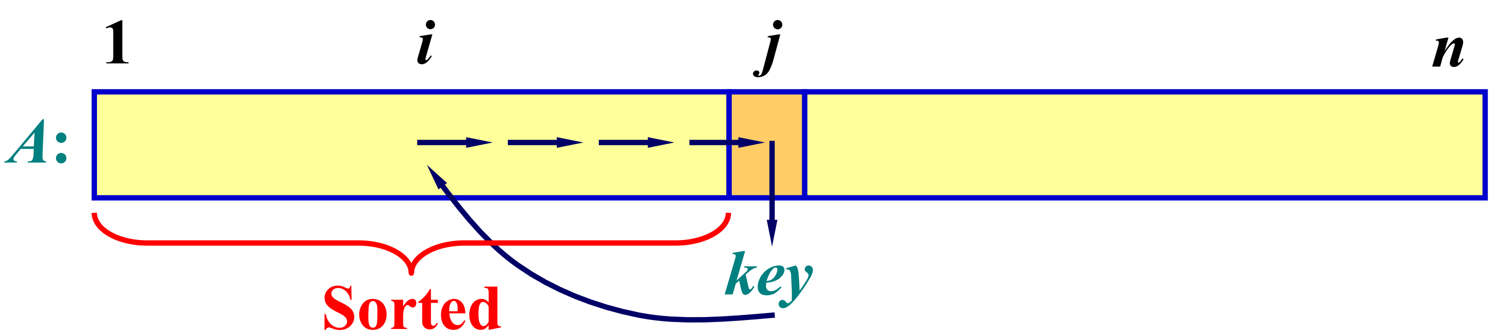
Output: <28, 32, 45, 45, 58, 64>

Example of insert 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*

Insertion sort

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
       // Insert A[j] into the sorted sequence A[1 .. j − 1]
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```



Analysis of insertion sort

INSERTION-SORT(A)	***cost***	***times***
for $j \leftarrow 2$ **to** $length[A]$	c_1	n
do $key \leftarrow A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Analysis of insertion sort: best and worst

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Best case:

- The best case occurs if the array is already sorted

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

***an* + *b* (linear function)**

Worst case:

- The worst case occurs if the array is in reverse sorted order

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

***an*^{2} + *bn* + c (quadratic function)**

Running time

- 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?*

  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"

Θ-notation

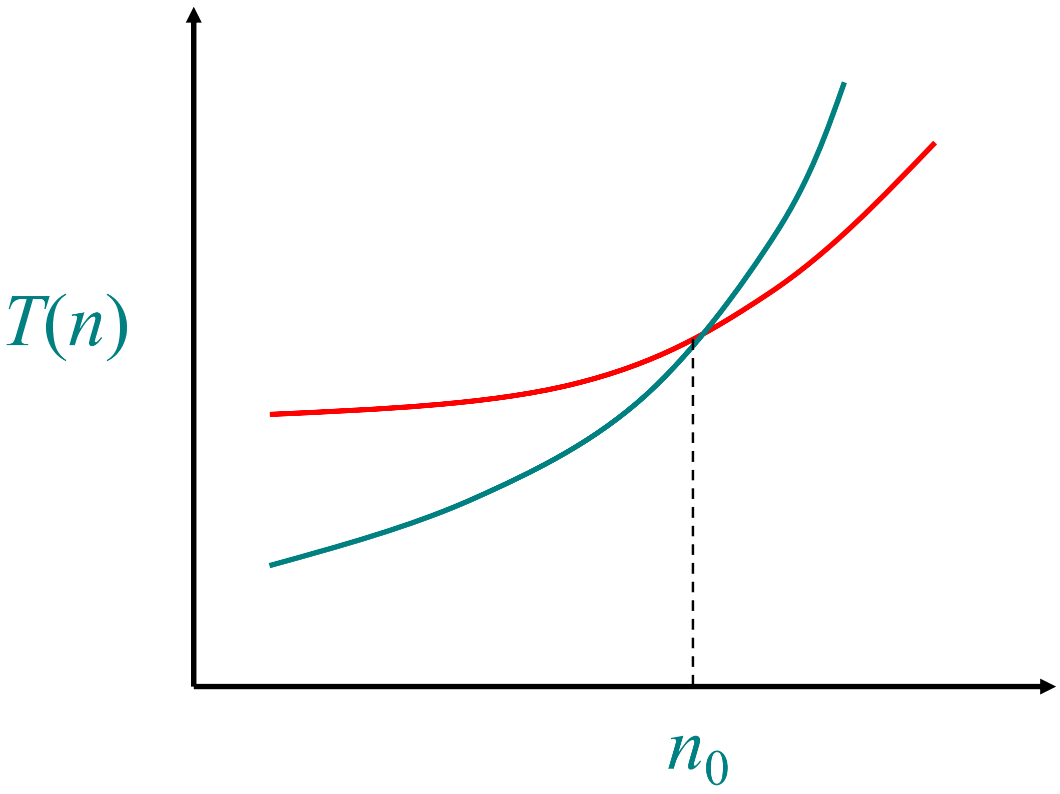
- ***Math:***

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

- ***Engineering:***
  - Drop low-order terms; ignore leading constants.
  - Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

Asymptotic performance

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



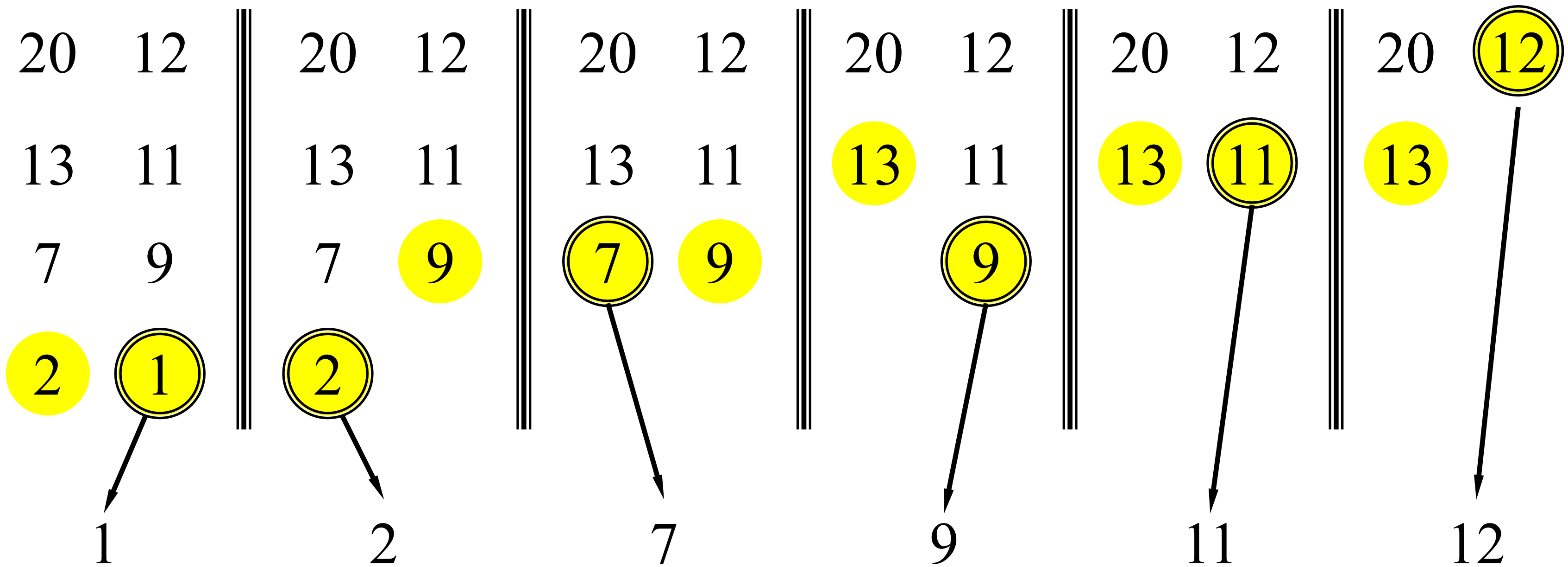
Merge sort

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

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

Key subroutine: *MERGE*

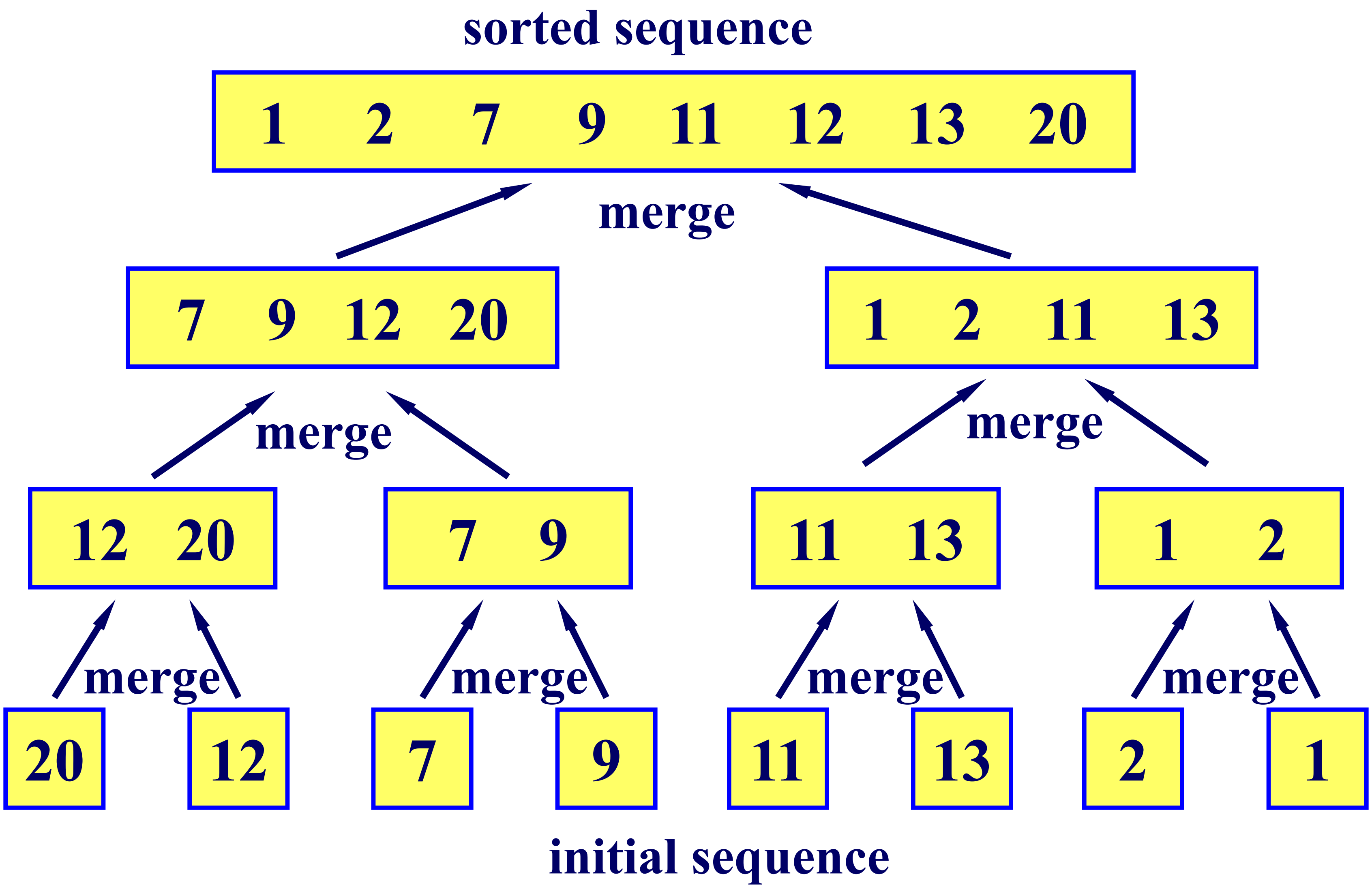
Merging two sorted arrays



$\boxed{1, 2, 7, 9, 11, 12, 13, 20}$

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

Operation of merge sort



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 $A[\lfloor n/2 \rfloor + 1 \,..\, n]$
$\Theta(n)$	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

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

Recurrence for merge sort

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

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

Recurrence for merge sort

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

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

$$T(n)$$

Recurrence for merge sort

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

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

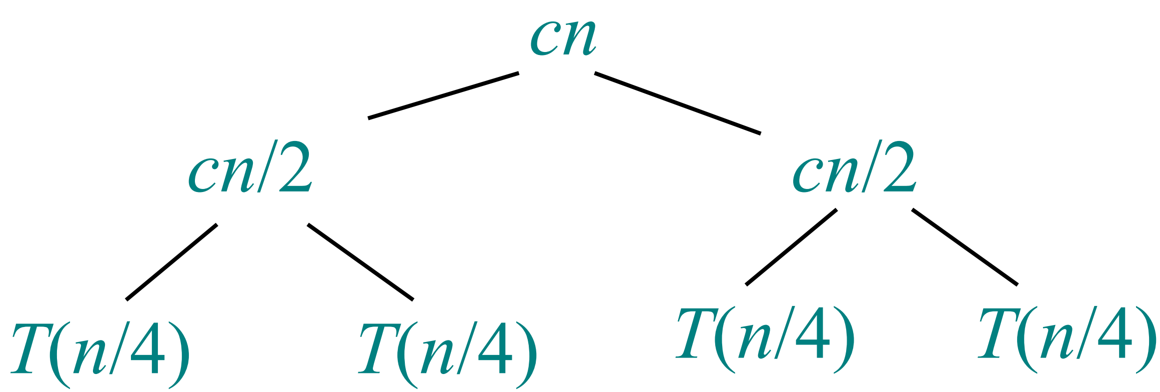
$$cn$$

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

Recurrence for merge sort

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

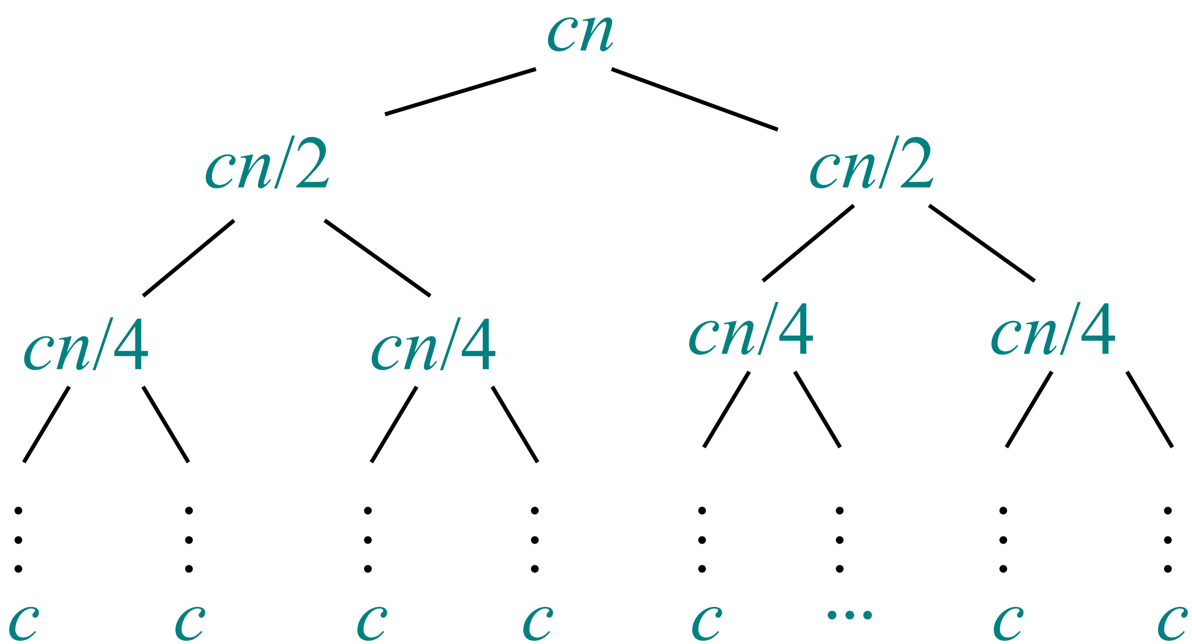
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recurrence for merge sort

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

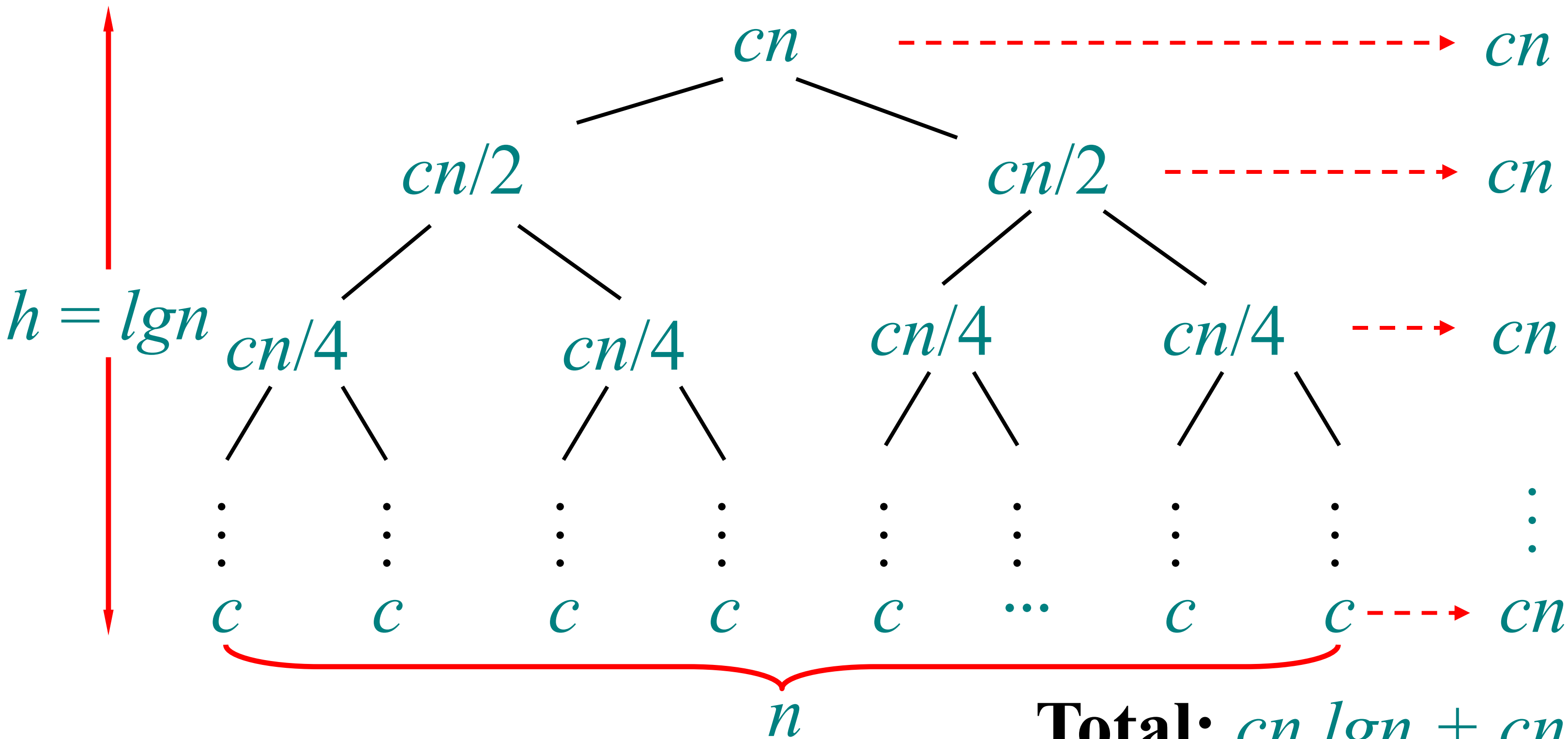
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recurrence for merge sort

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

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



Total: $cn\,lgn + cn$

Insertion sort and merge sort

Insertion sort: $c_1 n^2$

Computer *A* executes one billion instructions per second

$c_1 = 2$
Insertion sort: $2n^2$

***To sort one million numbers*:**

$$\frac{2 \cdot (10^6)^2 \text{ instructions}}{10^9 \text{ instructions/second}}$$

= 2000 seconds
(ten million, 2.3 day)

Merge sort: $c_2 n lg n$

Computer *B* executes ten million instructions per second

$c_2 = 50$
Merge sort: $50 n lg n$

***To sort one million numbers*:**

$$\frac{50 \cdot 10^6\, lg 10^6 \text{ instructions}}{10^7 \text{ instructions/second}}$$

= 100 seconds
(ten million, 20 minutes)

Analysis of algorithms

- *The theoretical study of computer-program performance and resource usage.*
- **What's more important than performance?**
  - Modularity
  - Correctness
  - Maintainability
  - Functionality
  - Robustness
  - User-friendliness
  - Programmer time
  - Simplicity
  - Extensibility
  - Reliability

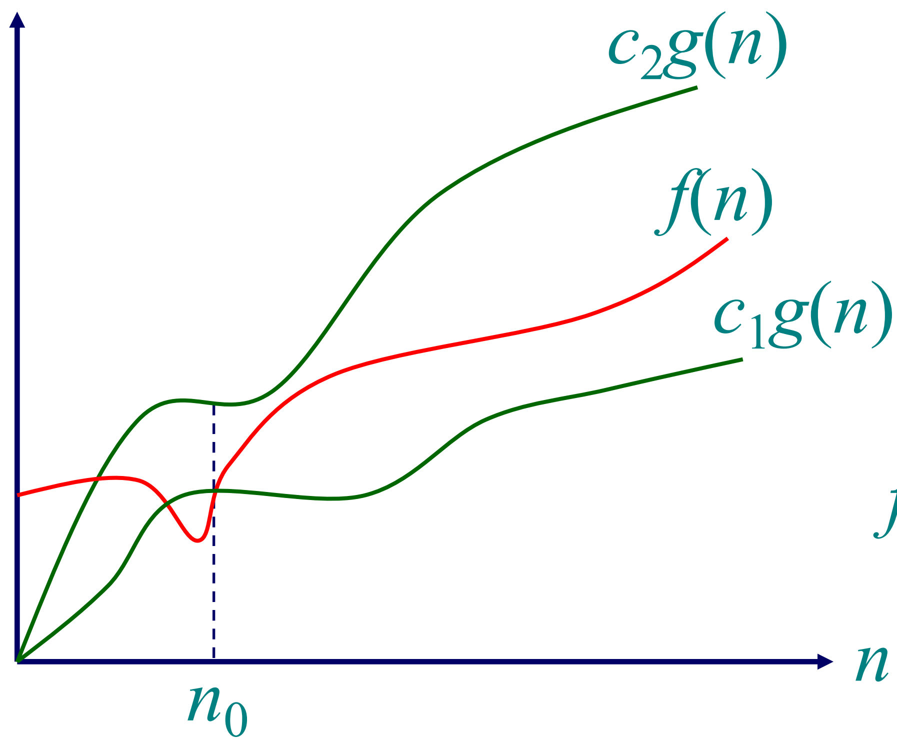
Comparison of running times

For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
lgn							
$n^{1/2}$							
n							
$nlgn$							
n^2							
n^3							
2^n							
$n!$							

Asymptotically tight bound

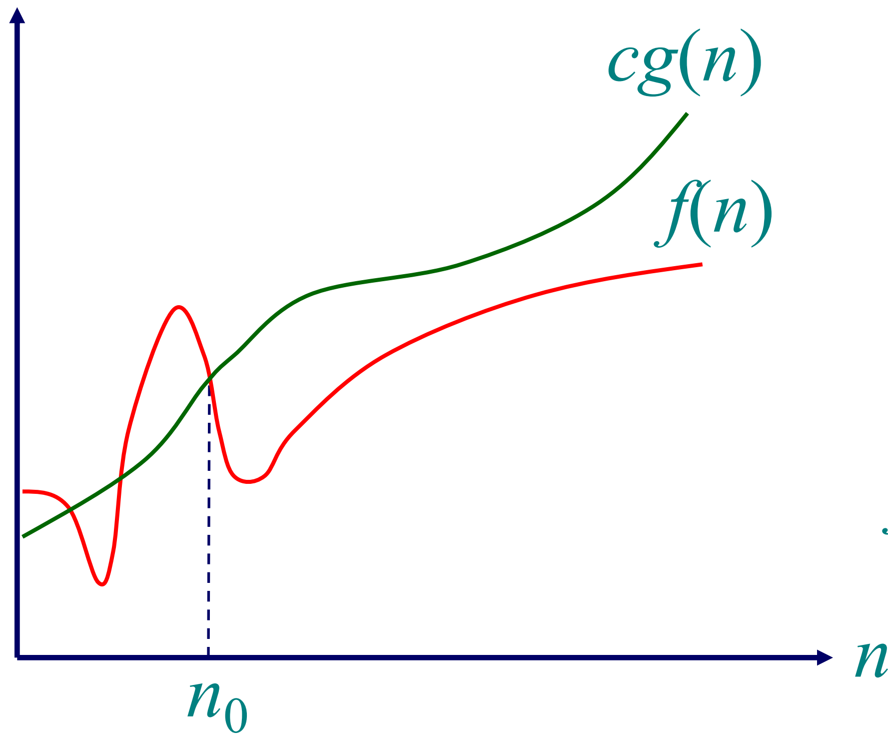
$\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$ $\}$



$f(n) = \Theta(g(n))$

Asymptotically upper bound

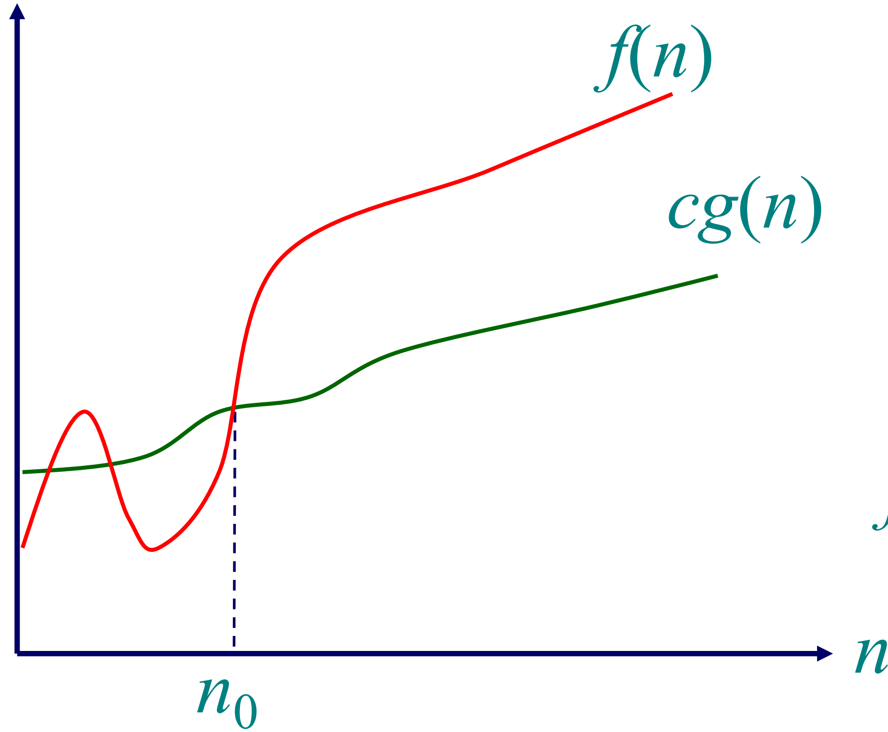
$O(g(n)) = \{ f(n) :$ there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$ $\}$



$f(n) = O(g(n))$

Asymptotically lower bound

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



$$f(n) = \Omega(g(n))$$

Asymptotic notations

An analogy between the asymptotic comparison of two functions f and g the comparison of two real numbers a and b:

$$\boxed{\begin{gathered} f(n) = O(g(n)) \approx a \leq b, \\ f(n) = \Omega(g(n)) \approx a \geq b, \\ f(n) = \Theta(g(n)) \approx a = b. \end{gathered}}$$

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

Recurrences

- **Substitution method**
- **Recursion-tree method**
- **Master method**

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 $O(n^3)$.
- Assume that $T(k) \leq ck^3$ for $k < n$.
- Prove $T(n) \leq cn^3$ by induction.

Example of substitution

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

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

residual

Example of substitution (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 $1 \le n < n_0$, we have "$\Theta(1)$" $\le cn^3$, if we pick c big enough.

This bound is not tight!

A tighter upper bound?

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

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

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

$= \cancel{O(n^2)}$ ***Wrong!*** We must prove the inductive hypothesis.

$= cn^2 - (-n)$ [*desired* $-$ *residual*]

$\le cn^2$ for no choice of $c > 0$. ***Lose!***

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 $k < n$.

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

Pick c_1 big enough to handle the initial conditions.

Substitution: changing variables

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + lgn$$

- Renaming $m = lgn$ yields

$$T(2^m) = 2T(2^{m/2}) + m$$

- Rename $S(m) = T(2^m)$ to produce the new recurrence

$$S(m) = 2S(m/2) + m$$

$$S(m) = O(m\ lgm)$$

- Changing back from $S(m)$ to $T(n)$

$$T(n) = T(2^m) = S(m) = O(m\ lgm) = O(lgn\ lglgn)$$

Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method promotes intuition.
- The recursion tree method is good for generating guesses for the substitution method.

Example of recursion tree

Solve $T(n) = 3T(\lfloor n/4 \rfloor) + n^2$

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

Example of recursion tree

Solve $T(n) = 3T(\lfloor n/4 \rfloor) + n^2$

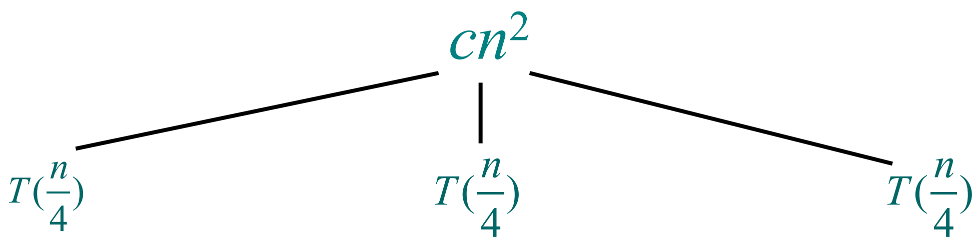
$$T(n) = 3T(n/4) + n^2$$

$$T(n)$$

Example of recursion tree

Solve $T(n) = 3T(\lfloor n/4 \rfloor) + n^2$

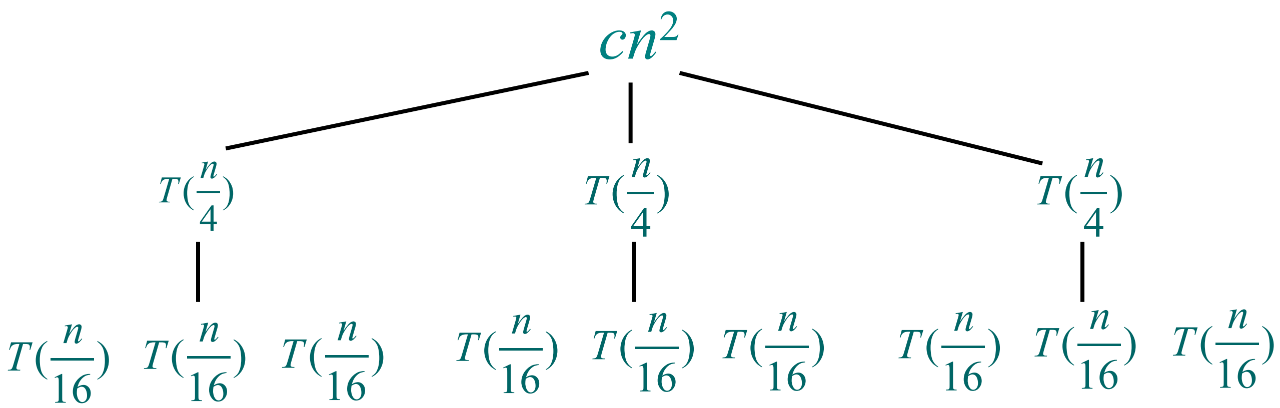
$$T(n) = 3T(n/4) + n^2$$



Example of recursion tree

Solve $T(n) = 3T(\lfloor n/4 \rfloor) + n^2$

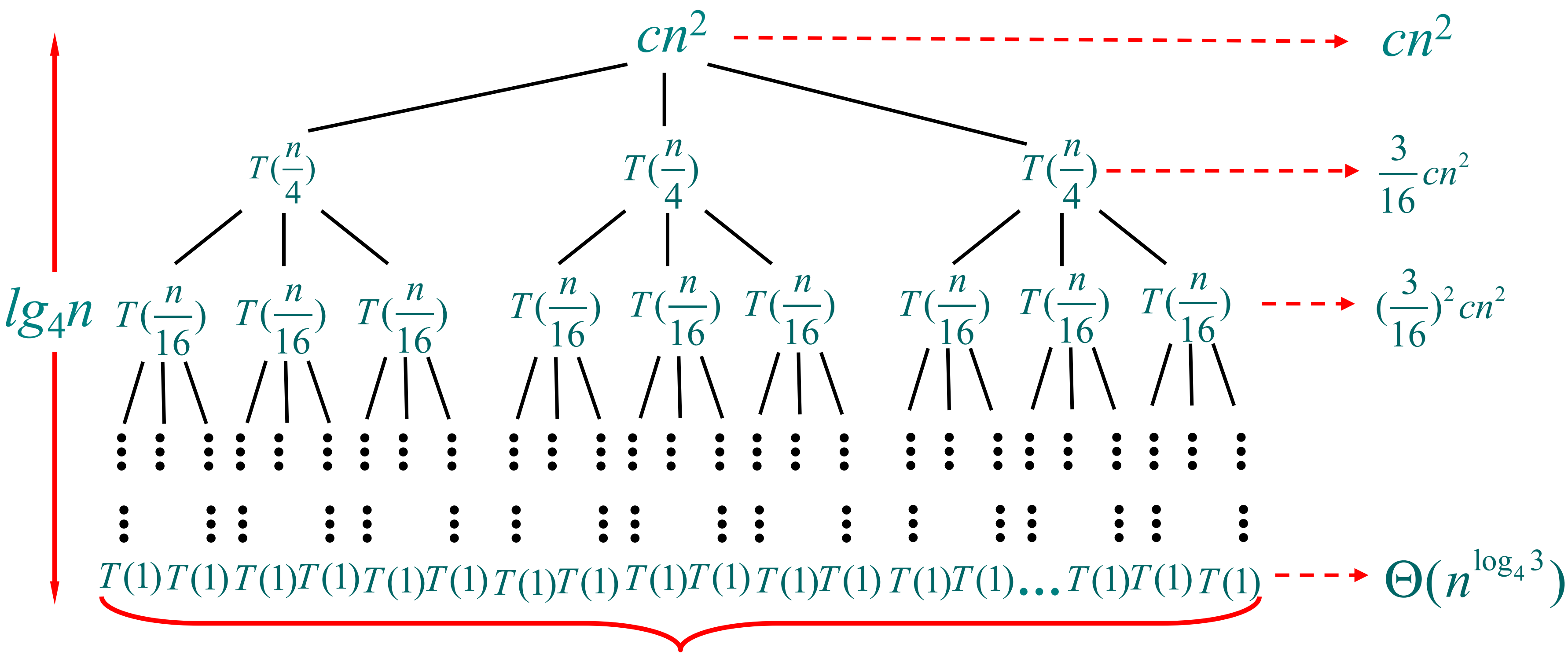
$$T(n) = 3T(n/4) + n^2$$



Example of recursion tree

Solve $T(n) = 3T(\lfloor n/4 \rfloor) + n^2$

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



Total: $O(n^2)$

Cost for the entire tree

$$T(n) = cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \cdots + (\frac{3}{16})^{\log_4 n-1} cn^2 + \Theta(n^{\log_4 3})$$

$$= \sum_{i=0}^{\log_4 n-1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3})$$

$$< \sum_{i=0}^{\infty} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2)$$

Substitution method to verify

$$
\begin{aligned}
T(n) &= 3T(\lfloor n/4 \rfloor) + \Theta(n^2) \\
&\le 3T(\lfloor n/4 \rfloor) + cn^2 \\
&\le 3d\lfloor n/4 \rfloor^2 + cn^2 \\
&\le 3d(n/4)^2 + cn^2 \\
&= \frac{3}{16}dn^2 + cn^2 \\
&\le dn^2
\end{aligned}
$$

Where the last step holds as long as $d \ge (16/13)c$

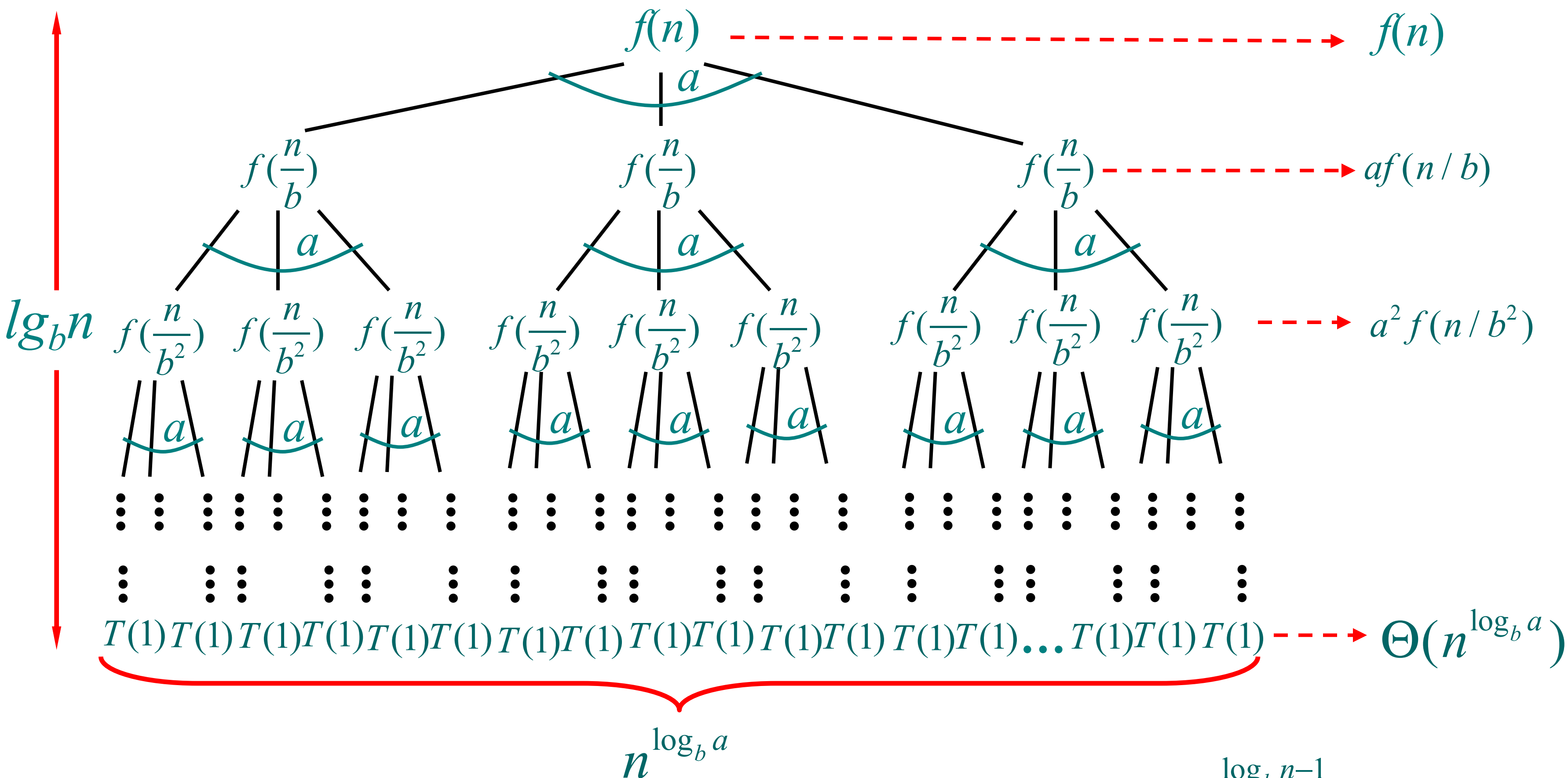
The master method

The master method applies to recurrences of the form

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

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

Idea of master theorem



Total: $\Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$

Three common cases

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

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 $n^{\log_b a - \varepsilon}$ (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 $n^{\log_b a - \varepsilon}$ (by an n^{ε} factor).

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

2. $f(n) = O(n^{\log_b a} \lg n)$.

   $f(n)$ and $n^{\log_b a}$ grow at similar rates.

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

Three common cases (continued)

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 $n^{\log_b a+\varepsilon}$ (by an n^{ε} factor).

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

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

Proof

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

$$\leq \sum_{j=0}^{\log_b n - 1} c^j f(n) \quad \text{if } af(n/b) \leq cf(n)$$

$$\leq f(n) \sum_{j=0}^{\log_b n - 1} c^j$$

$$= f(n)\left(\frac{1}{1-c}\right)$$

$$= O(f(n))$$

Examples

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

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

Examples

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

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

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

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

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: $f(n) = \Theta(n^2)$

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

Examples

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

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

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

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

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

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

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

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

Master method does not apply.

Any question?

Xiaoqing Zheng
Fudan University