CME 2001
Data Structures and Algorithms

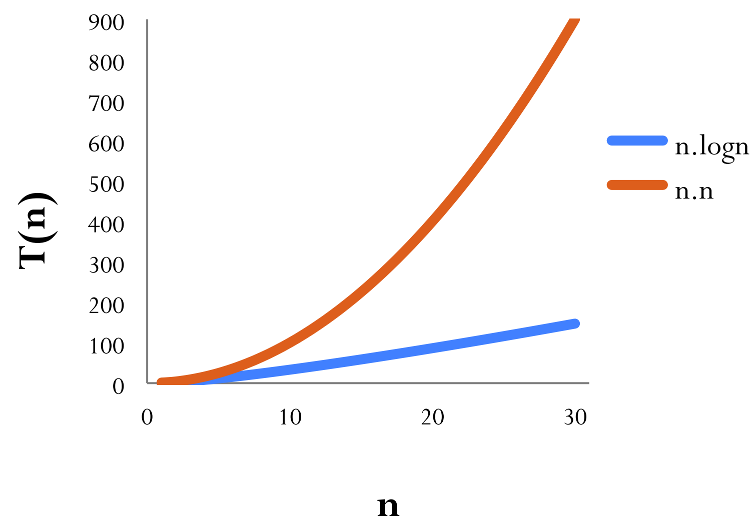
Zerrin Işık
zerrin@cs.deu.edu.tr

Growth of Functions

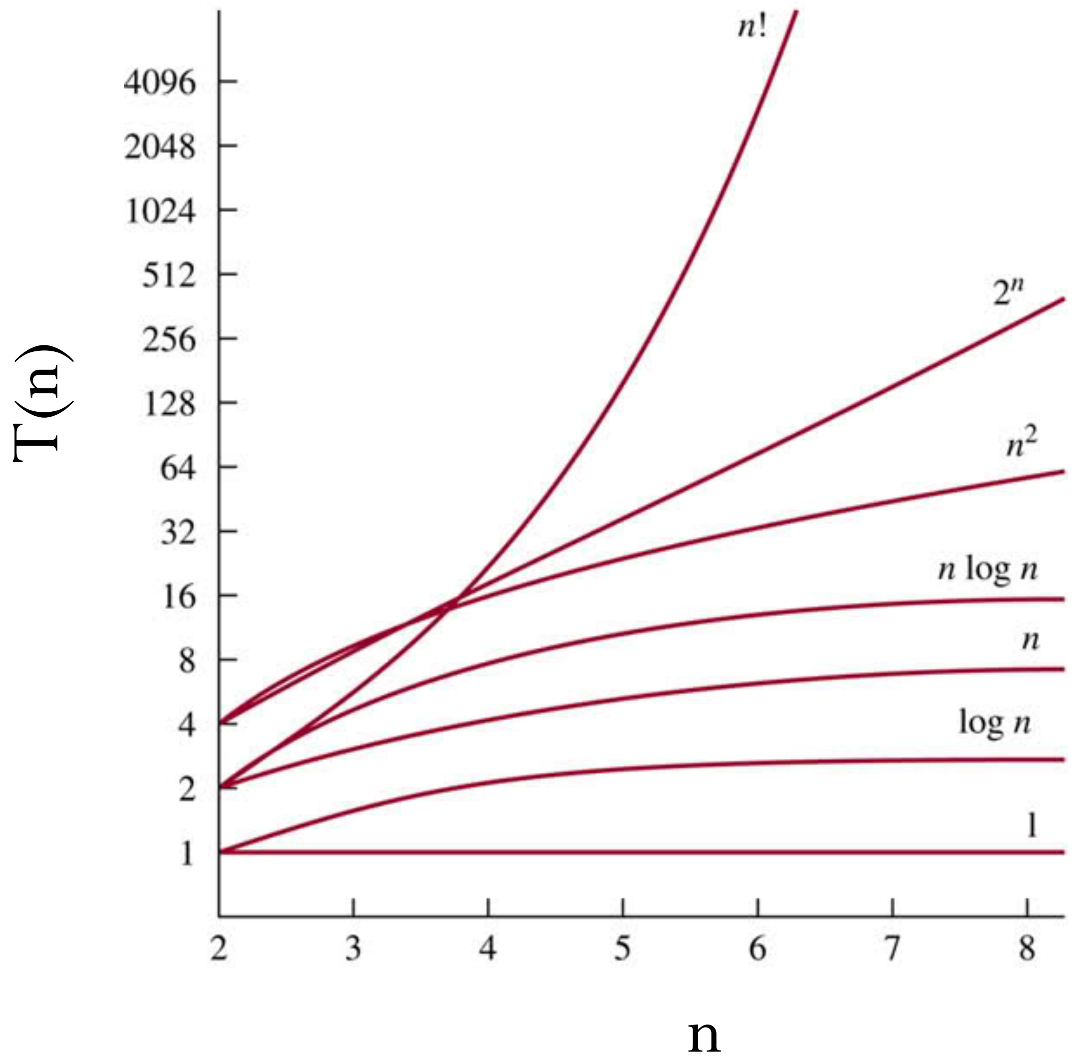
Growth Rate of Functions

- The order of growth of running time gives algorithm's efficiency and helps us to compare performance of alternative algorithms
  - e.g. for a large input size of *n*, merge sort (Θ(n.logn)) beats insertion Sort ($\Theta(n^2)$)

- We compare the efficiency of algorithms by comparing their growth rate.



Comparison of Growth-Rate Functions



Asymptotic Notation

- If an algorithm *A* requires time proportional to $f(n)$, it is order $\boldsymbol{f(n)}$, and it is denoted as $\boldsymbol{O(f(n))}$

- $\boldsymbol{f(n)}$ is called the **growth-rate function** of the algorithm *A*.

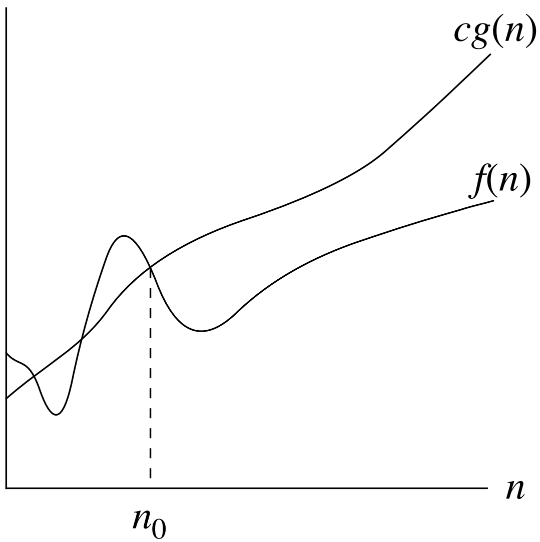
Big-O Notation

Given two growth-rate functions f(n) and g(n):

$$\mathbf{f(n) = O(g(n))}$$

if there exist positive constants c and n_0 such that $f(n) \leq c.g(n)$ for all $n \geq n_0$.

g(n) is an ***asymptotic upper bound*** for f(n).



Big-O Notation

$2n^2 = O(n^3)$ because $0 \le 2n^2 \le n^3$ where $c=1$, $n_0=2$

(! one way equality, not symmetric)

Examples of functions in $O(n^2)$:

n^2
$n^2 + n$
$n^2 + 1000n$
$1000n^2 + 1000n$
Also,
n
$n/1000$
$n^{1.99999}$
$n^2/ \lg\lg\lg n$

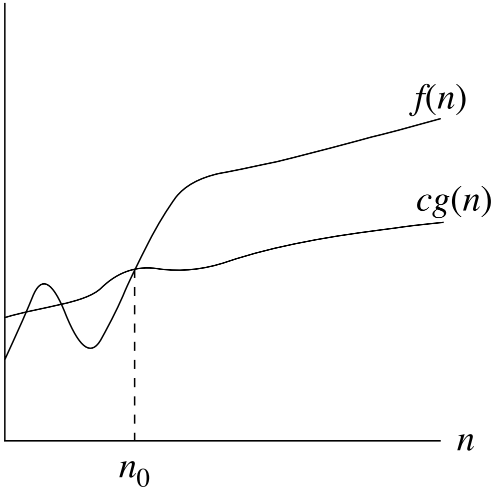
Ω (Omega) Notation

Given two growth-rate functions f(n) and g(n):

$$\mathbf{f(n) = \Omega\,(g(n))}$$

if there exist positive constants c and n_0 such that $c.g(n) \leq f(n)$ for all $n \geq n_0$.

g(n) is an ***asymptotic lower bound*** for f(n).



Ω - Notation

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

Examples of functions in $\Omega(n^2)$:

n^2
$n^2 + n$
$n^2 - n$
$1000n^2 + 1000n$
$1000n^2 - 1000n$
Also,
n^3
$n^{2.00001}$
$n^2 \lg\lg\lg n$
2^{2^n}

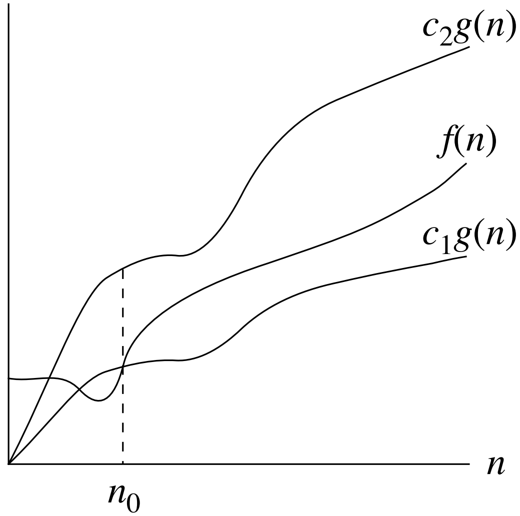
Θ-Notation

Given two growth-rate functions f(n) and g(n):

$$\mathbf{f(n) = \Theta(g(n))}$$

if there exist positive constants c_1, c_2 and n_0 such that $c_1.g(n) \leq f(n) \leq c_2.g(n)$ for all $n \geq n_0$.

g(n) is an ***asymptotic tight bound*** for f(n).



Θ-Notation

E.g. $n^2/2 - 2n = \Theta(n^2)$ where $c_1 = ¼$, $c_2 = ½$, $n_0 = 8n$

Theorem:

$f(n) = \Theta(g(n))$ iff $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$

* Leading constants and low-order terms do not matter

o-Notation and w-notation

O-notation and Ω-notation represent $\leq$ and $\geq$

o-notation and w-notation represent $<$ and $>$

f(n) = o(g(n)) for all constants c>0, there exits n_0 >0 such that $0 \leq f(n) < c.g(n)$ for all $n \geq n_0$.

E.g.

$$n^{1.9999} = o(n^2)$$
$$n^2/\lg n = o(n^2)$$
$$n^2 \neq o(n^2) \text{ (just like } 2 \not< 2)$$
$$n^2/1000 \neq o(n^2)$$

Note: Inequality must hold for all c (where c>0)

w-notation

f(n) = w(g(n)) for all constants c>0, there exits n_0 >0 such that $0 \leq c.g(n) < f(n)$ for all $n \geq n_0$.

E.g.

$$n^{2.0001} = \omega(n^2)$$
$$n^2 \lg n = \omega(n^2)$$
$$n^2 \neq \omega(n^2)$$

Substitution Method to Solve Recurrence

Recurrence relations represent the running times of divide-and-conquer algorithms.

To solve recurrence relations :

1. Guess the form of the solution.
2. Use mathematical induction to find the constants and show this solution works.

Substitution Method ...

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

1. Guess: T(n) = n lgn + n

2. Induction:

   Basis: n=1 => n lgn + n = 1 = T(n) ✓

   Inductive Step: Our hypothesis is that

   T(k) = k lgk + k for all k <n

   We will use this inductive hypothesis for T(n/2)

➔

Substitution Method …

Assume T(k) = k lgk + k, then

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n \\
&= 2\left(\frac{n}{2}\lg\frac{n}{2} + \frac{n}{2}\right) + n \qquad \text{(by inductive hypothesis)} \\
&= n\lg\frac{n}{2} + n + n \\
&= n(\lg n - \lg 2) + n + n \\
&= n\lg n - n + n + n \\
&= n\lg n + n\ . \qquad \blacksquare
\end{aligned}
$$

Substitution Method …

- Show the upper (O) and lower (Ω) bounds separately
- If necessary use different constants for each bound

Substitution Method ...

E.g. T(n) = 2T(n/2) + Θ (n).

Upper bound :

Aim is to show T(n)= 2T(n/2)+ O(n), then we should write

T(n) $\leq$ 2T(n/2) + cn for some positive constant c.

Guess: T(n) $\leq$ d n lgn for some positive constant d.

Substitution:

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + cn \\
&= 2\left(d\frac{n}{2}\lg\frac{n}{2}\right) + cn \\
&= dn\lg\frac{n}{2} + cn \\
&= dn\lg n - dn + cn \\
&\leq dn\lg n \qquad \text{if } -dn + cn \leq 0, \\
&\qquad\qquad\qquad\qquad\quad d \geq c
\end{aligned}
$$

Therefore, T(n) = O(n lgn).

Substitution Method …

Lower bound :

Write $T(n) \geq 2T(n/2) + cn$ for some positive constant c.

Guess: $T(n) \geq d\, n \lg n$ for some positive constant d.

Substitution:

$$\begin{aligned}
T(n) &\geq 2T(n/2) + cn \\
&= 2\left(d\frac{n}{2}\lg\frac{n}{2}\right) + cn \\
&= dn\lg\frac{n}{2} + cn \\
&= dn\lg n - dn + cn \\
&\geq dn\lg n \qquad \text{if } -dn + cn \geq 0, \\
&\qquad\qquad\qquad\qquad d \leq c
\end{aligned}$$

Therefore, $T(n) = \Omega(n \lg n)$.

Substitution Method ...

E.g. $T(n) = 8T(n/2) + \Theta(n^2)$.

Upper bound :

$T(n) \leq 8T(n/2) + cn^2$

Guess: $T(n) \leq dn^3$

Substitution:
$$
\begin{aligned}
T(n) &\leq 8d(n/2)^3 + cn^2 \\
&= 8d(n^3/8) + cn^2 \\
&= dn^3 + cn^2 \\
&\not\leq dn^3
\end{aligned}
$$

does not work

Solution: Subtract off a lower-order term.

Substitution Method …

Guess: $T(n) \leq dn^3 - d'n^2$

Substitution:
$$
\begin{aligned}
T(n) &\leq 8(d(n/2)^3 - d'(n/2)^2) + cn^2 \\
&= 8d(n^3/8) - 8d'(n^2/4) + cn^2 \\
&= dn^3 - 2d'n^2 + cn^2 \\
&= dn^3 - d'n^2 - d'n^2 + cn^2 \\
&\leq dn^3 - d'n^2 \qquad \text{if } -d'n^2 + cn^2 \leq 0, \\
&\qquad\qquad\qquad\qquad\quad d' \geq c
\end{aligned}
$$

It worked now. ■

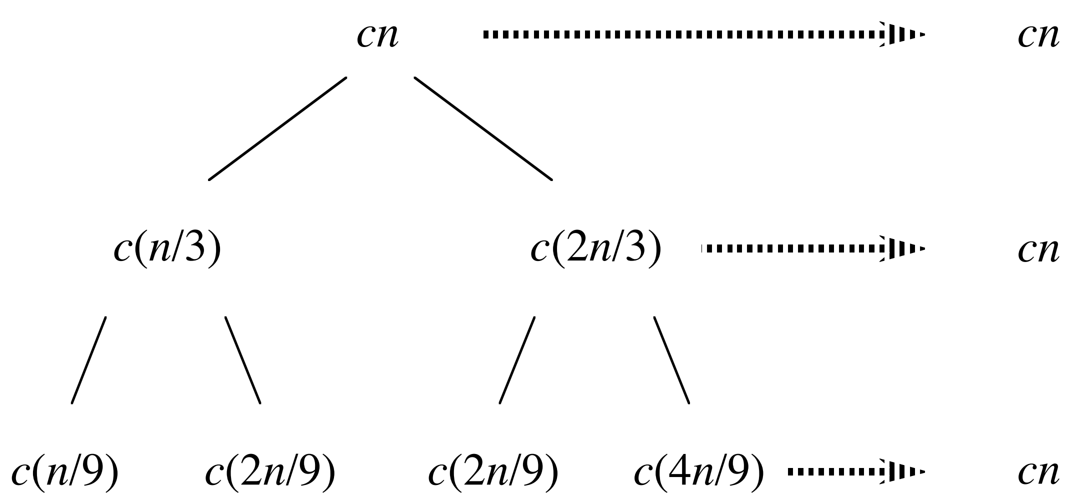
Recursion Tree to Solve Recurrence

- Useful to generate better guesses for substitution method

E.g. $T(n) = T(n/3) + T(2n/3) + \Theta(n)$

Height : $\lg_{3/2} n$

Each level $\leq$ cn



Recursion Tree ...

Upper bound rewrite as $T(n) \leq T(n/3) + T(2n/3) + cn$

Guess: $T(n) \leq dn \lg n$ for some positive constant d

Substitition:
$$\begin{aligned}
T(n) &\leq T(n/3) + T(2n/3) + cn \\
&\leq d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\
&= (d(n/3)\lg n - d(n/3)\lg 3) \\
&\qquad + (d(2n/3)\lg n - d(2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg 3 - (2n/3)\lg 2) + cn \\
&= dn\lg n - dn(\lg 3 - 2/3) + cn \\
&\leq dn\lg n \qquad \text{if } -dn(\lg 3 - 2/3) + cn \leq 0\,, \\
&\qquad\qquad d \geq \frac{c}{\lg 3 - 2/3}\,.
\end{aligned}$$

Therefore, $T(n) = O(n \lg n)$.

Master Method to Solve Recurrence

Useful to solve recurrences of the form:

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

$$\text{where } a \geq 1, b > 1, \text{and } f(n) > 0.$$

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

\# of leaves

- **<u>Case 1:</u>** $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.

  * $f(n)$ is polynomially smaller than $n^{\log_b a}$

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

(* cost is dominated by leaves.)

Master Method to Solve Recurrence

- **Case 2:** $f(n) = \Theta(n^{\log_b a} \lg^k n)$, where $k \geq 0$.

  $*$ $f(n)$ is within a polylog factor of $n^{\log_b a}$, but not smaller.

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

  (* cost is $n^{\log_b a} \lg^k n$ at each level, and there are $\Theta(\lg n)$ levels.)

- **Case 3:** $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$ and $f(n)$ satisfies the regularity condition $a f(n/b) \leq c f(n)$ for some constant $c<1$.

  $*$ $f(n)$ is polynomially greater than $n^{\log_b a}$.

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

  (* cost is dominated by root.)

Examples for Master Method

$T(n) = 5T(n/2) + \Theta(n^2)$
$n^{\log_2 5}$ vs. n^2
Since $\log_2 5 - \epsilon = 2$ for some constant $\epsilon > 0$ ✓
use Case 1 $\Rightarrow T(n) = \Theta(n^{\lg 5})$

$T(n) = 4T(n/2) + n^2$
$n^{\log_2 4}$ vs. n^2
Since $n^2 = n^2$ (for k=0) ✓
use Case 2 => $T(n) = \Theta(n^2 \lg n)$

Examples for Master Method …

$T(n) = 5T(n/2) + \Theta(n^3)$
$n^{\log_2 5}$ vs. n^3
Now $\lg 5 + \epsilon = 3$ for some constant $\epsilon > 0$

$af(n/b) = 5(n/2)^3 = 5n^3/8 \le cn^3$ for $c = 5/8 < 1$
Use Case 3 $\Rightarrow T(n) = \Theta(n^3)$

$T(n) = 27T(n/3) + \Theta(n^3/\lg n)$
$n^{\log_3 27} = \boxed{n^3 \text{ vs. } n^3/\lg n} = n^3 \lg^{-1} n \neq \Theta(n^3 \lg^k n)$ for any $k \ge 0$.
Cannot use the master method.

Next Week Topics

- Elementary Data Structures (Chapter 10)
- Hash Tables (Chapter 11)