Winter 2016
COMP-250: Introduction to Computer Science

Lecture 12, February 18, 2016

Master Theorem
(CLRS 4.3)

Master Theorem

Used for many divide-and-conquer recurrences

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

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

a = (constant) number of sub-instances,
b = (constant) size ration of sub-instances,
f(n) = time used for dividing and recombining.

Master Theorem

Used for many divide-and-conquer recurrences

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

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

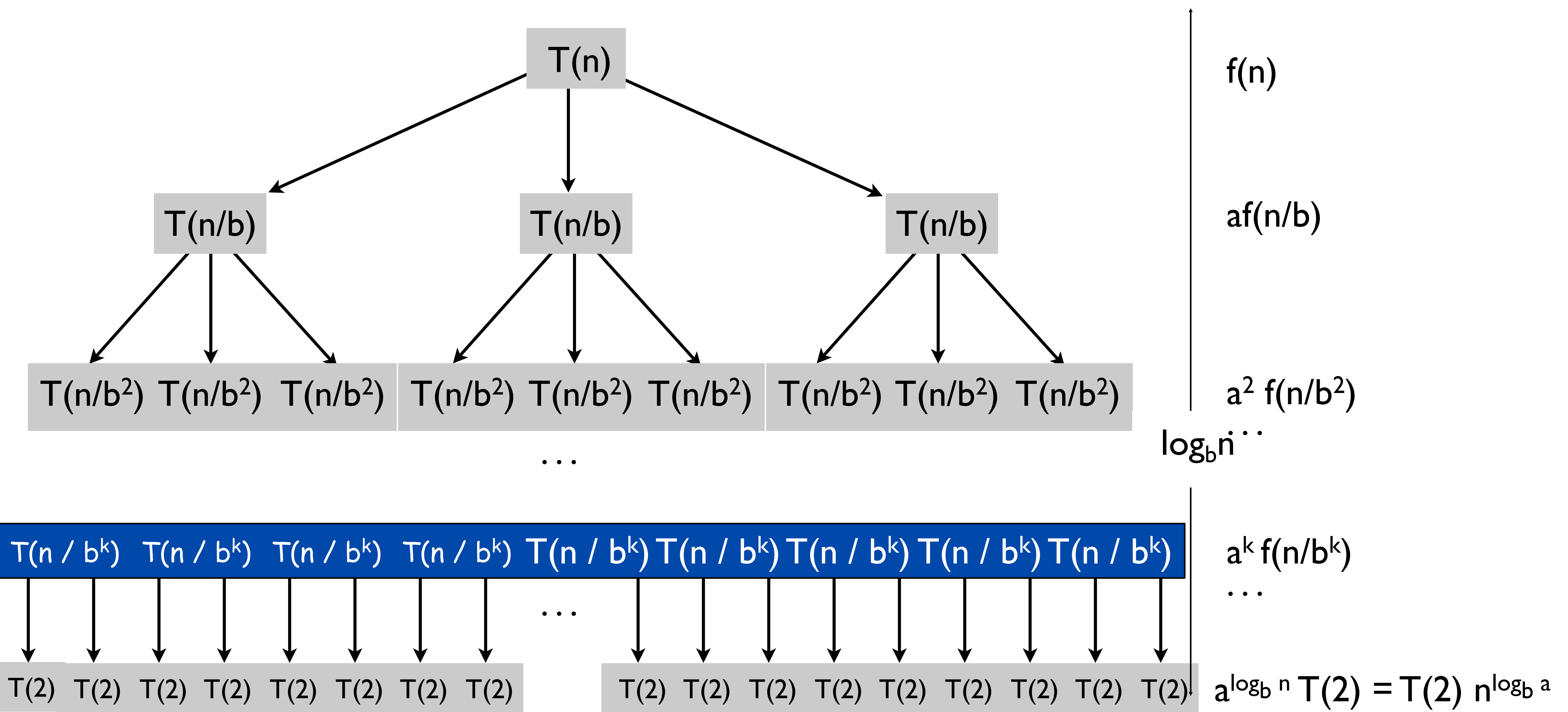
a = (constant) number of sub-instances,
b = (constant) size ration of sub-instances,
f(n) = time used for dividing and recombining.

Based on the ***master theorem*** (Theorem 4.1).

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

Proof by recursion tree

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



Master Theorem

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

Case 1: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a})$

Case 2: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

Case 3: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

Solution: $T(n)$ **is** $\Theta(f(n))$

Master Theorem

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

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

<u>Case 1</u>: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a})$

(Intuitively: cost is dominated by leaves.)

Master Theorem

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

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

Case 1: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

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

Solution: $T(n)$ **is** $\Theta(n^{\log_b a})$

(Intuitively: cost is dominated by leaves.)

Master Theorem

Case 1: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a})$

Master Theorem

<u>Case 1</u>: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a})$

$$T(n) = 5T(n/2) + \Theta(n^2)$$

Master Theorem

<u>Case 1</u>: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a})$

$$T(n) = 5T(n/2) + \Theta(n^2)$$

Master Theorem

Case 1: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a})$

$$T(n) = 5T(n/2) + \Theta(n^2)$$

Compare $n^{\log_2 5}$ vs. n^2.

Master Theorem

<u>Case 1</u>: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a})$

$$T(n) = 5T(n/2) + \Theta(n^2)$$

Compare $n^{\log_2 5}$ vs. n^2.

Master Theorem

Case 1: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a})$

$$T(n) = 5T(n/2) + \Theta(n^2)$$

Compare $n^{\log_2 5}$ vs. n^2.

Since $2 < \log_2 5$ use **Case 1**

Master Theorem

<u>Case 1</u>: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a})$

$$T(n) = 5T(n/2) + \Theta(n^2)$$

Compare $n^{\log_2 5}$ vs. n^2.

Since $2 < \log_2 5$ use **<u>Case 1</u>**

Master Theorem

<u>Case 1</u>: $f(n)$ **is** $O(n^L)$ for some constant $L < \log_b a$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a})$

$T(n) = 5T(n/2) + \Theta(n^2)$

Compare $n^{\log_2 5}$ vs. n^2.

Since $2 < \log_2 5$ use **<u>Case 1</u>**

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_2 5})$

Master Theorem

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

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

Simple Case 2: $f(n)$ **is** $\Theta(n^{\log_b a})$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log n)$

Master Theorem

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

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

Case 2: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

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

Master Theorem

<u>Case 2</u>: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

Master Theorem

<u>Case 2</u>: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3 \log n)$$

Master Theorem

<u>Case 2</u>: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3 \log n)$$

Master Theorem

Case 2: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3 \log n)$$

Compare $n^{\log_3 27}$ vs. n^3.

Master Theorem

Case 2: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3 \log n)$$

Compare $n^{\log_3 27}$ vs. n^3.

Master Theorem

<u>Case 2</u>: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3 \log n)$$

Compare $n^{\log_3 27}$ vs. n^3.

Since $3 = \log_3 27$ use **<u>Case 2</u>**

Master Theorem

Case 2: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3 \log n)$$

Compare $n^{\log_3 27}$ vs. n^3.

Since $3 = \log_3 27$ use **Case 2**

Master Theorem

<u>Case 2</u>: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3 \log n)$$

Compare $n^{\log_3 27}$ vs. n^3.

Since $3 = \log_3 27$ use **<u>Case 2</u>**

<u>Solution:</u> $T(n)$ **is** $\Theta(n^3 \log^2 n)$

Master Theorem

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

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

<u>Case 3</u>: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

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

<u>Solution:</u> $T(n)$ **is** $\Theta(f(n))$

(Intuitively: cost is dominated by root.)

Master Theorem

<u>Case 3</u>: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

<u>Solution:</u> $T(n)$ **is** $\Theta(f(n))$

Master Theorem

<u>Case 3</u>: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

<u>Solution:</u> $T(n)$ **is** $\Theta(f(n))$

What's with the Case 3 regularity condition?

- Generally not a problem.
- It always holds whenever $f(n) = n^k$ and $f(n)$ **is** $\Omega(n^{\log_b a+\epsilon})$ for constant $\epsilon > 0$.

Master Theorem

<u>Case 3</u>: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

<u>Solution:</u> $T(n)$ **is** $\Theta(f(n))$

Master Theorem

Case 3: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$
and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

Solution: $T(n)$ **is** $\Theta(f(n))$

$$T(n) = 5T(n/2) + \Theta(n^3)$$

Master Theorem

<u>Case 3</u>: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

<u>Solution:</u> $T(n)$ **is** $\Theta(f(n))$

$$T(n) = 5T(n/2) + \Theta(n^3)$$

Master Theorem

<u>Case 3</u>: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

<u>Solution:</u> $T(n)$ **is** $\Theta(f(n))$

$$T(n) = 5T(n/2) + \Theta(n^3)$$

Compare $n^{\log_2 5}$ vs. n^3.

Master Theorem

<u>Case 3</u>: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$

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

<u>Solution:</u> $T(n)$ **is** $\Theta(f(n))$

$$T(n) = 5T(n/2) + \Theta(n^3)$$

Compare $n^{\log_2 5}$ vs. n^3.

Master Theorem

<u>Case 3</u>: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$
and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

<u>Solution:</u> $T(n)$ **is** $\Theta(f(n))$

$$T(n) = 5T(n/2) + \Theta(n^3)$$

Compare $n^{\log_2 5}$ vs. n^3.

Since $3 > \log_2 5$ use **<u>Case 3</u>**

Master Theorem

Case 3: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$
and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

Solution: $T(n)$ **is** $\Theta(f(n))$

$$T(n) = 5T(n/2) + \Theta(n^3)$$

Compare $n^{\log_2 5}$ vs. n^3.

Since $3 > \log_2 5$ use **Case 3**

Master Theorem

Case 3: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

Solution: $T(n)$ **is** $\Theta(f(n))$

$T(n) = 5T(n/2) + \Theta(n^3)$

Compare $n^{\log_2 5}$ vs. n^3.

Since $3 > \log_2 5$ use **Case 3**

$af(n/b) = 5(n/2)^3 = 5/8\ n^3 \leq cn^3$, for $c = 5/8$

Master Theorem

<u>Case 3</u>: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

<u>Solution:</u> $T(n)$ **is** $\Theta(f(n))$

$T(n) = 5T(n/2) + \Theta(n^3)$

Compare $n^{\log_2 5}$ vs. n^3.

Since $3 > \log_2 5$ use **<u>Case 3</u>**

$af(n/b) = 5(n/2)^3 = 5/8\, n^3 \leq cn^3$, for $c = 5/8$

Master Theorem

<u>Case 3</u>: $f(n)$ **is** $\Omega(n^L)$ for some constant $L > \log_b a$
and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

<u>Solution:</u> $T(n)$ **is** $\Theta(f(n))$

$T(n) = 5T(n/2) + \Theta(n^3)$

Compare $n^{\log_2 5}$ vs. n^3.

Since $3 > \log_2 5$ use **<u>Case 3</u>**

$af(n/b) = 5(n/2)^3 = 5/8\ n^3 \leq cn^3$, for $c = 5/8$

<u>Solution:</u> $T(n)$ **is** $\Theta(n^3)$

~~Master Theorem~~

<u>Case 2</u>: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

~~Master Theorem~~

<u>Case 2</u>: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3/\log n)$$

~~Master Theorem~~

<u>Case 2</u>: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3/\log n)$$

~~Master Theorem~~

<u>Case 2</u>: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

<u>Solution:</u> $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3/\log n)$$

Compare $n^{\log_3 27}$ vs. n^3.

~~Master Theorem~~

Case 2: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3/\log n)$$

Compare $n^{\log_3 27}$ vs. n^3.

~~Master Theorem~~

Case 2: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$T(n) = 27T(n/3) + \Theta(n^3/\log n)$

Compare $n^{\log_3 27}$ vs. n^3.

Since $3 = \log_3 27$ use **Case 2**

~~Master Theorem~~

Case 2: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$T(n) = 27T(n/3) + \Theta(n^3/\log n)$

Compare $n^{\log_3 27}$ vs. n^3.

Since $3 = \log_3 27$ use **Case 2**

~~Master Theorem~~

Case 2: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$T(n) = 27T(n/3) + \Theta(n^3/\log n)$

Compare $n^{\log_3 27}$ vs. n^3.

Since $3 = \log_3 27$ use **Case 2**

but $n^3/\log n$ **is not** $\Theta(n^3 \log^k n)$ for $k \geq 0$

~~Master Theorem~~

Case 2: $f(n)$ **is** $\Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n)$ **is** $\Theta(n^{\log_b a} \log^{k+1} n)$

$T(n) = 27T(n/3) + \Theta(n^3/\log n)$

Compare $n^{\log_3 27}$ vs. n^3.

Since $3 = \log_3 27$ use **Case 2**

but $n^3/\log n$ **is not** $\Theta(n^3 \log^k n)$ for $k \geq 0$

Cannot use Master Method.

Divide-and-Conquer Paradigm

Divide-and-Conquer

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

Divide-and-Conquer

Divide-and-conquer.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

Divide-and-Conquer

Divide-and-conquer.
- Break up problem into several parts.
- Solve each part recursively.

> Divide et impera.
> Veni, vidi, vici.
> - Julius Caesar

Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

> Divide et impera.
> Veni, vidi, vici.
> - Julius Caesar

Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

- Break up problem of size n into **two** equal parts of size $^n/_2$.

> Divide et impera.
> Veni, vidi, vici.
> - Julius Caesar

Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

- Break up problem of size n into **two** equal parts of size $n/2$.
- Solve two parts recursively.

> Divide et impera.
> Veni, vidi, vici.
> - Julius Caesar

Divide-and-Conquer

Divide-and-conquer.
- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.
- Break up problem of size n into **two** equal parts of size $^n/_2$.
- Solve two parts recursively.
- Combine two solutions into overall solution in **linear time**.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

Divide-and-Conquer

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

- Break up problem of size n into **two** equal parts of size $^n/_2$.
- Solve two parts recursively.
- Combine two solutions into overall solution in **linear time**.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Most common usage.

- Break up problem of size n into **two** equal parts of size $^n/_2$.
- Solve two parts recursively.
- Combine two solutions into overall solution in **linear time**.

> Divide et impera.
> Veni, vidi, vici.
> - Julius Caesar

Divide-and-Conquer

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

- Break up problem of size n into **two** equal parts of size $^n/_2$.
- Solve two parts recursively.
- Combine two solutions into overall solution in **linear time**.

Consequence.

> Divide et impera.
> Veni, vidi, vici.
> \- Julius Caesar

Divide-and-Conquer

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

- Break up problem of size n into **two** equal parts of size $^n/_2$.
- Solve two parts recursively.
- Combine two solutions into overall solution in **linear time**.

Consequence.

- Straightforward: n^2.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

Divide-and-Conquer

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

- Break up problem of size n into **two** equal parts of size $^n/_2$.
- Solve two parts recursively.
- Combine two solutions into overall solution in **linear time**.

Consequence.

- Straightforward: n^2.
- Divide-and-conquer: n log n.

> Divide et impera.
> Veni, vidi, vici.
> - Julius Caesar

Divide-and-Conquer: Binary Search

Binary Search

Find a value v in a sorted array of elements.

$$[a_0 \le a_1 \le ,\dots, \le a_{Size-1}]$$

Size = number of elements.

Binary Search

Algorithm: binarySearch$(a, v, low, high)$
Input: array a, value v, lower and upper bound indices $low, high$ ($low = 0, high = n - 1$ initially)
Output: the index i of element v (if it is present), -1 (if v is not present)

```
if low == high then
    if a[low] == v then
        return low
    else
        return −1
    end if
else
    mid ← (low + high)/2
    if v ≤ a[mid] then
        return binarySearch(a, v, low, mid)
    else
        return binarySearch(a, v, mid + 1, high)
    end if
end if
```

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

	0	0000	L	L			
	1	0001					
	2	0010					
	3	0011					
	4	0100			L	L	
*	5	0101				H	L=H
	6	0110					
	7	0111		H	H		
	8	1000					
	9	1001					
	10	1010					
	11	1011					
	12	1100					
	13	1101					
	14	1110					
	15	1111	H				

Binary Search

```
  0   L
  1
  2
  3       L   L
* 4           H  L=H
  5   H   H
```

Binary Search

	0	L			
	1				
	2				
	3		L	L	
*	4			H	L=H
	5	H	H		

Binary Search

```
  0   L
  1
  2
  3       L   L
* 4           H  L=H
  5   H   H
```

Binary Search

	0	L			
	1				
	2				
	3		L	L	
*	4			H	L=H
	5	H	H		

Binary Search

```
  0   L
  1
  2
  3       L   L
* 4           H  L=H
  5   H   H
```

Binary Search

```
  0   L
  1
  2
  3       L   L
* 4           H   L=H
  5   H   H
```

Binary Search

```
  0   L
  1
  2
  3       L   L
* 4           H  L=H
  5   H   H
```

Binary Search

0	L			
1				
2				
3		L	L	
* 4			H	L=H
5	H	H		

Binary Search

```
  0   L
  1
  2
  3       L   L
* 4           H  L=H
  5   H   H
```

Recurrence Relation

Def. T(n) = number of comparisons to find v among n sorted elements.

Binary Search recurrence.

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

Solution. T(n) is O(log n) (Master Theorem Case 2).

Divide-and-Conquer Multiplication

Integer Multiplication

Multiply. Given two n-digit integers a and b, compute a × b.

- Grade School solution: $\Theta(n^2)$ bit operations.

								1	1	0	1	0	1	0	1	0
								×	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1	0
							0	0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1	0		
					1	1	0	1	0	1	0	1	0			
				1	1	0	1	0	1	0	1	0				
			1	1	0	1	0	1	0	1	0					
		1	1	0	1	0	1	0	1	0						
	0	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0

Multiply

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Add

Integer Multiplication

Add. Given two n-digit integers a and b, compute a + b.

- $\Theta(n)$ bit operations.

Multiply. Given two n-digit integers a and b, compute a × b.

- Grade School solution: $\Theta(n^2)$ bit operations.

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Add

Multiply

								1	1	0	1	0	1	0	1	0
								×	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1	0
							0	0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1	0		
					1	1	0	1	0	1	0	1	0			
				1	1	0	1	0	1	0	1	0				
			1	1	0	1	0	1	0	1	0					
		1	1	0	1	0	1	0	1	0						
	0	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0

D&C Multiplication

To multiply two n-digit integers:

- Multiply four $^n/_2$-digit integers.
- Add two $^n/_2$-digit integers, and shift to obtain result.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0
\end{aligned}
$$

$$
\mathrm{T}(n) = \underbrace{4T\left(n/2\right)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow \mathrm{T}(n) \text{ is } \Theta(n^2)
$$

↑

assumes n is a power of 2

Telescoping Proof

$$T(n) = \underbrace{4T\left(n/2\right)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Telescoping Proof

Claim.

$$\mathrm{T}(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow \mathrm{T}(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Telescoping Proof

Claim.

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Telescoping Proof

Claim.

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Telescoping Proof

Claim.

$$\underbrace{\mathrm{T}(n) = 4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow \mathrm{T}(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Telescoping Proof

Claim.

$$\mathrm{T}(n) = \underbrace{4T\left(n/2\right)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow \mathrm{T}(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Pf. For n > 1: T(n)/n = 4T(n/2)/n + C

Telescoping Proof

Claim.

$$\underbrace{T(n) = 4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Pf. For n > 1: $T(n)/n = 4T(n/2)/n + C$

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

Telescoping Proof

Claim.

$$\mathrm{T}(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow \mathrm{T}(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Pf. For n > 1:

$$\begin{aligned} T(n)/n &= 4T(n/2)/n + C \\ &= 2\,T(n/2)/(n/2) + C \\ &= 2\,[\,2\,T(n/4)/(n/4) + C\,] + C \end{aligned}$$

Telescoping Proof

Claim.

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Pf. For n > 1:

$$\begin{aligned} T(n)/n &= 4T(n/2)/n + C \\ &= 2\,T(n/2)/(n/2) + C \\ &= 2\,[\,2\,T(n/4)/(n/4) + C\,] + C \\ &= 4\,T(n/4)/(n/4) + 2C + C \end{aligned}$$

Telescoping Proof

Claim.

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Pf. For n > 1:

$$\begin{aligned} T(n)/n &= 4T(n/2)/n + C \\ &= 2\,T(n/2)/(n/2) + C \\ &= 2\,[\,2\,T(n/4)/(n/4) + C\,] + C \\ &= 4\,T(n/4)/(n/4) + 2C + C \\ &= 4\,[\,2\,T(n/8)/(n/8) + C\,] + 2C + C \end{aligned}$$

Telescoping Proof

Claim.

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Pf. For n > 1:

$$\begin{aligned} T(n)/n &= 4T(n/2)/n + C \\ &= 2\,T(n/2)/(n/2) + C \\ &= 2\,[\,2\,T(n/4)/(n/4) + C\,] + C \\ &= 4\,T(n/4)/(n/4) + 2C + C \\ &= 4\,[\,2\,T(n/8)/(n/8) + C\,] + 2C + C \\ &= 8\,T(n/8)/(n/8) + 4C + 2C + C \end{aligned}$$

Telescoping Proof

Claim.

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Pf. For n > 1:

$$\begin{aligned} T(n)/n &= 4T(n/2)/n + C \\ &= 2\,T(n/2)/(n/2) + C \\ &= 2\,[\,2\,T(n/4)/(n/4) + C\,] + C \\ &= 4\,T(n/4)/(n/4) + 2C + C \\ &= 4\,[\,2\,T(n/8)/(n/8) + C\,] + 2C + C \\ &= 8\,T(n/8)/(n/8) + 4C + 2C + C \\ &\ldots \end{aligned}$$

Telescoping Proof

Claim.

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Pf. For n > 1:

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

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

$= 2\,[\,2\,T(n/4)/(n/4) + C\,] + C$

$= 4\,T(n/4)/(n/4) + 2C + C$

$= 4\,[\,2\,T(n/8)/(n/8) + C\,] + 2C + C$

$= 8\,T(n/8)/(n/8) + 4C + 2C + C$

…

$= n\,T(1)/1 + n/2\,C + n/4\,C + \ldots + 4C + 2C + C$

Telescoping Proof

Claim.

$$\underbrace{T(n) = 4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) \text{ is } \Theta(n^2)$$

↑ assumes n is a power of 2

Pf. For n > 1:

$$\begin{aligned}
T(n)/n &= 4T(n/2)/n + C \\
&= 2\,T(n/2)/(n/2) + C \\
&= 2\,[\,2\,T(n/4)/(n/4) + C\,] + C \\
&= 4\,T(n/4)/(n/4) + 2C + C \\
&= 4\,[\,2\,T(n/8)/(n/8) + C\,] + 2C + C \\
&= 8\,T(n/8)/(n/8) + 4C + 2C + C \\
&\dots \\
&= n\,T(1)/1 + n/2\,C + n/4\,C + \dots + 4C + 2C + C \\
&= C\,(n/2 + n/4 + \dots + 2 + 1) = C(n-1).
\end{aligned}$$

Karatsuba Multiplication

To multiply two n-digit integers:

- Add two $^n/_2$ digit integers.
- Multiply **three** $^n/_2$-digit integers.
- Add, subtract, and shift $^n/_2$-digit integers to obtain result.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^{n} \cdot \underset{A}{x_1 y_1} + 2^{n/2} \cdot \big(\underset{B}{(x_1 + x_0)(y_1 + y_0)} - \underset{A}{x_1 y_1} - \underset{C}{x_0 y_0} \big) + \underset{C}{x_0 y_0}
\end{aligned}
$$

Theorem. [Karatsuba-Ofman, 1962] Can multiply two n-digit integers in $O(n^{1.585})$ bit operations.

$$
T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}}
$$

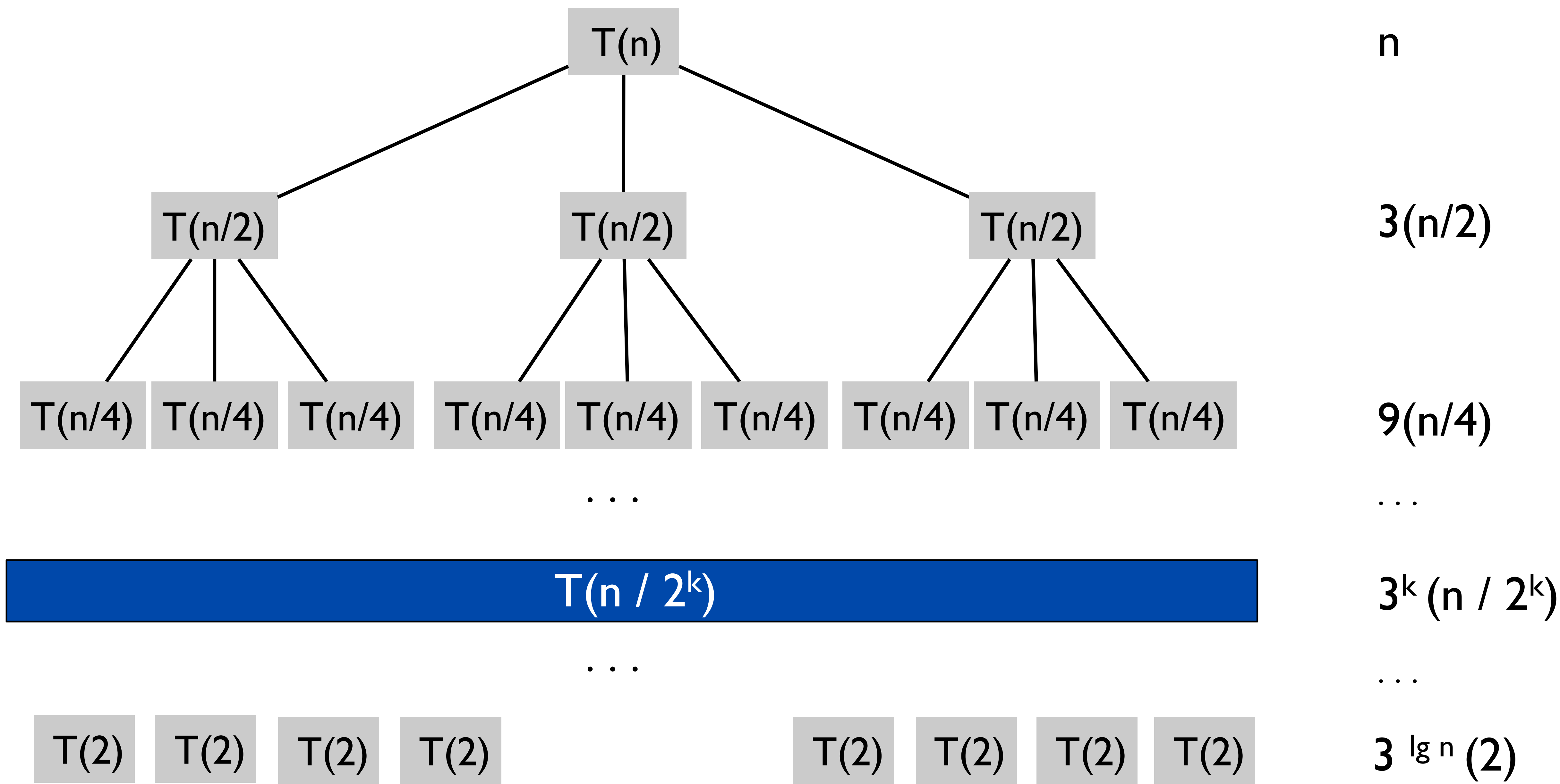
$$
\Rightarrow T(n) \text{ is } O(n^{\log_2 3}) \text{ is } O(n^{1.585})
$$

$$\sum_{k=0}^{n-1} ar^k = a\,\frac{1-r^n}{1-r}$$

Karatsuba Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\log_2 n} n \left(\tfrac{3}{2}\right)^k = n\frac{\left(\tfrac{3}{2}\right)^{1+\log_2 n} - 1}{\tfrac{3}{2} - 1} = 3n^{\log_2 3} - 2$$



Karatsuba Multiplication

Generalization: $O(n^{1+\varepsilon})$ for any $\varepsilon > 0$.

Best known: $n \log n \, 2^{O(\log^* n)}$

where $\log^*(x) = \begin{cases} 0 & \text{if } x \le 1 \\ 1 + \log^*(\log x) & \text{if } x > 1 \end{cases}$

Conjecture: $\Omega(n \log n)$ but not proven yet.

Winter 2016
COMP-250: Introduction to Computer Science

Lecture 12, February 18, 2016