CS 381 – Spring 2021

Week 2, Lecture 2
Part 1

Many recurrence relations arising from divide-and-conquer algorithms have the form:

$$\mathbf{T(n) = aT(n/b) + f(n)}$$

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

- There are **a** instances of the problem, each of size **n/b**
  - a is the branching factor
  - b is the reduction factor

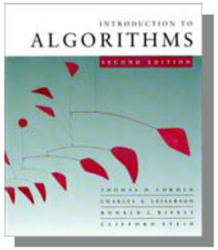
- Setting up the problem instance before the recursive calls and combining subsolutions returned by the recursive calls takes **f(n)** work.

Master Theorem (4.5 CLRS)

- The Master Theorem (MS) is a method for solving many recurrences of the form

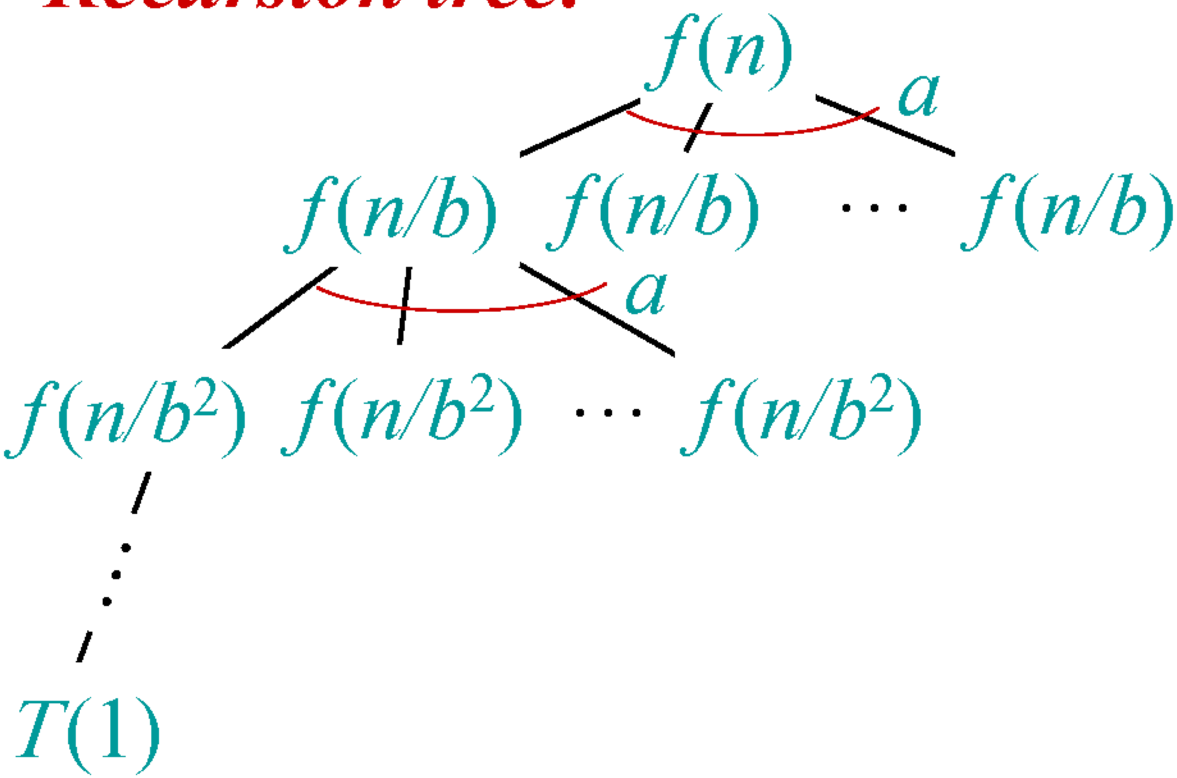
$$\mathbf{T(n) = aT(n/b) + f(n)}$$

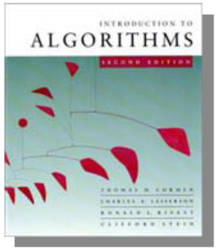
- There exist multiple versions of the Master Theorem
  - All are based on the same idea, with three cases
  - Each with slightly different assumption about $f(n)$; ***none cover all possibilities***



Idea of master theorem

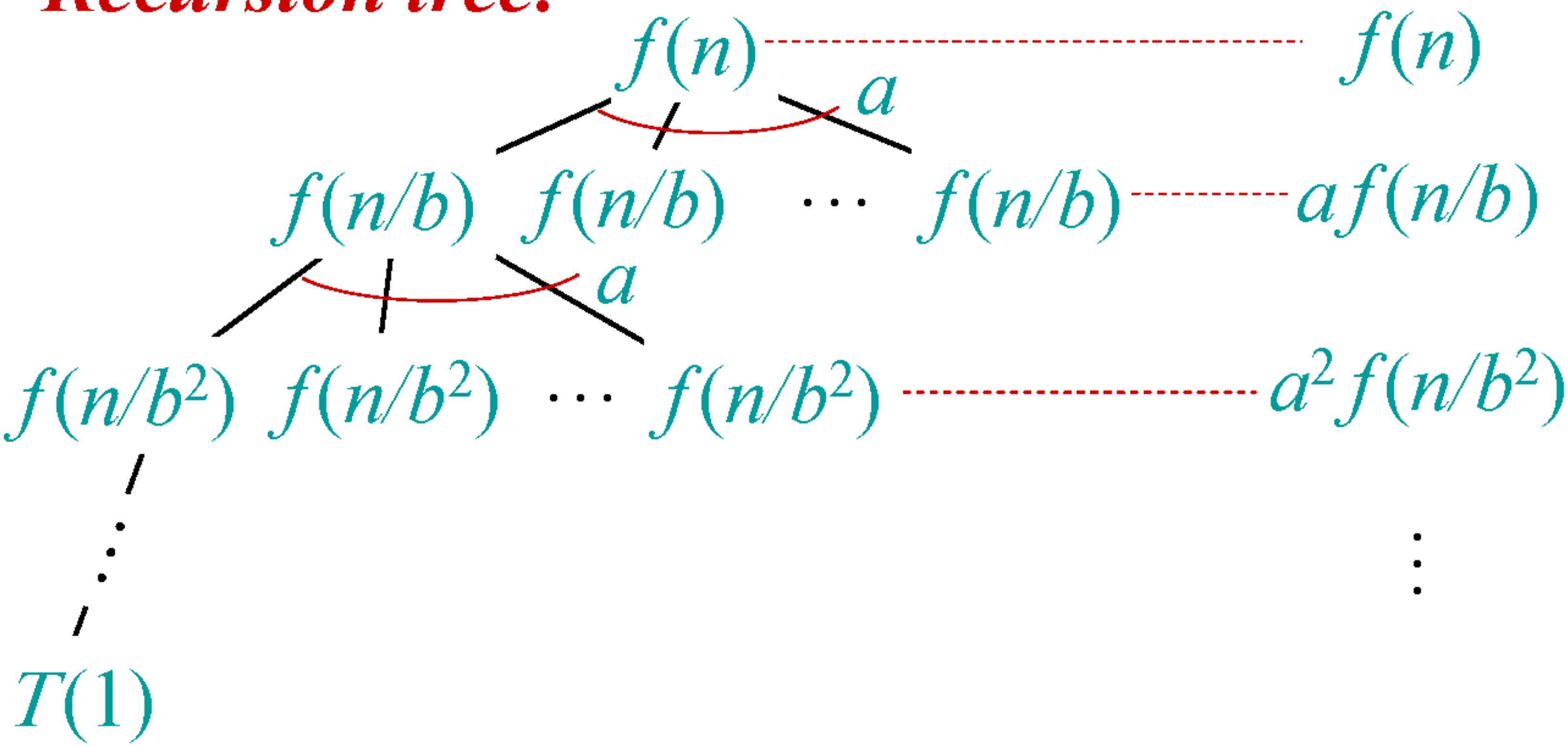
Recursion tree:

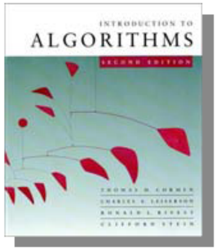




Idea of master theorem

Recursion tree:





Idea of master theorem

Recursion tree:

$$h = \log_b n$$

$f(n)$ ——— $f(n)$

a

$f(n/b) \quad f(n/b) \quad \cdots \quad f(n/b)$ ——— $a\,f(n/b)$

a

$f(n/b^2) \quad f(n/b^2) \quad \cdots \quad f(n/b^2)$ ——— $a^2 f(n/b^2)$

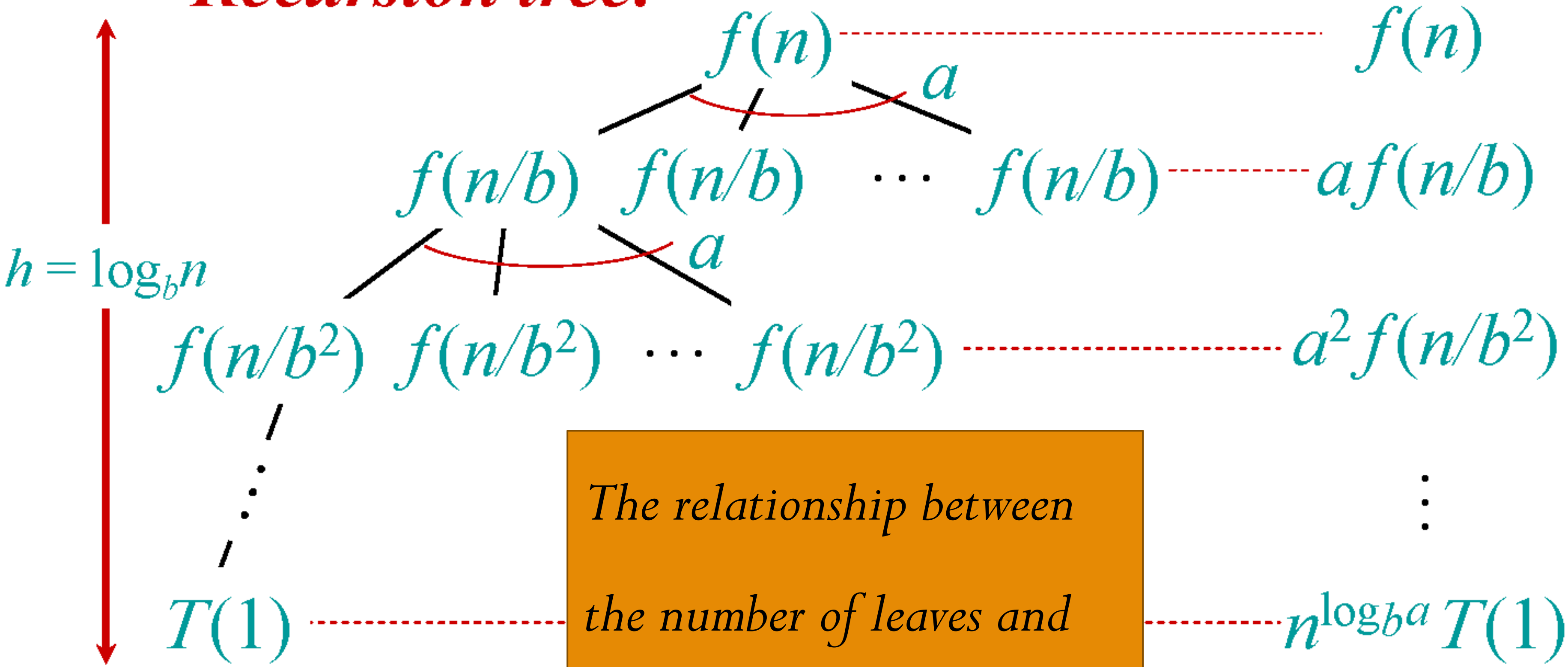
$\vdots$

$T(1)$ ——— $n^{\log_b a}\,T(1)$

$$\begin{aligned} \#\text{leaves} &= a^h \\ &= a^{\log_b n} \\ &= n^{\log_b a} \end{aligned}$$

Idea of master theorem

Recursion tree:



The relationship between the number of leaves and f(n) determines T(n).

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

- Height of recursion tree: $\log_b n$
- Number of leaves: $n^{\log_b a}$
- Total number of nodes in the recursion tree:

$$\sum_{i=0}^{(\log_b n)-1} a^i = \frac{a^{\log_b n} - 1}{a-1}$$

Reminder: $\boldsymbol{\sum_{i=0}^{k} x^i = \frac{x^{k+1}-1}{x-1}}$

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

Matches $T(n) = a\,T(n/b) + f(n)$ with $a = 4, b = 2, f(n) = n$.

- Number of levels in recursion tree: $\log_2 n$
- Number of leaves: $n^{log_b a} = n^{log_2 4} = n^2$
- Work done for a problem of size m: $f(m) = O(m)$

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

Total work done:

$$n + 4\frac{n}{2} + 4^2\frac{n}{2^2} + 4^3\frac{n}{2^3} + \ldots + n^2$$

$$= n + 2n + 4n + 8n + \ldots + n2^{\log n}$$

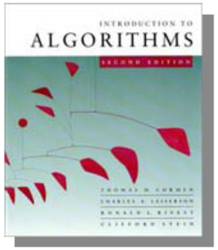
$$= n\sum_{i=0}^{\log n} 2^i = n\left(2^{(\log n)+1} - 1\right) = n(2n-1)$$

$$= O(n^2)$$

Master Theorem

Let $a \geq 1$ and $b > 1$ be constants. Let $f(n)$ be a function, and $T(n)$ be defined on the nonnegative integers by $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for a constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$.
3. If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for a constant $\epsilon > 0$, and if $\mathrm{a} \cdot \mathrm{f}\left(\frac{\mathrm{n}}{\mathrm{b}}\right) \leq cf(n)$ for some constant c > 1 and all large enough n, then $T(n) = \Theta(f(n))$.



Idea of master theorem

Recursion tree:

$f(n)$ .... $f(n)$

a

$f(n/b) \quad f(n/b) \quad \cdots \quad f(n/b)$ .... $af(n/b)$

a

$h = \log_b n$

$f(n/b^2) \quad f(n/b^2) \quad \cdots \quad f(n/b^2)$ .... $a^2 f(n/b^2)$

$\vdots$

$T(1)$ .... $n^{\log_b a} T(1)$

Case 1: Work to split/recombine a problem is dwarfed by subproblems.

$\Theta(n^{\log_b a})$

Master Theorem: T(n) = aT(n/b) + f(n)

Case 1: $f(n) = O\left(n^{\log_b(a)-\varepsilon}\right)$, *for a constant* $\varepsilon > 0$.

This means the leaves do a fraction of the total work (number of nodes dominate the time, not the work done at the nodes).

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

Master Theorem: T(n) = aT(n/b) + f(n)

Case 1: $f(n) = O\left(n^{\log_b(a)-\varepsilon}\right)$, *for a constant* $\varepsilon > 0$.

This means the leaves do a fraction of the total work (number of nodes dominate the time, not the work done at the nodes).

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

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

What is T(n)?

Master Theorem: T(n) = aT(n/b) + f(n)

Case 1: $f(n) = O\left(n^{\log_b(a)-\varepsilon}\right)$, *for a constant* $\varepsilon > 0$.

This means the leaves do a fraction of the total work (number of nodes dominate the time, not the work done at the nodes).

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

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

$a=4,\ b=2,\ f(n) = n$

$n^{\log_b a} = n^2$ versus $f(n) = O(n)$ [set $\varepsilon = 1$]

Hence, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$

Another example: $T(n) = 5T(n/2) + n^2$

What is $T(n)$?

Another example: $T(n) = 5T(n/2) + n^2$

- $a = 5, b = 2$
- $f(n) = n^2$

Number of levels: $\log_2 n$

Number of leaves: $n^{\log_b a} = n^{\log_2 5} \approx n^{2.32}$

Does n^2 grow polynomially slower than $n^{2.32}$? Yes, by $n^{0.3}$

$$\mathbf{T(n) = \Theta(n^{\log_2 5})}$$

f(n) grows polynomially slower than g(n)

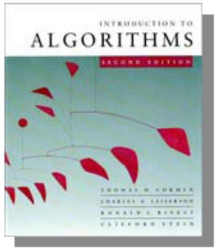
Does a polynomial n^{ε} separate f(n) and g(n)?

f(n)	**g(n)**
n	n^2
O(1)	n log n
$\log^8 n$	$n^{1.3}$
log n	$\sqrt{n}$
n	n log n
log n	$(\log n)^3$
n^5	2^n

f(n) grows polynomially slower than g(n)

Does a polynomial n^{ε} separate f(n) and g(n)?

f(n)	**g(n)**
n	n^2
O(1)	n log n
$\log^8 n$	$n^{1.3}$
log n	$\sqrt{n}$
n	*n log n* -- *not true*
log n	*(log n)^3* -- *not true*
n^5	2^n



Idea of master theorem

Recursion tree:

$f(n)$ .... $f(n)$

a

$f(n/b)$ $f(n/b)$ $\cdots$ $f(n/b)$ .... $a f(n/b)$

a

$h = \log_b n$

$f(n/b^2)$ $f(n/b^2)$ $\cdots$ $f(n/b^2)$ .... $a^2 f(n/b^2)$

$\vdots$

$T(1)$ .... $n^{\log_b a} T(1)$

Case 2: Work to split/recombine a problem is comparable to subproblems.

$\Theta(n^{\log_b a} \lg n)$

Master Theorem: $T(n) = aT(n/b) + f(n)$

Case 2 : $f(n) = \theta\,(n^{\log_b a})$. This means $f(n)$ and $n^{\log_b a}$ grow at the same rate, so total work done at every level is about the same.

Then, $T(n) = \theta(n^{\log_b a} \log n)$.

Master Theorem: $T(n) = aT(n/b) + f(n)$

Case 2 : $f(n) = \boldsymbol{\theta}\ (n^{\log_b a})$. *This means* $\boldsymbol{f}(\boldsymbol{n})$ *and* $\boldsymbol{n}^{\log_b a}$ *grow at the same rate, so total work done at every level is about the same.*

Then, $\boldsymbol{T}(\boldsymbol{n}) = \theta(\mathbf{n}^{\log_b a} \log \boldsymbol{n})$.

Example:

$\mathbf{T(n) = 2T(n/2) + cn.}$

What is T(n)?

Master Theorem: T(n) = aT(n/b) + f(n)

Case 2 : f(n) = θ ($n^{\log_b a}$). This means $f(n)$ and $n^{\log_b a}$ grow at the same rate, so total work done at every level is about the same.

Then, $T(n) = \theta(n^{\log_b a} \log n)$.

Example:

$T(n) = 2T(n/2) + cn = \theta\ (n \log n)$

[a=b=2, $n^{\log_b a} = n = f(n)$]

Master Theorem: $T(n) = aT(n/b) + f(n)$

Case 2 : $f(n) = \theta\,(n^{\log_b a})$. *This means* $f(n)$ *and* $n^{\log_b a}$ *grow at the same rate, i.e., total work done at every level is about the same.*

Then, $T(n) = \Theta(n^{\log_b a} \log n)$.

More Examples:

$T(n) = T(n/2) + c$

What is T(n)?

Master Theorem: $T(n) = aT(n/b) + f(n)$

Case 2 : $f(n) = \theta\,(n^{\log_b a})$. This means $f(n)$ and $n^{\log_b a}$ grow at the same rate, i.e., total work done at every level is about the same.

Then, $T(n) = \Theta(n^{\log_b a} \log n)$.

More Examples:

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

$a=1,\ b=2,\ f(n) = O(1) = n^{\log_2 1} = O(1)$

Master Theorem: $T(n) = aT(n/b) + f(n)$

Case 2 : *$f(n) = \theta\ (n^{\log_b a})$. This means $f(n)$ and $n^{\log_b a}$ grow at the same rate, i.e., total work done at every level is about the same.*

Then, $T(n) = \Theta(n^{\log_b a} \log n)$.

More Examples:

$T(n) = T(n/2) + c = \Theta\ (\log n)$

$a=1,\ b=2,\ f(n) = O(1) = n^{\log_2 1} = O(1)$

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

What is T(n)?

Master Theorem: $T(n) = aT(n/b) + f(n)$

Case 2 : $f(n) = \theta\,(n^{\log_b a})$. This means $f(n)$ and $n^{\log_b a}$ grow at the same rate, i.e., total work done at every level is about the same.

Then, $T(n) = \theta(n^{\log_b a} \log n)$.

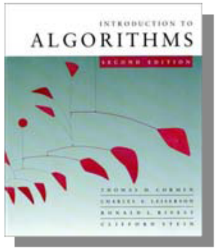
More Examples:

$T(n) = T(n/2) + c = \theta\,(\log n)$

$a=1,\ b=2,\ f(n) = O(1) = n^{\log_2 1} = O(1)$

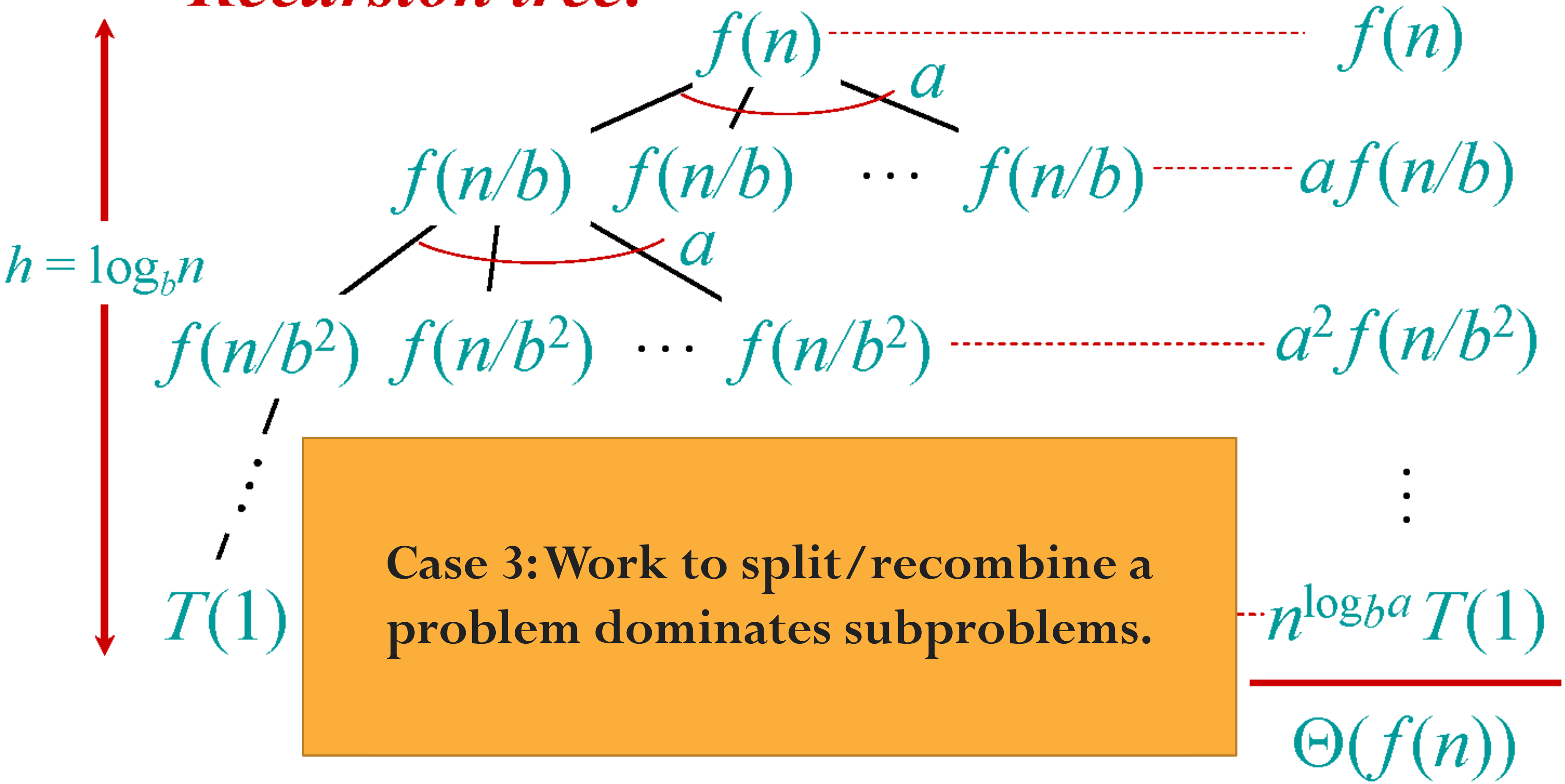
$T(n) = 4T(n/2) + n^2 = \theta\,(n^2 \log n)$

$a=4,\ b=2,\ f(n) = n^2 = n^{\log_2 4} = n^2$



Idea of master theorem

Recursion tree:



Case 3: Work to split/recombine a problem dominates subproblems.

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

Case 3: f(n) grows polynomially faster than $n^{\log_b a}$ *; i.e.,*

$f(n) = \Omega(n^{\log_b a})$ *and a regularity condition is satisfied:*

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n) \text{ for some constant } c < 1$$

Then, $T(n) = \Theta(f(n))$.

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

Case 3: f(n) grows polynomially faster than $n^{\log_b a}$*; i.e.,* $f(n) = \Omega(n^{\log_b a})$ *and a regularity condition is satisfied:*

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n) \text{ for some constant } c < 1$$

Then, $T(n) = \Theta(f(n))$.

E.g.: $T(n) = T(n/2) + n \log n$.

What is T(n)?

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

Case 3: f(n) grows polynomially faster than $n^{\log_b a}$ *; i.e.,*
$f(n) = \Omega(n^{\log_b a})$ *and a regularity condition is satisfied:*

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n) \text{ for some constant } c < 1$$

Then, $T(n) = \Theta(f(n))$.

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

- $a = 1, b = 2, n^{\log_b a} = O(1), f(n) = n \log n$
- Regularity condition: $1 * \frac{n}{2} * \log \frac{n}{2} < \frac{n}{2} \log n = \frac{1}{2} n \log n$
- We have $c = \frac{1}{2} < 1$