Master Theorem

Solving Recurrences

Master Theorem

- Solve $T(n) = 2T(n/3) + f(n)$
- Master Theorem helps in solving the recurrences which characterize divide and conquer algorithms.
- Master Theorem is based on the recursion tree model, which helps in estimating the costs (time) of a recursive execution of an algorithm.

Master Theorem

Solve $T(n) = 2T(n/3) + f(n)$:
$T(1) = 1$

$f(n) can\ be\ n, n^2, \ldots$

The no. of program steps of an algorithm is $f(n)$ plus 2 times the program steps needed for processing $n/3$ elements. The algorithm makes two recursive calls with $n/3$ elements each.

Example of recursion tree

Solve $T(n) = 2T(n/3) + f(n)$:

$$T(n)$$

Example of recursion tree

Solve $T(n) = 2T(n/3) + f(n)$:

$$
\begin{array}{ccc}
 & f(n) & \\
 \diagup & & \diagdown \\
T(n/3) & & T(n/3)
\end{array}
$$

Example of recursion tree

Solve $T(n) = 2T(n/3) + f(n)$:

$$f(n)$$

$$f(n/3) \qquad\qquad f(n/3)$$

$$T(n/9) \quad T(n/9) \quad T(n/9) \quad T(n/9)$$

Example of recursion tree

Solve $T(n) = 2T(n/3) + f(n)$:

$$
\begin{array}{cccc}
 & & f(n) & \\
 & f(n/3) & & f(n/3) \\
f(n/9) & f(n/9) & f(n/9) & f(n/9)
\end{array}
$$

Example of recursion tree

Solve $T(n) = 2T(n/3) + f(n)$:

$$f(n)$$

$$f(n/3) \qquad\qquad f(n/3)$$

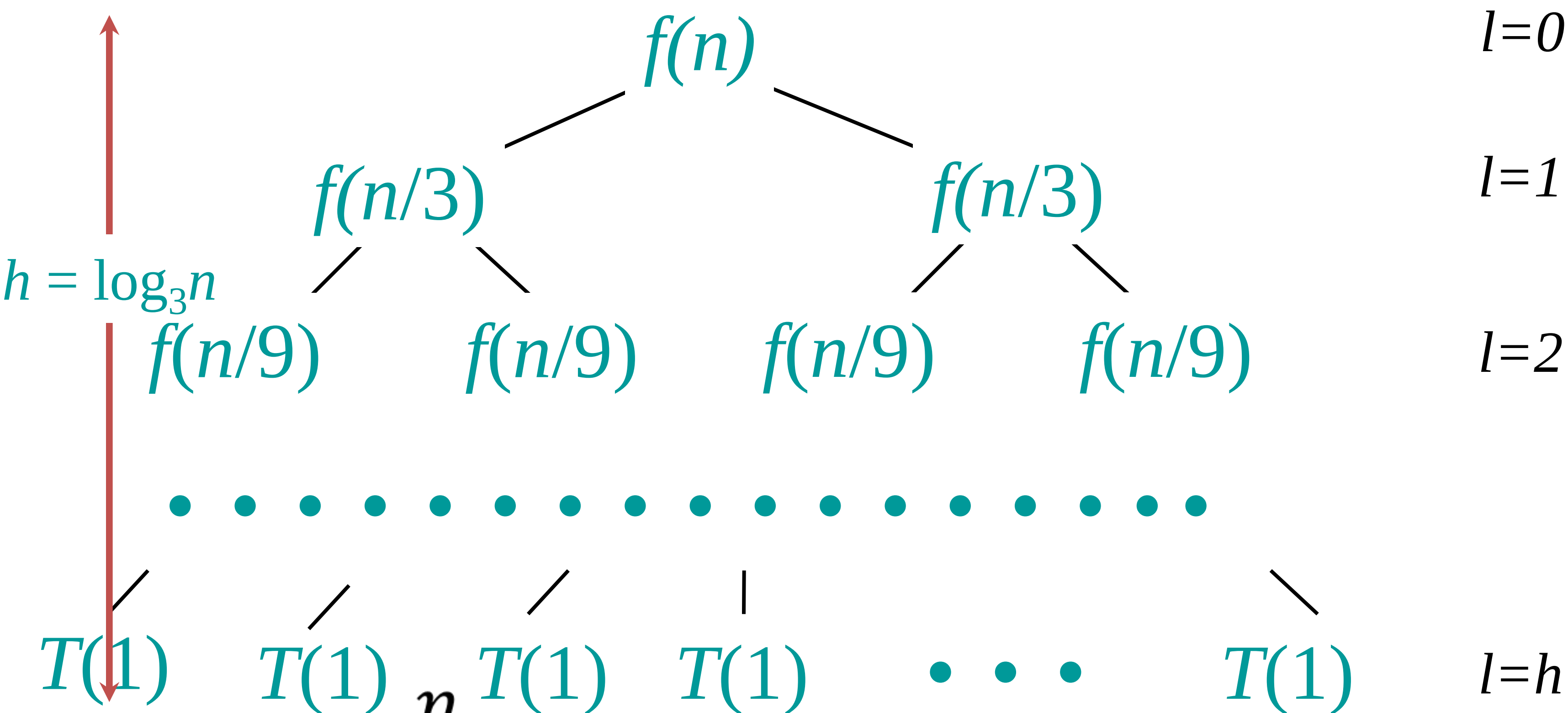
$$f(n/9) \qquad f(n/9) \qquad f(n/9) \qquad f(n/9)$$

$$\cdots$$

$$T(1) \quad T(1) \quad T(1) \quad T(1) \quad \cdots \quad T(1)$$

Example of recursion tree

Solve $T(n) = 2T(n/3) + f(n)$:



$$\frac{n}{3^h} = 1, 3^h = n, h = \log_3 n$$

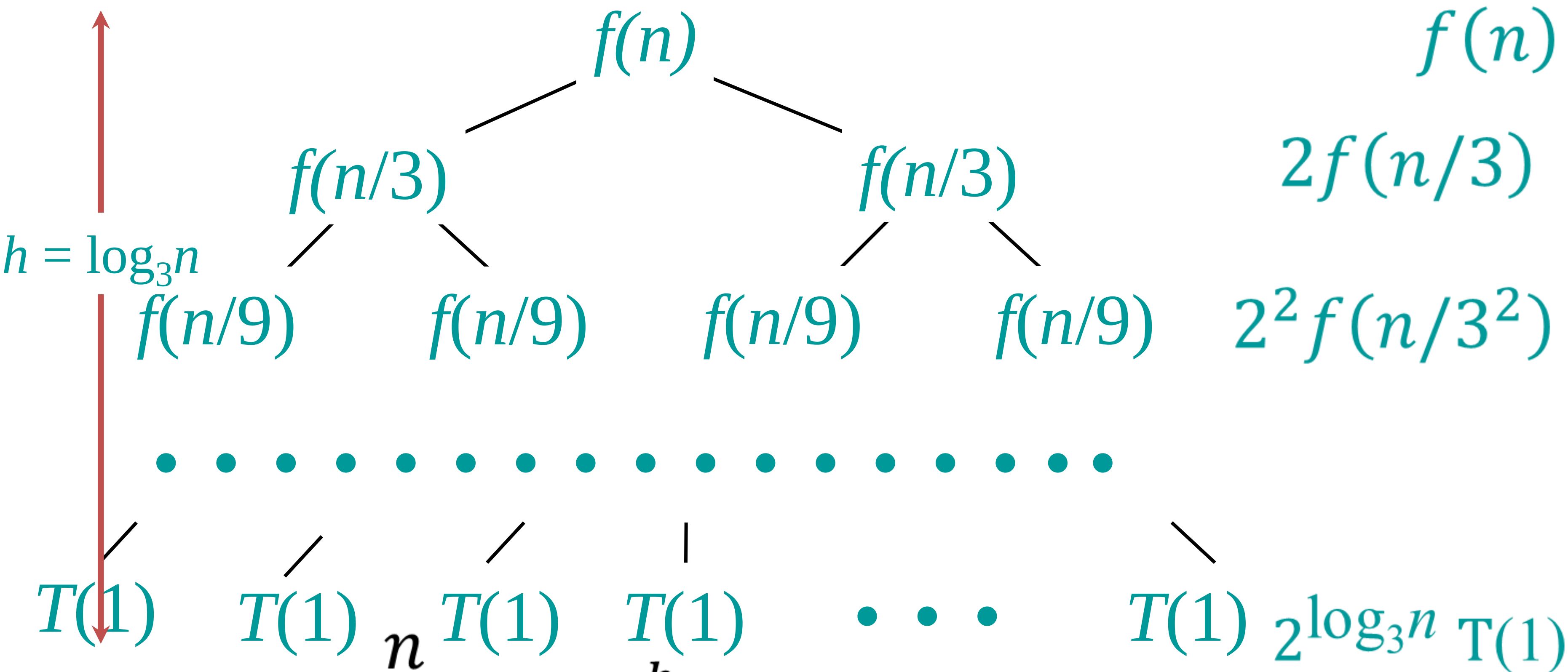
Recursion tree

- if $T(n) = aT(n/b) + f(n)$
- *The height of the recursion tree is*

$$\log_b n$$

Example of recursion tree

Solve $T(n) = 2T(n/3) + f(n)$:



$$\frac{n}{3^h} = 1, 3^h = n, h = \log_3 n$$

Recursion tree

- If $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- The number of nodes at each level are
- $a^0, a^1, a^2, a^3, \ldots,$
- The number of leaf nodes are

  $a^{\log_b n}$= $n^{\log_b a}$

Example of recursion tree

- The performance analysis of a recursive algorithm defined by

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

- is governed by the order of the first term which effects the number of leaves in the recursion tree, and the order of $f(n)$, the computational effort needed at each recursive step.

Example of recursion tree

- Consider $T(n) = 4T\left(\frac{n}{2}\right) + n$
- $a = 4, b = 2, \log_b a$=2
- The leaves are in the order of n^2
- $f(n)$ is in the order of n
- The complexity is dominated by the leaves than $f(n)$.
- Hence $T(n) = \theta\left(n^{\log_b a}\right) = \theta(n^2)$

Example of recursion tree

- Consider $T(n) = 4T\left(\frac{n}{2}\right) + n^3$
- $a = 4, b = 2, \log_b a$=2
- The leaves are in the order of n^2
- $f(n)$ is in the order of n^3
- The complexity is dominated by $f(n)$ than the leaves.
- Hence $T(n) = \theta(n^3)$

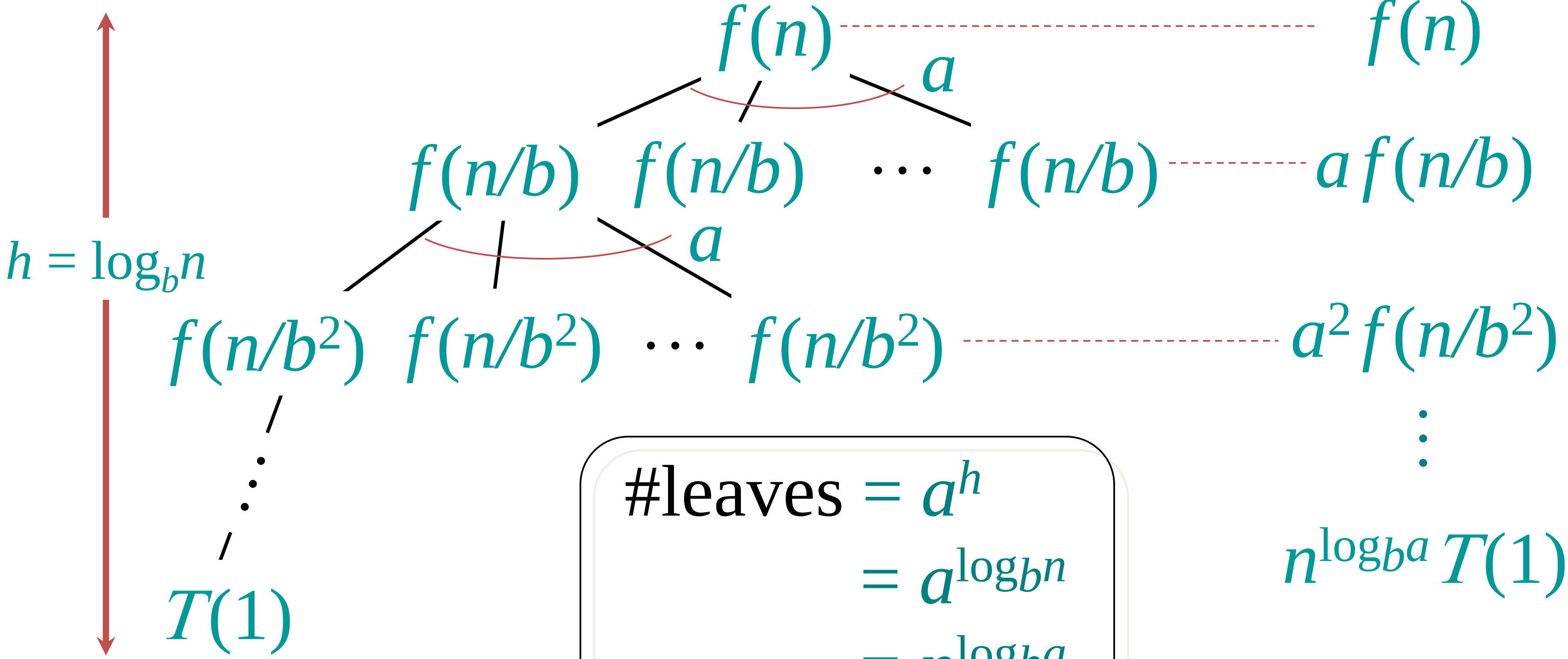
The master method

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.

Idea of master theorem

Recursion tree:



$$\#\text{leaves} = a^h = a^{\log_b n} = n^{\log_b a}$$

Three common cases

1. $f(n) = \theta(n^k)$ *and* $\log_b a > k$ *or* $a > b^k$

- $f\,(n)$ grows polynomialy slower than $n^{\log_b a}$ which means a major portion of execution is spent at the leaves.
- ***Solution:*** $T(n) = \Theta(n^{\log_b a})$.

Example-Case 1

- $T(n) = 4T\left(\frac{n}{2}\right) + n$
- $a = 4, b = 2, k = 1$
- $\log_b a > k$
- $T(n) = \theta(n^2)$
- Find the order if $T(n) = 3T\left(\frac{n}{2}\right) + \sqrt{n}$

Idea of master theorem

Recursion tree:

$$f(n) \qquad\qquad f(n)$$

$$a$$

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

$$a$$

$$h = \log_b n$$

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

$$\vdots$$

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

CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

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

Three common cases

2. $f(n) = \theta\left(n^k \log^p n\right), k = \log_b a\,, p \geq 0$

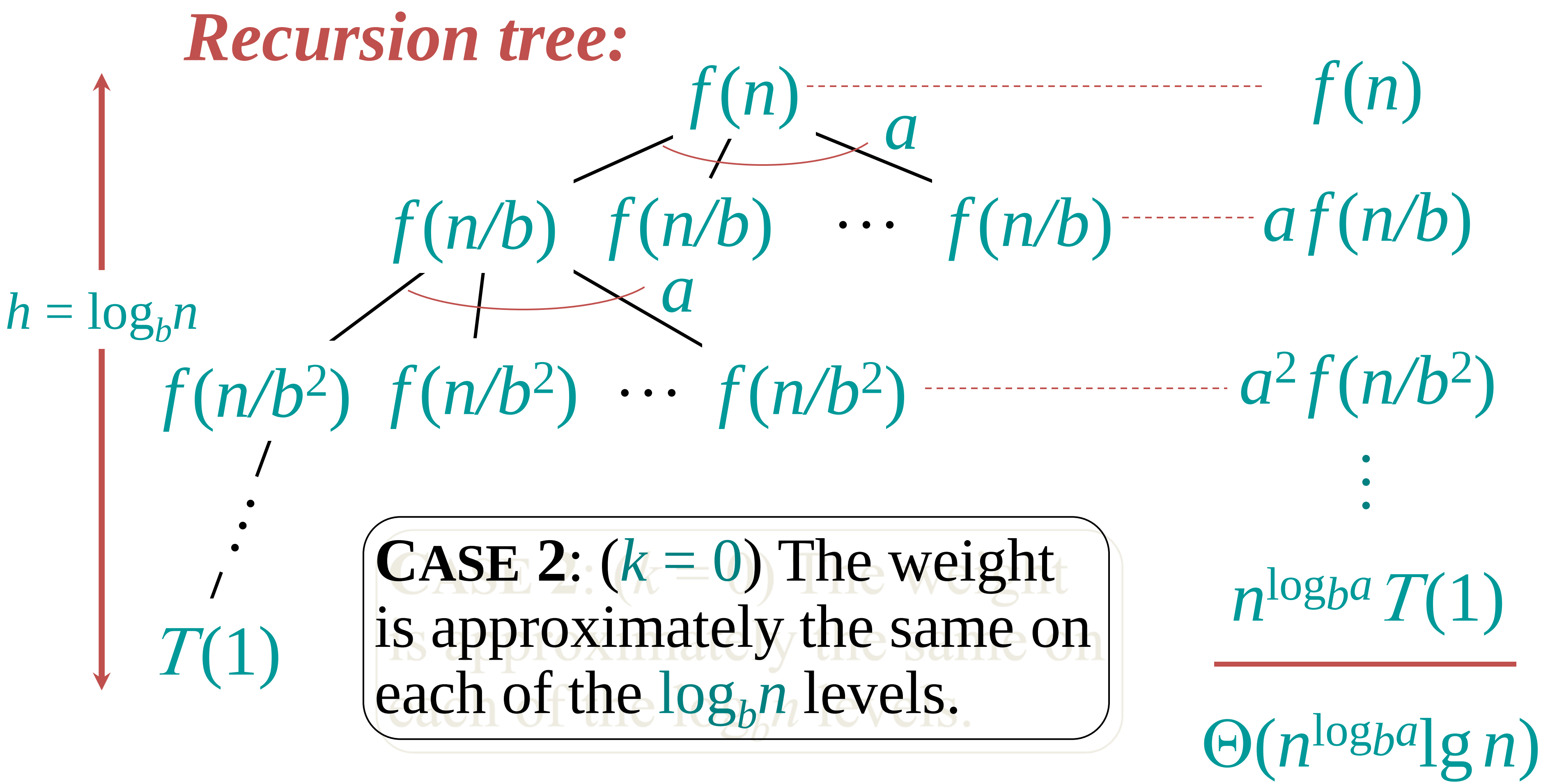
$f(n)$ and $n^{\log_b a}$ grow at similar rates.

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

Example-Case 2

- $T(n) = 4T\left(\frac{n}{2}\right) + n^2$
- $a = 4, b = 2, k = 2$
- $\log_b a = k, p = 0$
- $T(n) = \theta(n^2 \log n)$
- Find the order if $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$

Idea of master theorem



CASE 2: ($k = 0$) The weight is approximately the same on each of the $\log_b n$ levels.

Three common cases (cont.)

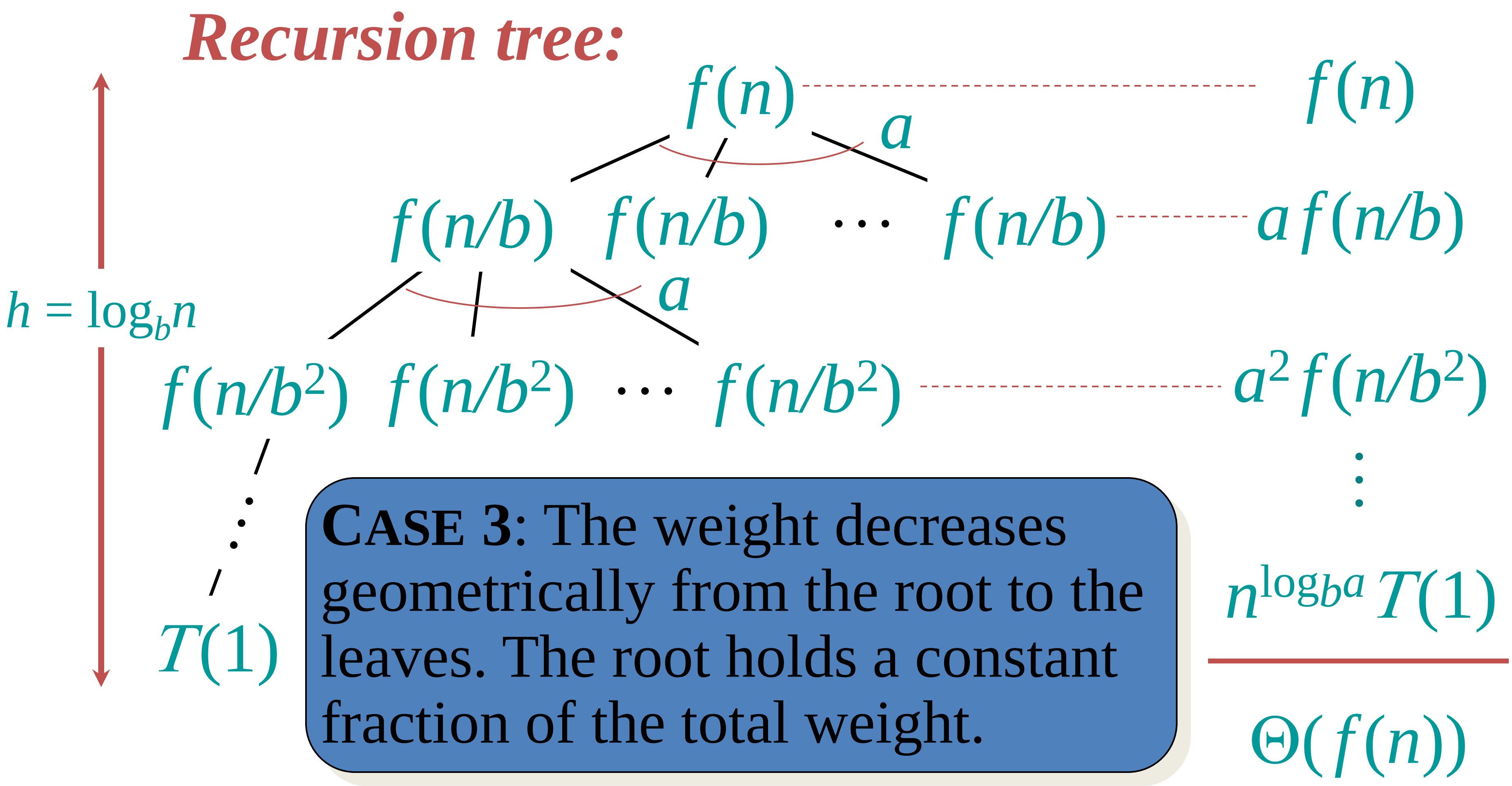
3. $f(n) = \theta\left(n^k\right), k > \log_b a$.
   - $f(n)$ grows polynomialy faster than $n^{\log_b a}$
   - ***and*** $f(n)$ satisfies the ***regularity condition*** that $a f(n/b) \le c f(n)$ for some constant $c < 1$.

Solution: $T(n) = \Theta(n^k)$.

Example-Case 3

- $T(n) = 3T\left(\frac{n}{2}\right) + n^2$
- $a = 3, b = 2, k = 2$
- $\log_b a > k$
- $T(n) = \theta(n^2)$
- Find the order if $T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n}$

Idea of master theorem



CASE 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

Inadmissible cases

- $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n$

  a is not a constant; the number of subproblems should be fixed

- $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

  non-polynomial difference between f(n) and $n^{\log_b a}$

$$\frac{f(n)}{n^{\log_b a}} = \frac{\frac{n}{\log n}}{n^{\log_2 2}} = \frac{1}{\log n}$$ which is not polynomial

- $T(n) = 0.5T\left(\frac{n}{2}\right) + n$

  a<1 cannot have less than one sub problem

Inadmissible cases

- $T(n) = 64T\left(\frac{n}{8}\right) - n^2 \log n$

- $T(n) = T\left(\frac{n}{2}\right) + n(2 - \cos n)$

  case 3 but regularity violation.