CSE 301 Combinatorial Optimization

Lecture 2
Recurrence

Today’s Topic

- Recurrence
  - Substitution Method
  - Recursive Method
  - Master Method
  - Akra-bazzi method

Recurrence Relations

- A ***recurrence relation*** is the recursive part of a *recursive definition* of either a number sequence or integer function.

Recursively Defined Sequences

- Fibonacci sequence:
  - $\{f_n\} = 0,1,1,2,3,5,8,13,21,34,55,\ldots$
  - Recursive definition for $\{f_n\}$:
  - INITIALIZE: $f_0 = 0, f_1 = 1$
  - RECURSE: $f_n = f_{n-1} + f_{n-2}$ for $n > 1$.
  - The recurrence relation is the recursive part
  - $f_n = f_{n-1} + f_{n-2}$. Thus a recurrence relation for a sequence consists of an equation that expresses each term in terms of lower terms.

Substitution method

The most general method:

1. ***Guess*** the form of the solution.
2. ***Verify*** by induction.
3. ***Solve*** for constants.

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

- [Assume that $T(1) = \Theta(1)$.]
- Guess $O(n \lg n)$.
- Assume that $T(k) \le ck \lg k$ for $k < n$

  .
- Prove $T(n) \le cn\, lg\, n$ by induction.

Example of substitution

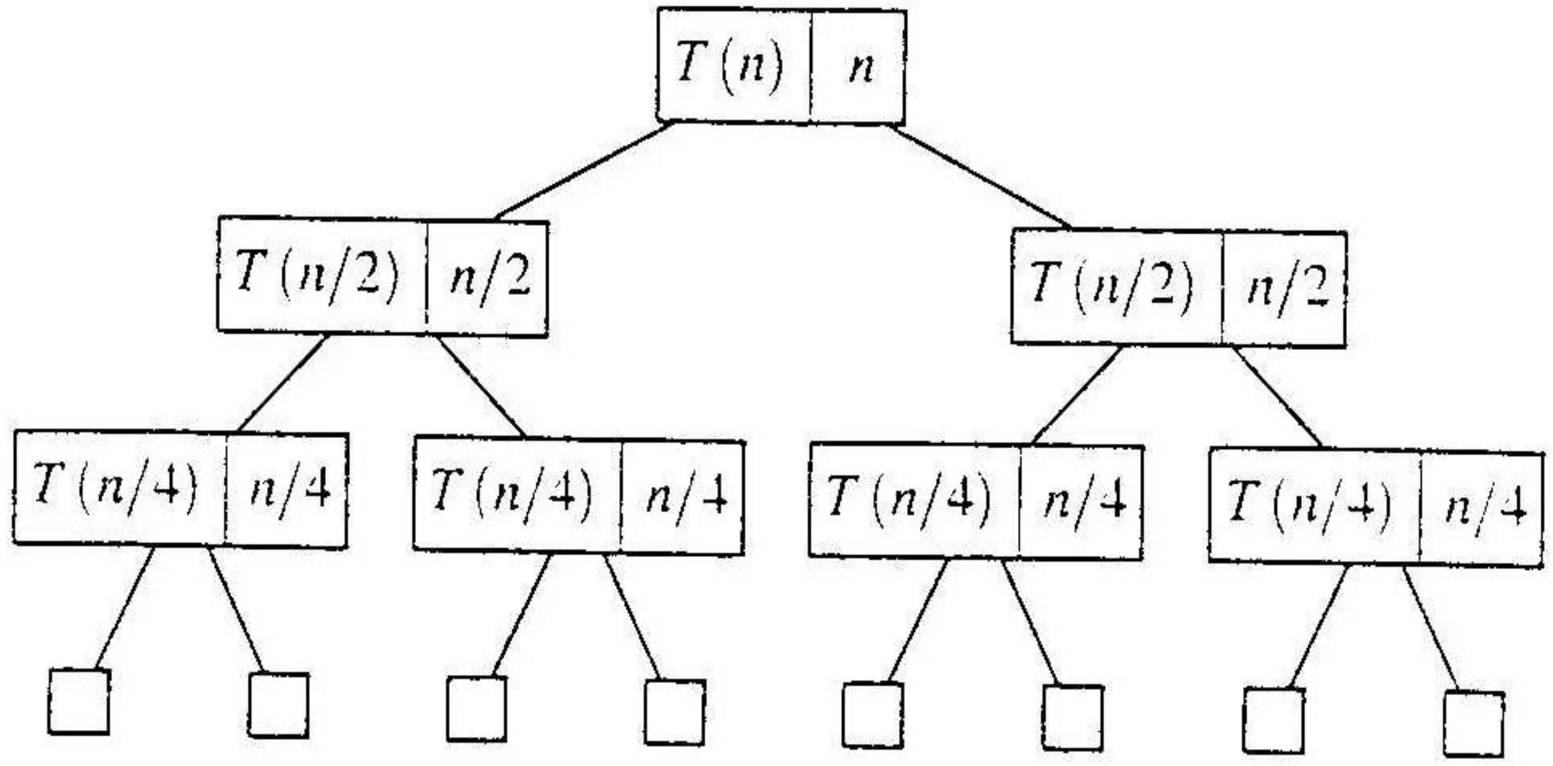
$$\begin{aligned}
T(n) &= 2T(n/2) + n \\
&\le 2(c(n/2)\ \lg(n/2)) + n \\
&\le cn\lg(n/2) + n \\
&= cn\lg n - cn\lg 2 + n \\
&= cn\lg n - (cn - n) &&\leftarrow \textit{desired} - \textit{residual} \\
&\le cn\lg n &&\leftarrow \textit{desired}
\end{aligned}$$

whenever $cn - n \ge 0$, for example, if $c \ge 2$ and $n \ge 1$.

$\nwarrow$ *residual*

Evaluate recursive equation using Recursion Tree

- Evaluate: $T(n) = T(n/2) + T(n/2) + n$
  - Work copy: $T(k) = T(k/2) + T(k/2) + k$
  - For $k=n/2$, $T(n/2) = T(n/4) + T(n/4) + (n/2)$
- [size|cost]



Recursion-tree method

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

Recursion Tree e.g.

- To evaluate the total cost of the recursion tree
  - sum all the non-recursive costs of all nodes
  - = Sum (rowSum(cost of all nodes at the same depth))
- Determine the maximum depth of the recursion tree:
  - For our example, at tree depth d
    the size parameter is $n/(2^d)$
  - the size parameter converging to base case, i.e. case 1
  - such that, $n/(2^d) = 1$,
  - $d = \lg(n)$
  - The rowSum for each row is n
- Therefore, the total cost, $T(n) = n \lg(n)$

Example of recursion tree

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

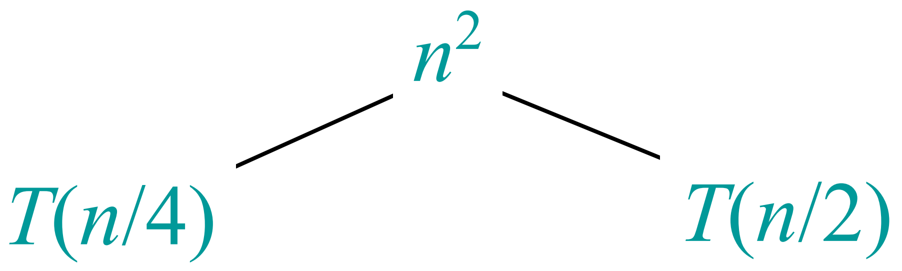
Example of recursion tree

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

$$T(n)$$

Example of recursion tree

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



Example of recursion tree

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

$$\begin{tikzcd}[column sep=small]
 & & & n^2 \arrow[dll, no head] \arrow[drr, no head] & & & \\
 & (n/4)^2 \arrow[dl, no head] \arrow[dr, no head] & & & & (n/2)^2 \arrow[dl, no head] \arrow[dr, no head] & \\
T(n/16) & & T(n/8) & & T(n/8) & & T(n/4)
\end{tikzcd}$$

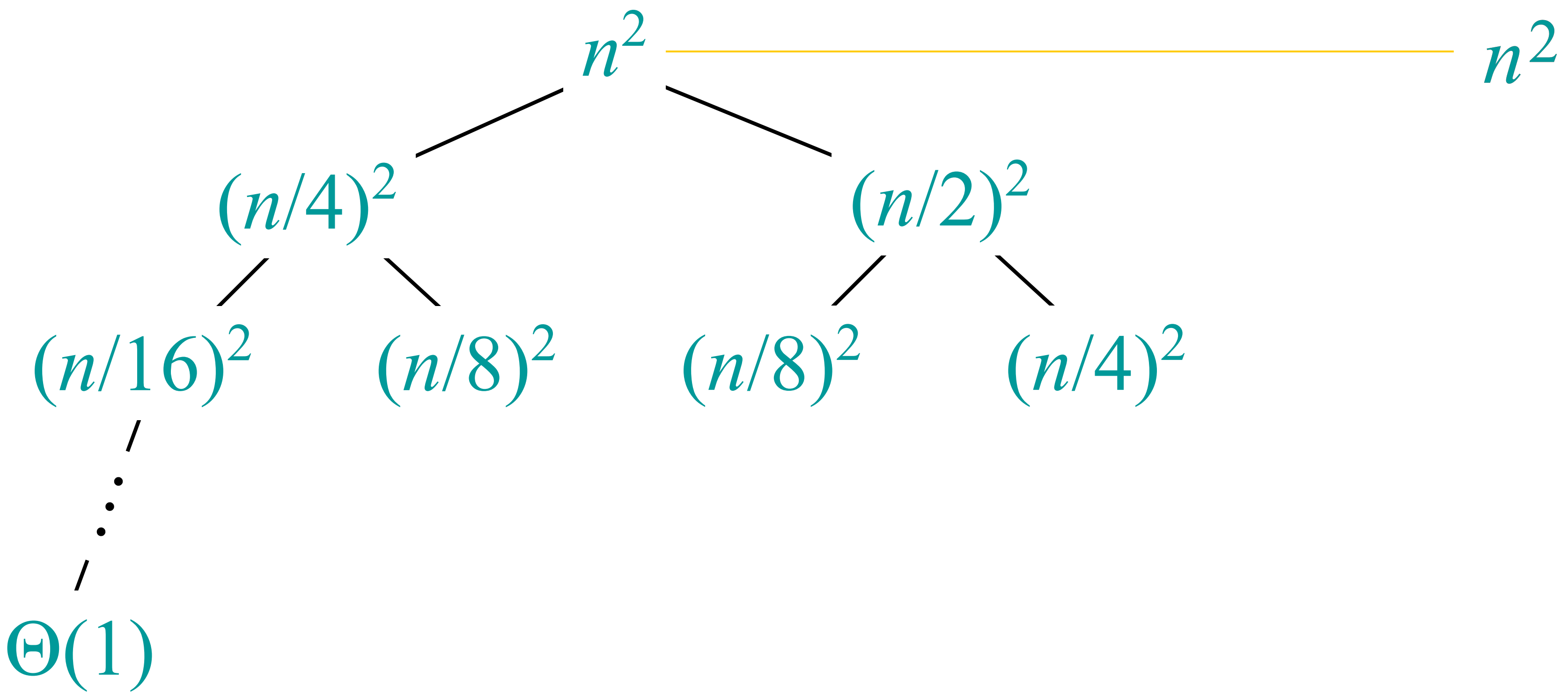
Example of recursion tree

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

$$\begin{tikzcd}
 & & & n^2 \arrow[dll, no head] \arrow[drr, no head] & & & \\
 & (n/4)^2 \arrow[dl, no head] \arrow[dr, no head] & & & & (n/2)^2 \arrow[dl, no head] \arrow[dr, no head] & \\
(n/16)^2 \arrow[d, no head] & & (n/8)^2 & & (n/8)^2 & & (n/4)^2 \\
\vdots \arrow[d, no head] & & & & & & \\
\Theta(1) & & & & & &
\end{tikzcd}$$

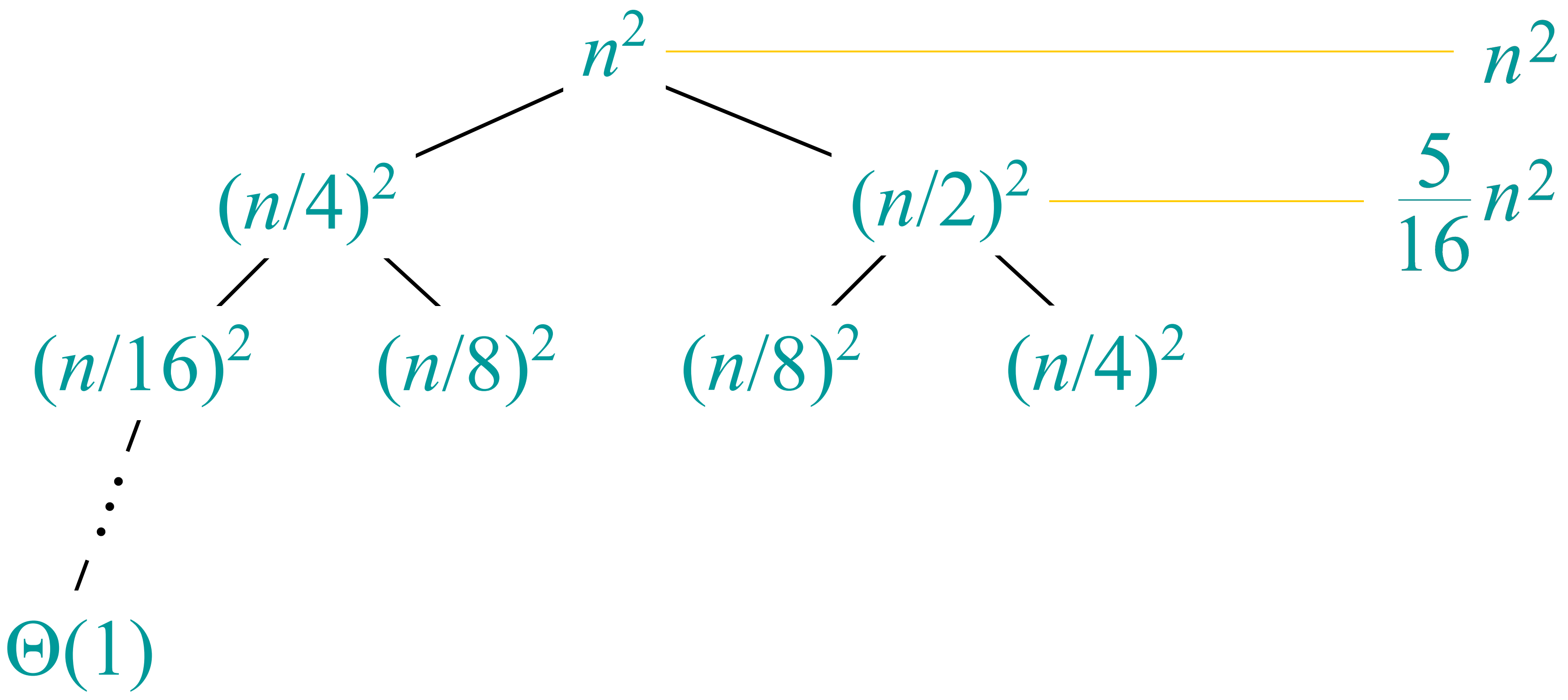
Example of recursion tree

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



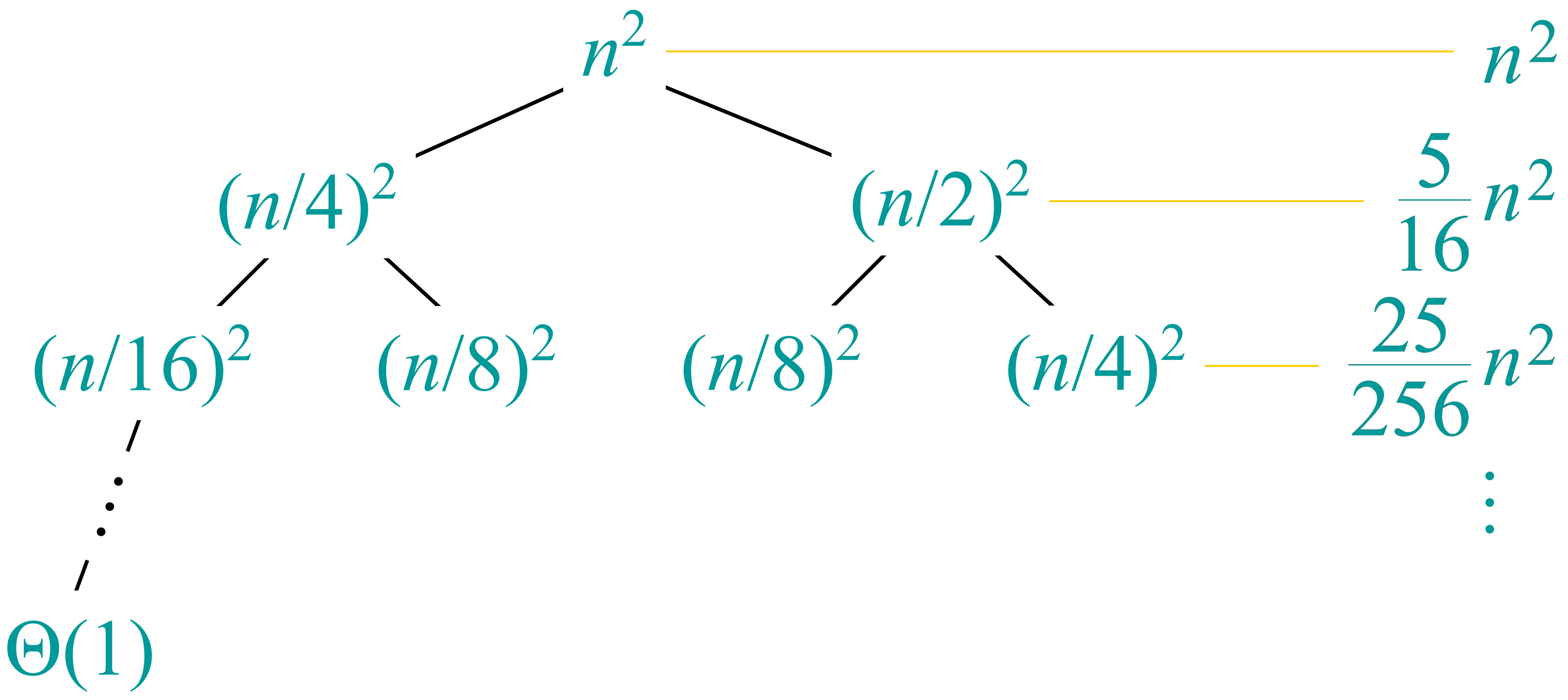
Example of recursion tree

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



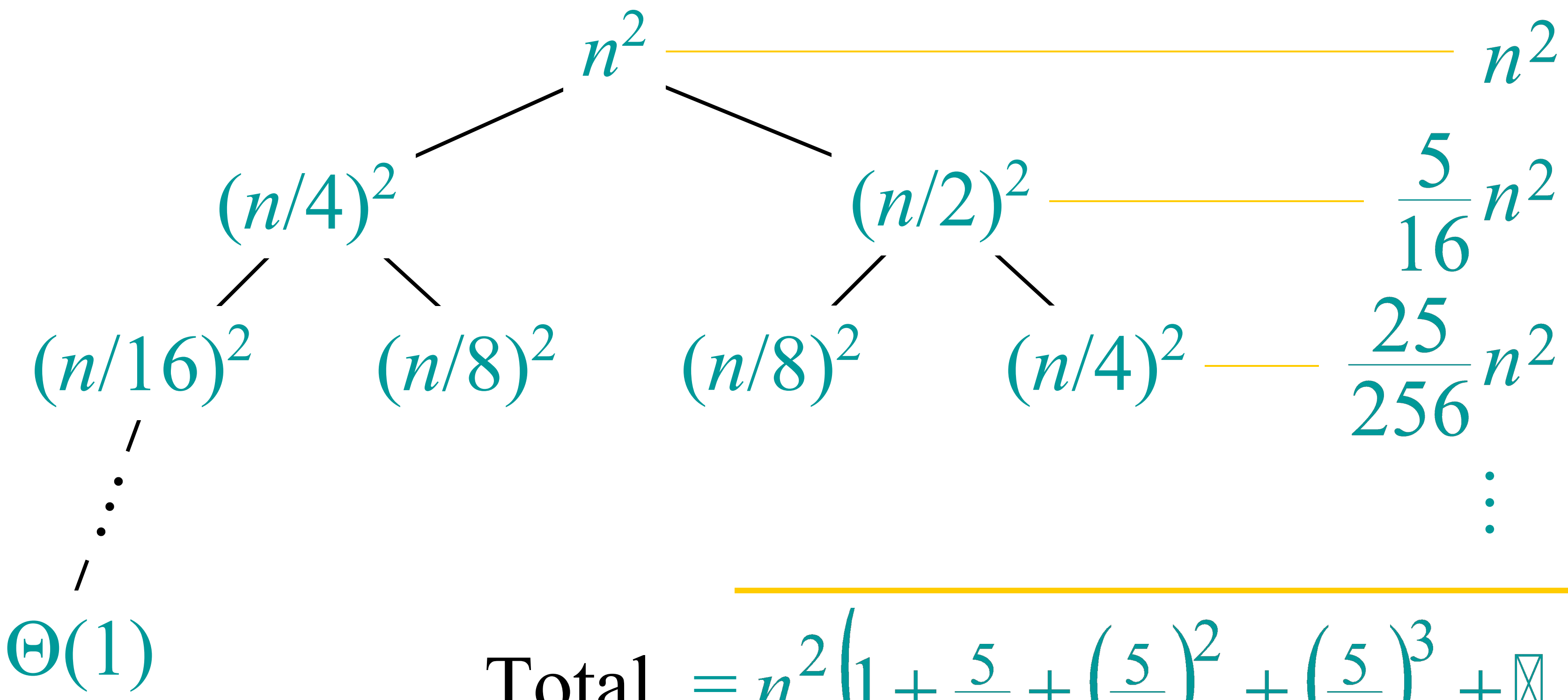
Example of recursion tree

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



Example of recursion tree

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



$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$$
$$= \Theta(n^2)$$

geometric series

The divide-and-conquer design paradigm

1. *Divide* the problem (instance) into subproblems.
2. *Conquer* the subproblems by solving them recursively.
3. *Combine* subproblem solutions.

Example: merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

$$T(n) = \underbrace{2}_{\#\ \textit{subproblems}}\, T(\underbrace{n/2}_{\textit{subproblem size}}) + \underbrace{O(n)}_{\substack{\textit{work dividing}\\ \textit{and combining}}}$$

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.

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 $n^{\log_b a}$ (by an n^{ε} factor).

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

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

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

Three common cases (cont.)

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 $n^{\log_b a}$ (by an n^{ε} factor),

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

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

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

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

Case 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

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

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

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

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

Master method does not apply. In particular, for every constant $\varepsilon > 0$, we have $n^{\varepsilon} = \omega(\lg n)$.

General method (Akra-Bazzi)

- The Master method is fairly powerful and results in a closed form solution for divide-and-conquer recurrences with a special form.
- Akra and Bazzi discovered a far more general solution to divide-and-conquer recurrences.

The Akra-Bazzi Method

$$T(x) = \begin{cases} \Theta(1) & \text{for } 1 \le x \le x_0 \\ \sum_{i=1}^{k} a_i T(b_i x) + g(x) & \text{for } x > x_0 \end{cases} \tag{1}$$

where[1]

1. $x \geq 1$ is a real number,
2. x_0 is a constant such that $x_0 \geq 1/b_i$ and $x_0 \geq 1/(1 - b_i)$ for $1 \leq i \leq k$,
3. $a_i > 0$ is a constant for $1 \leq i \leq k$,
4. $b_i \in (0, 1)$ is a constant for $1 \leq i \leq k$,
5. $k \geq 1$ is a constant, and
6. $g(x)$ is a nonnegative function that satisfies the polynomial-growth condition specified below.

Definition. We say that $g(x)$ satisfies the *polynomial-growth condition* if there exist positive constants c_1, c_2 such that for all $x \geq 1$, for all $1 \leq i \leq k$, and for all $u \in [b_i x, x]$,

$$c_1 g(x) \leq g(u) \leq c_2 g(x).$$

The Akra-Bazzi Solution

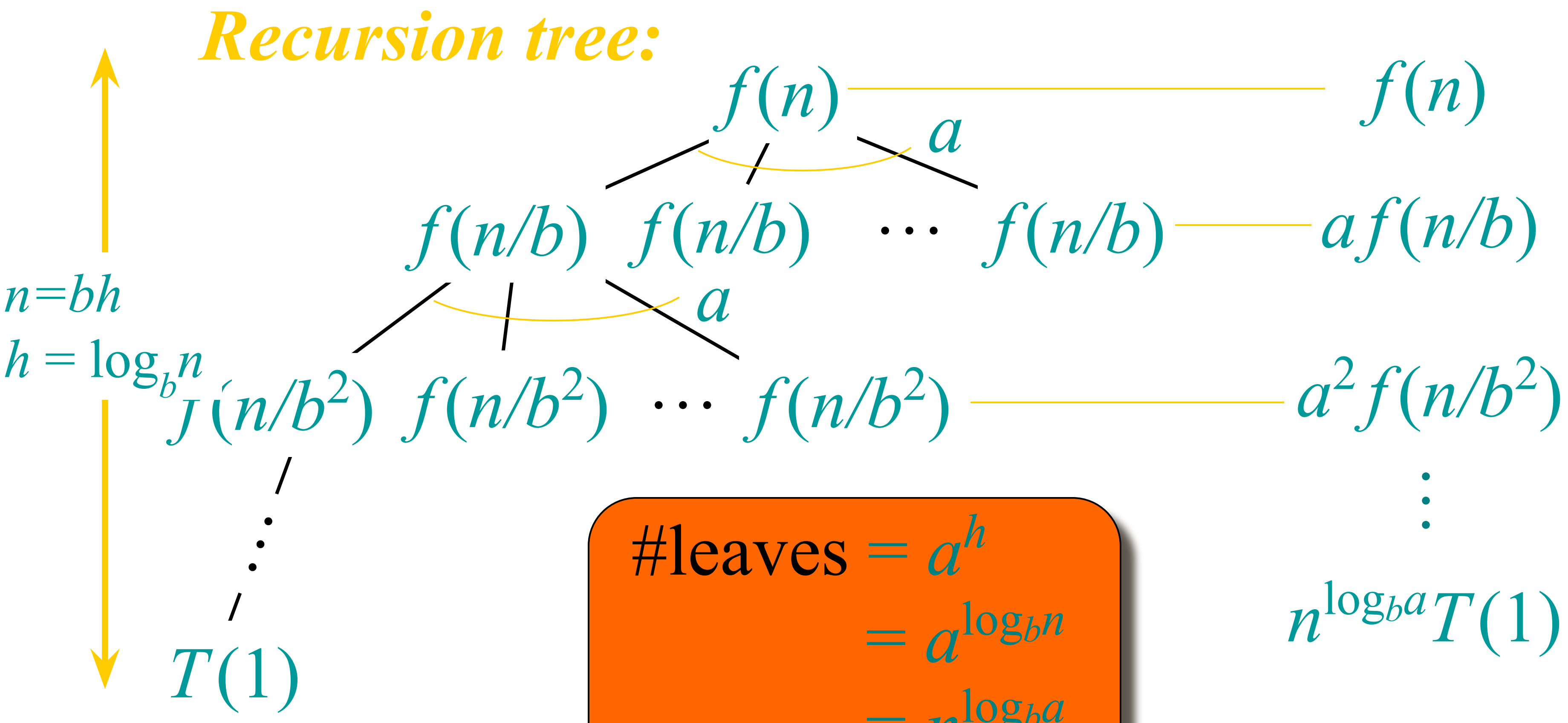
Theorem 1 ([1]). *Given a recurrence of the form specified in Equation 1, let p be the unique real number for which $\sum_{i=1}^{k} a_i b_i^p = 1$. Then*

$$T(x) = \Theta\left(x^p\left(1 + \int_1^x \frac{g(u)}{u^{p+1}}\,du\right)\right).$$

Examples.

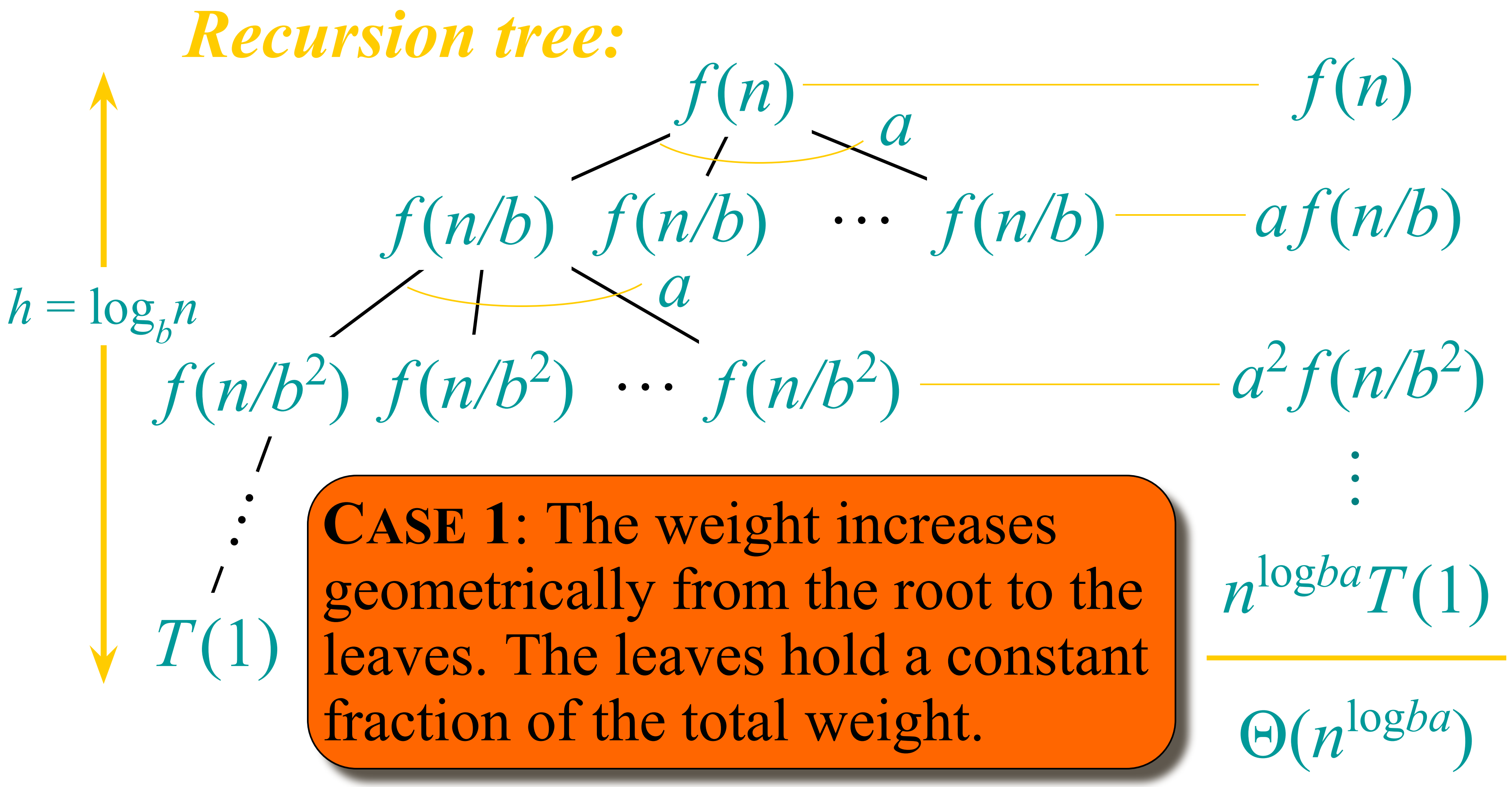
- If $T(x) = 2T(x/4) + 3T(x/6) + \Theta(x \log x)$, then $p = 1$ and $T(x) = \Theta(x \log^2 x)$.
- If $T(x) = 2T(x/2) + \frac{8}{9}T(3x/4) + \Theta(x^2/\log x)$, then $p = 2$ and $T(x) = \Theta(x^2/\log\log x)$.
- If $T(x) = T(x/2) + \Theta(\log x)$, then $p = 0$ and $T(x) = \Theta(\log^2 x)$.
- If $T(x) = \frac{1}{2}T(x/2) + \Theta(1/x)$, then $p = -1$ and $T(x) = \Theta((\log x)/x)$.
- If $T(x) = 4T(x/2) + \Theta(x)$, then $p = 2$ and $T(x) = \Theta(x^2)$.

Idea of master theorem



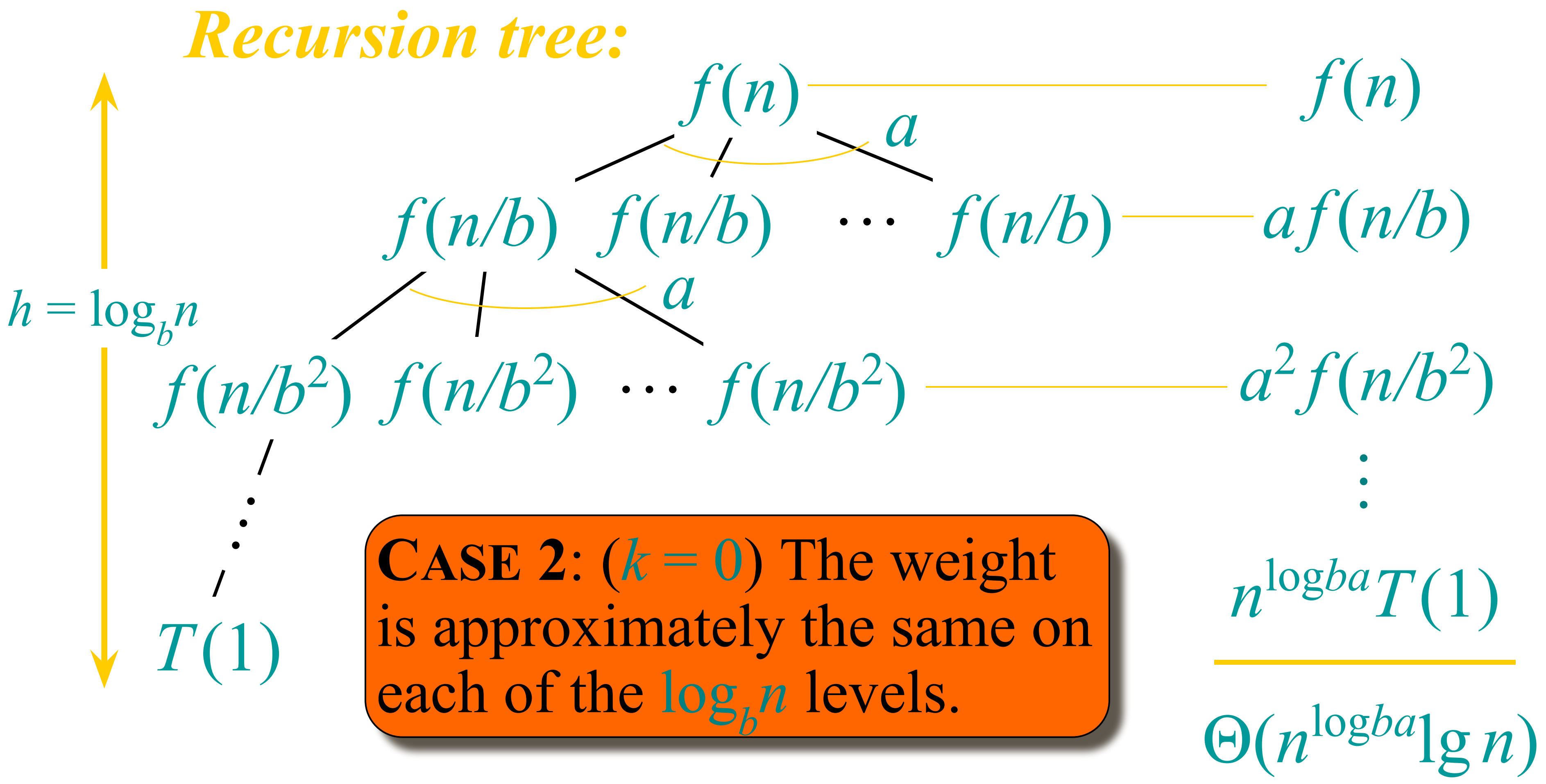
$$\#\text{leaves} = a^h = a^{\log_b n} = n^{\log_b a}$$

Idea of master theorem



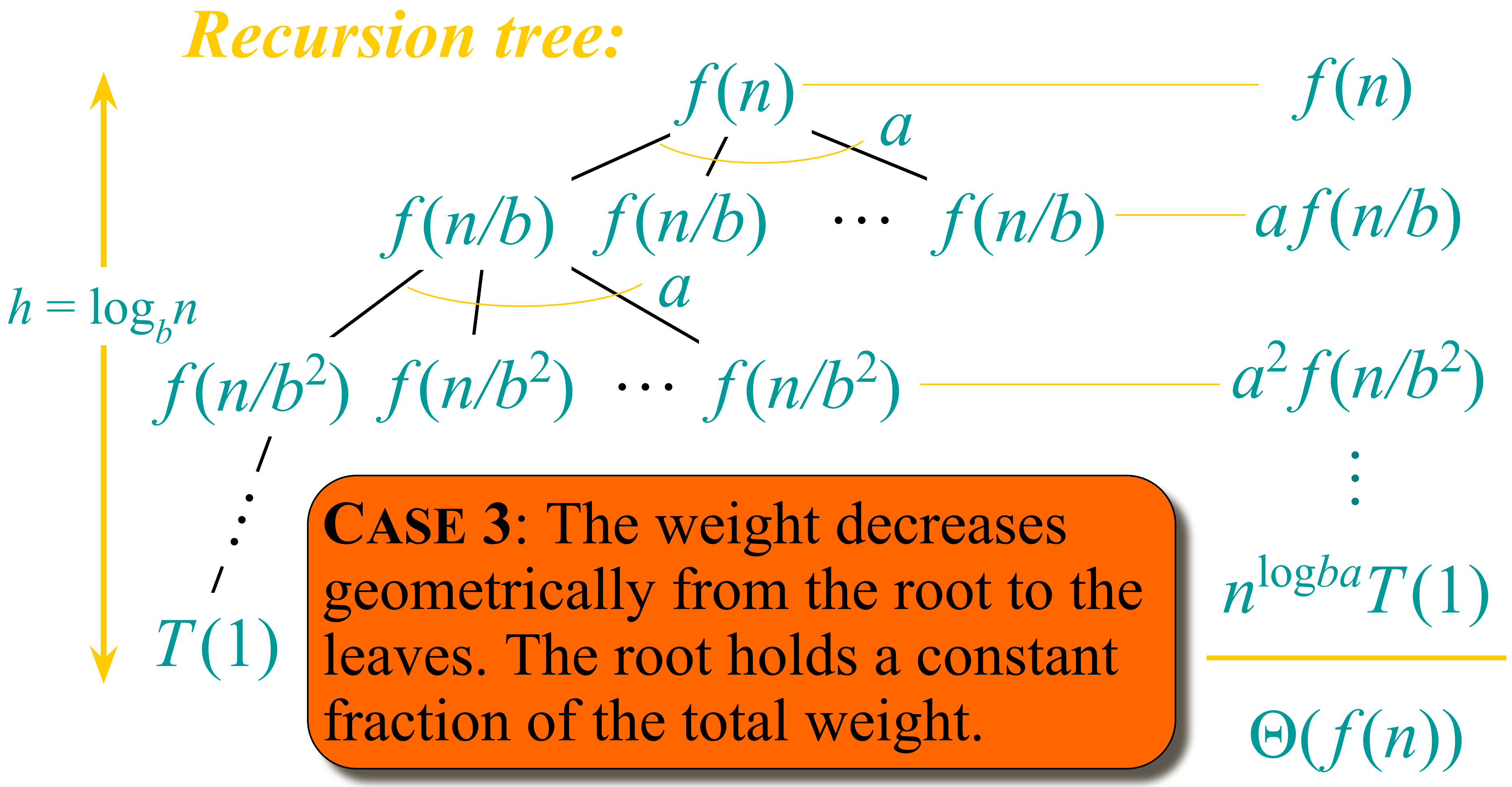
CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

Idea of master theorem


Recursion tree:
$f(n)$
a
$f(n/b)$ $f(n/b)$ $\cdots$ $f(n/b)$
a
$f(n/b^2)$ $f(n/b^2)$ $\cdots$ $f(n/b^2)$
$h = \log_b n$
$T(1)$
$f(n)$
$af(n/b)$
$a^2 f(n/b^2)$
$\vdots$
$n^{\log_b a} T(1)$
$\Theta(n^{\log_b a} \lg n)$

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

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.