Design and Analysis of Algorithms

Recurrences

Reference:
CLRS Chapter 4

Topics:

- **Substitution method**
- **Recursion-tree method**
- **Master method**

Solving recurrences

- The analysis of Mergesort from Lecture 2 required us to solve a recurrence.
- Recurrences are a major tool for analysis of algorithms
  - Today: Learn a few methods.
    - » Substitution method
    - » Recursion- tree method
    - » Master method

- Divide and Conquer algorithms which are analyzable by recurrences.

Recall: Mergesort

MERGESORT

```
MERGE-SORT(A,p,r)
if  p < r
    then  q ← ⌊(p+r)/2⌋
          MERGE-SORT (A, p, q)
          MERGE-SORT (A, q+1, r)
          MERGE (A, p, q, r)
```

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

Substitution method

- **The most general method:**
  - **Guess** the form of the solution.
  - **Verify** by induction.
  - **Solve** for constants.

- **Ex.** $T(n) = 4T(n/2) + 100n$
  - Assume that $T(1) = \Theta(1)$.
  - Guess $O(n^3)$. (Prove O and Ω separately.)
  - Assume that $T(k) \leq ck^3$ for $k < n$.
  - Prove $T(n) \leq cn^3$ by induction.

Example of substitution

- $T(n) = 4T(n/2) + 100n$

  $\leq 4c(n/2)^3 + 100n$

  $= (c/2)\, n^3 + 100n$

  $= cn^3 - ((c/2)\, n^3 - 100n)$ ⟵ *desired – residual*

  $\leq cn^3$ ⟵ *desired*

- **whenever $(c/2)n^3 - 100n \geq 0$, for example, if $c \geq 200$ and $n \geq 1$.**

  ↖ *residual*

Example (continued)

- We must also handle the **initial conditions/the boundary conditions**, that is, ground the induction with base cases.
- Base: $T(n) = \Theta(1)$ for all $n < n_0$, where n_0 is a suitable constant.
- For $1 \le n < n_0$, we have "$\Theta(1)$" $\le cn^3$, if we pick c big enough.

- This bound is not tight!

A tighter upper bound?

- We shall prove that $T(n) = O(n^2)$.
- Assume that $T(k) \le ck^2$ for $k < n$:

Making a good guess

$$T(n) = 4T(n/2) + 100n$$

$$\le cn^2 + 100n$$

-
- Which does not imply $T(n) \le cn^2$ for any choice of c.

A tighter upper bound?

subtleties

- IDEA: Strengthen the induction hypothesis.
  - Subtract a low-order term.
- Assume that $T(k) \le c_1k^2 - c_2k$ for $k < n$.
- $T(n) = 4T(n/2) + 100n$

$$\le 4(c_1(n/2)^2 - c_2(n/2)) + 100n$$
$$= c_1n^2 - 2c_2n + 100n$$
$$= c_1n^2 - c_2n - (c_2n - 100n)$$
$$\le c_1n^2 - c_2n$$

- The last step holds as long as $c_2 > 100$.
- Pick c_1 big enough to handle the initial conditions.

Avoiding Pitfalls

- Be careful not to misuse asymptotic notation. For example:
  - We can **falsely** prove $T(n) = O(n)$ by guessing $T(n) \le cn$ for $T(n) = 2T(\lfloor n/2 \rfloor) + n$

$$T(n) \le 2c\lfloor n/2 \rfloor + n$$
$$\le cn + n$$
$$= O(n) \Leftarrow \text{Wrong!}$$

  - The error is that we haven't proved the **exact form** of the inductive hypothesis $T(n) \le cn$.

Changing Variables

- Use algebraic manipulation to make an unknown recurrence similar to what you have seen before.
  - Consider $T(n) = 2T(\lfloor n^{1/2} \rfloor) + \lg n$,
  - Rename $m = \lg n$ and we have
    $T(2^m) = 2T(2^{m/2}) + m$.
  - Set $S(m) = T(2^m)$ and we have
    $S(m) = 2S(m/2) + m \Rightarrow S(m) = O(m \lg m)$.
  - Changing back from $S(m)$ to $T(n)$, we have
    $T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$.

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, just like any method that uses ellipses (…).**
- **The recursion-tree method promotes intuition, however.**

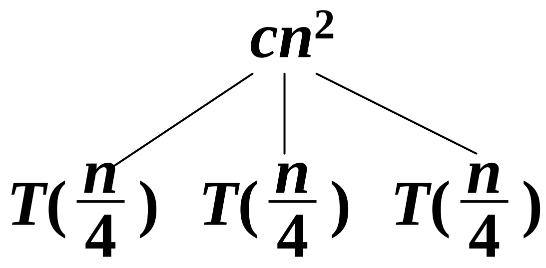
The Construction of a Recursion Tree

- **Solve $T(n) = 3T(n/4) + \Theta(n^2)$, we have**

$$T(n)$$

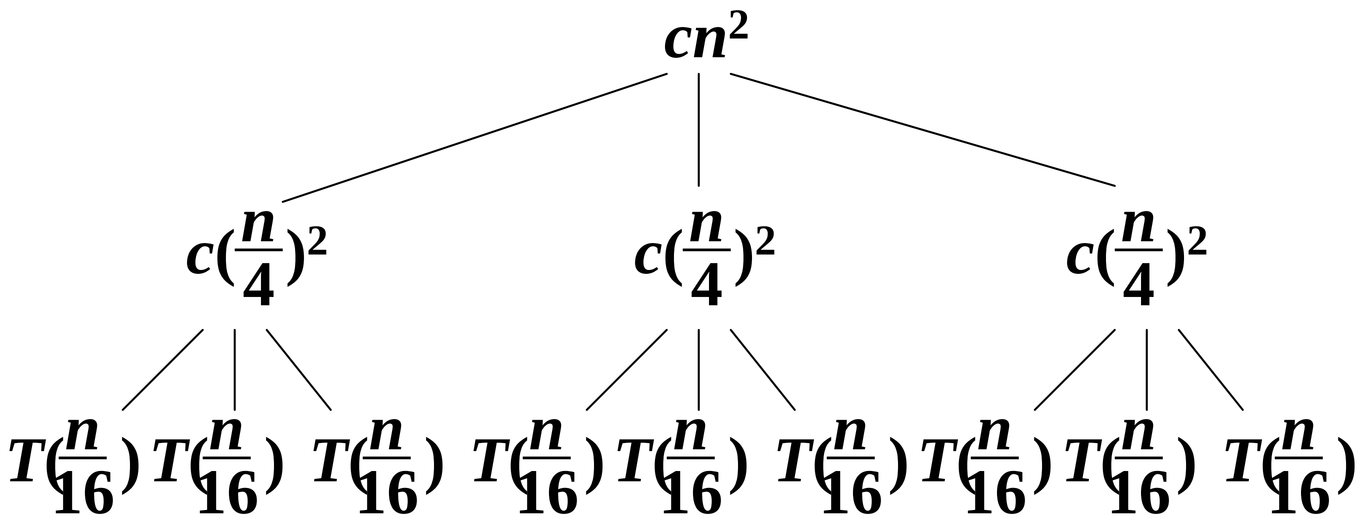
The Construction of a Recursion Tree

- **Solve $T(n) = 3T(n/4) + \Theta(n^2)$, we have**

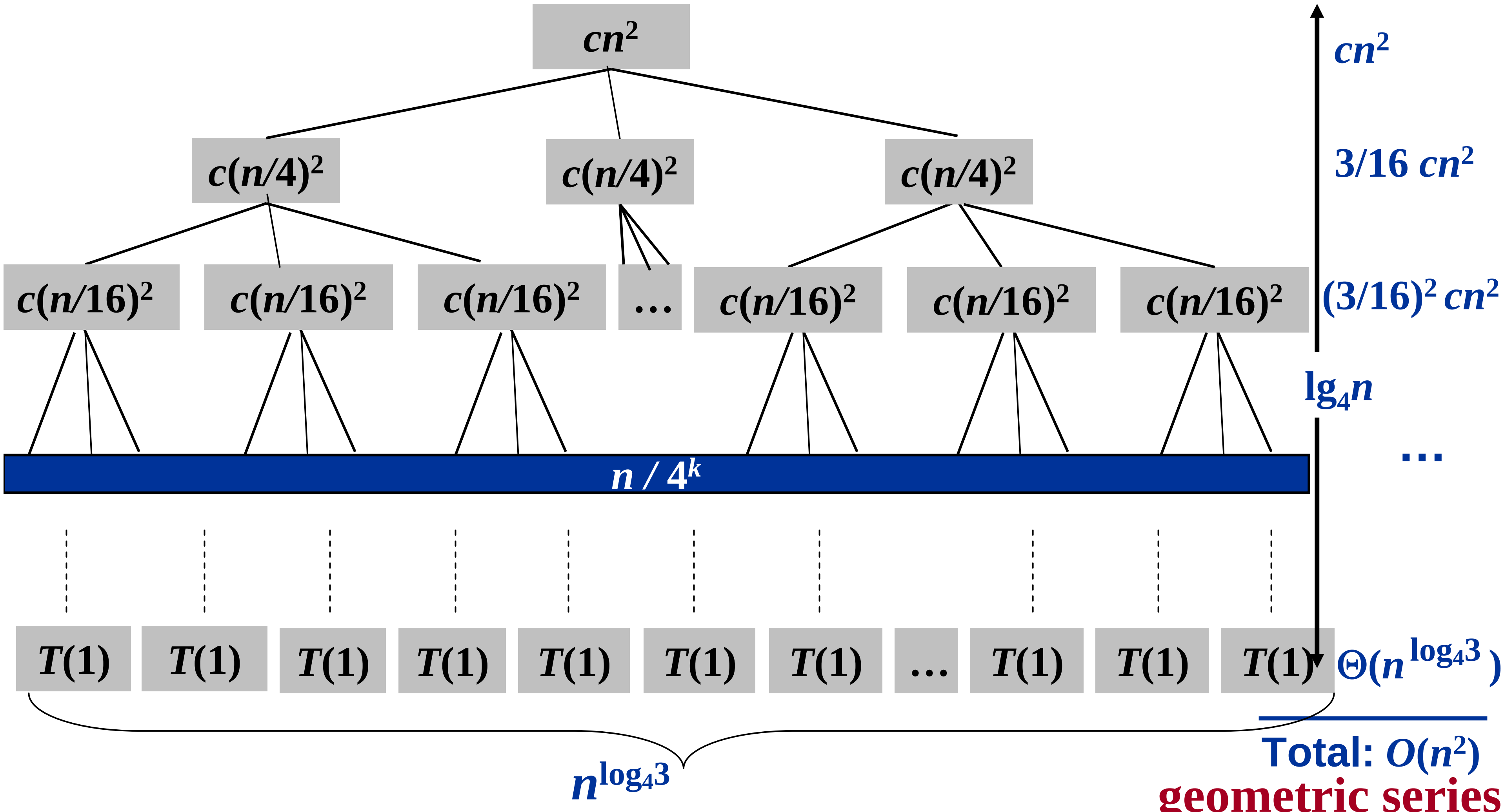


The Construction of a Recursion Tree

- **Solve $T(n) = 3T(n/4) + \Theta(n^2)$, we have**



Construction of recursion tree



The fully expanded tree has $\lg_4 n + 1$ levels, i.e., it has height $\lg_4 n$.

Master Method

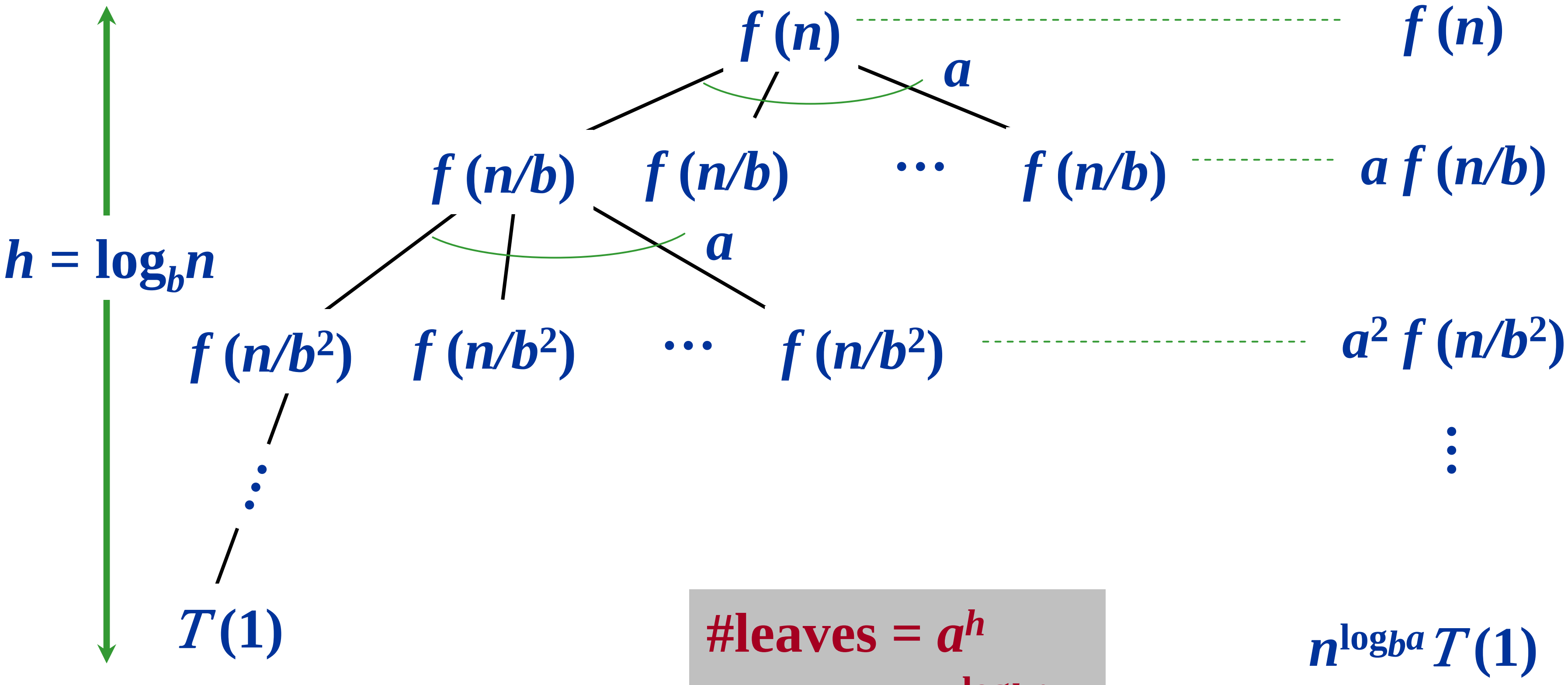
- It provides a "cookbook" method for solving recurrences of the form:

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

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function

Idea of master theorem

- **Recursion tree**



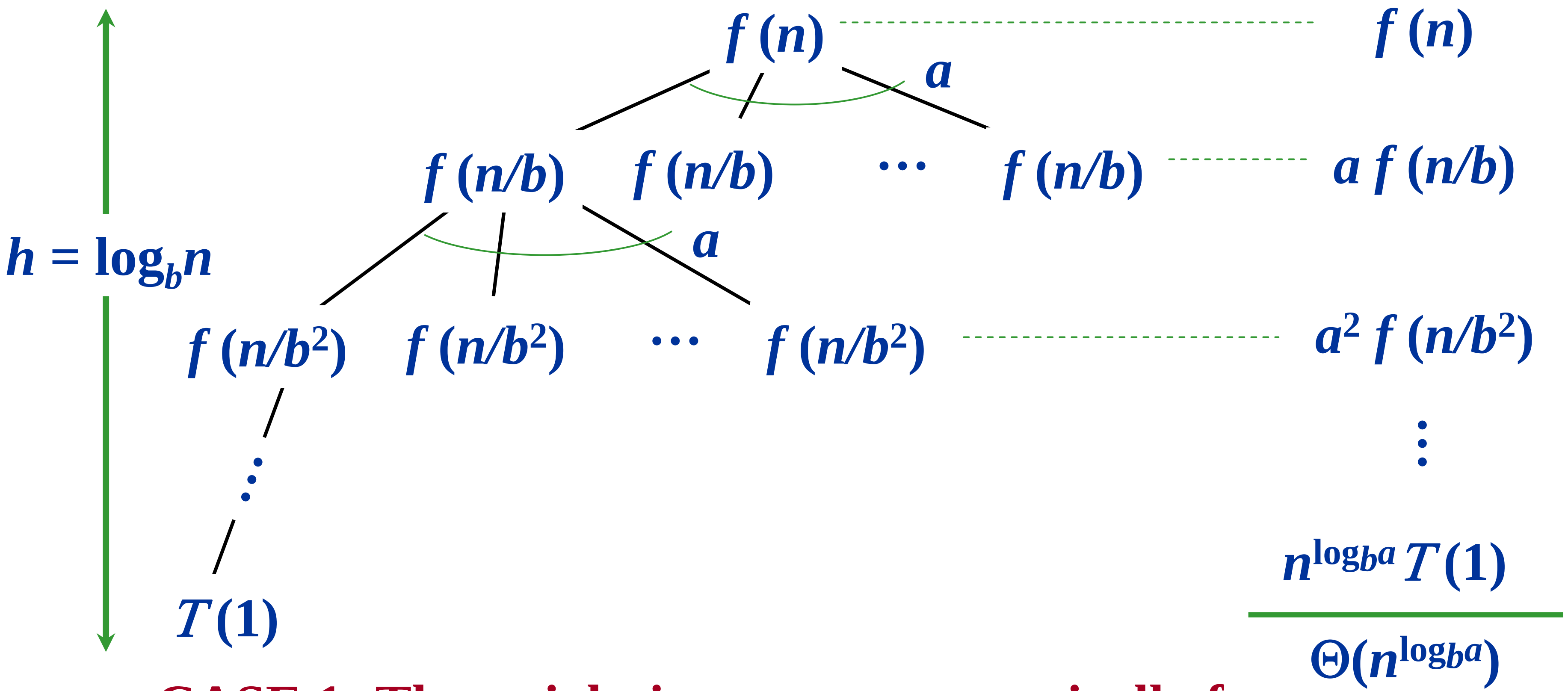
$$\#\text{leaves} = a^h = a^{\log_b n} = n^{\log_b a}$$

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

Idea of master theorem

- Recursion tree



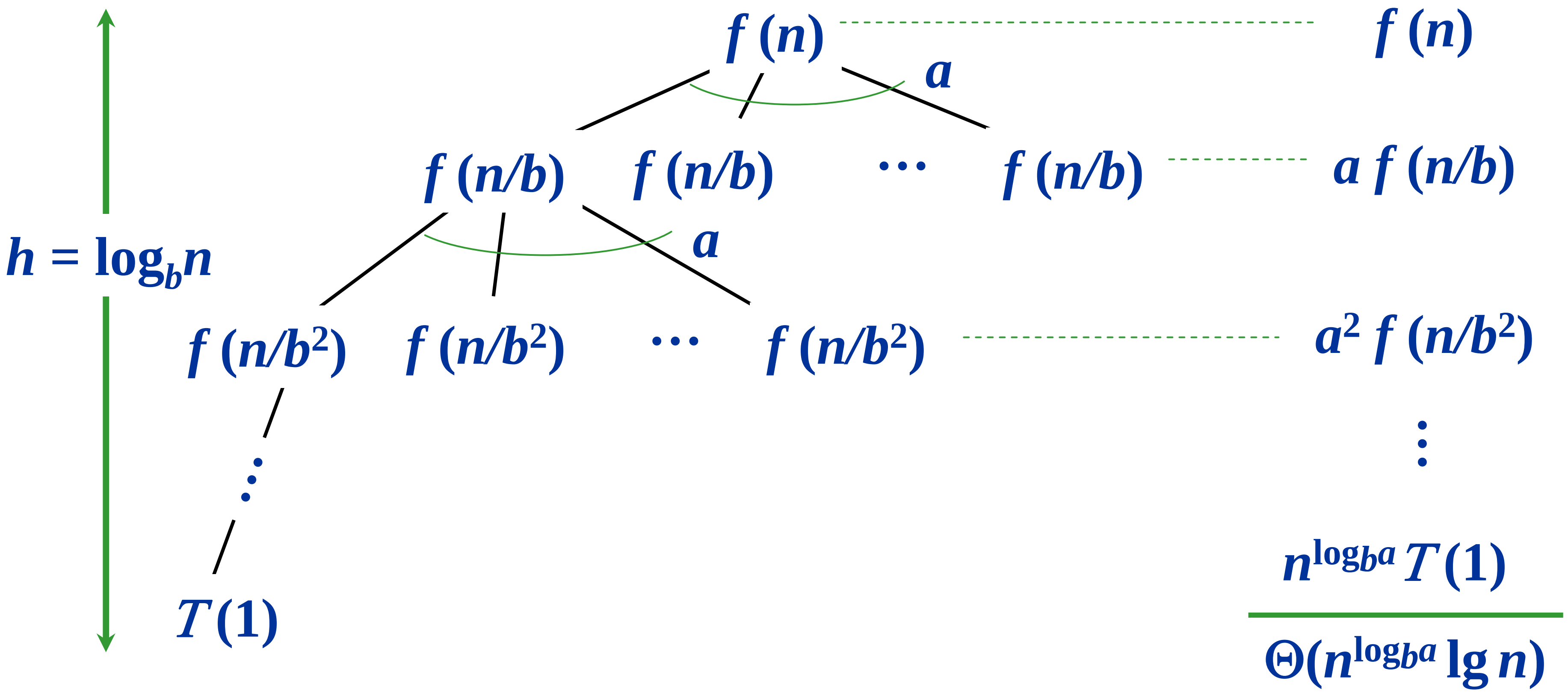
CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

Three common cases

- Compare $f(n)$ with $n^{\log_b a}$:
  - 2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.
    - » $f(n)$ and $n^{\log_b a}$ grow at similar rates.
  - Solution: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

Idea of master theorem

- Recursion tree



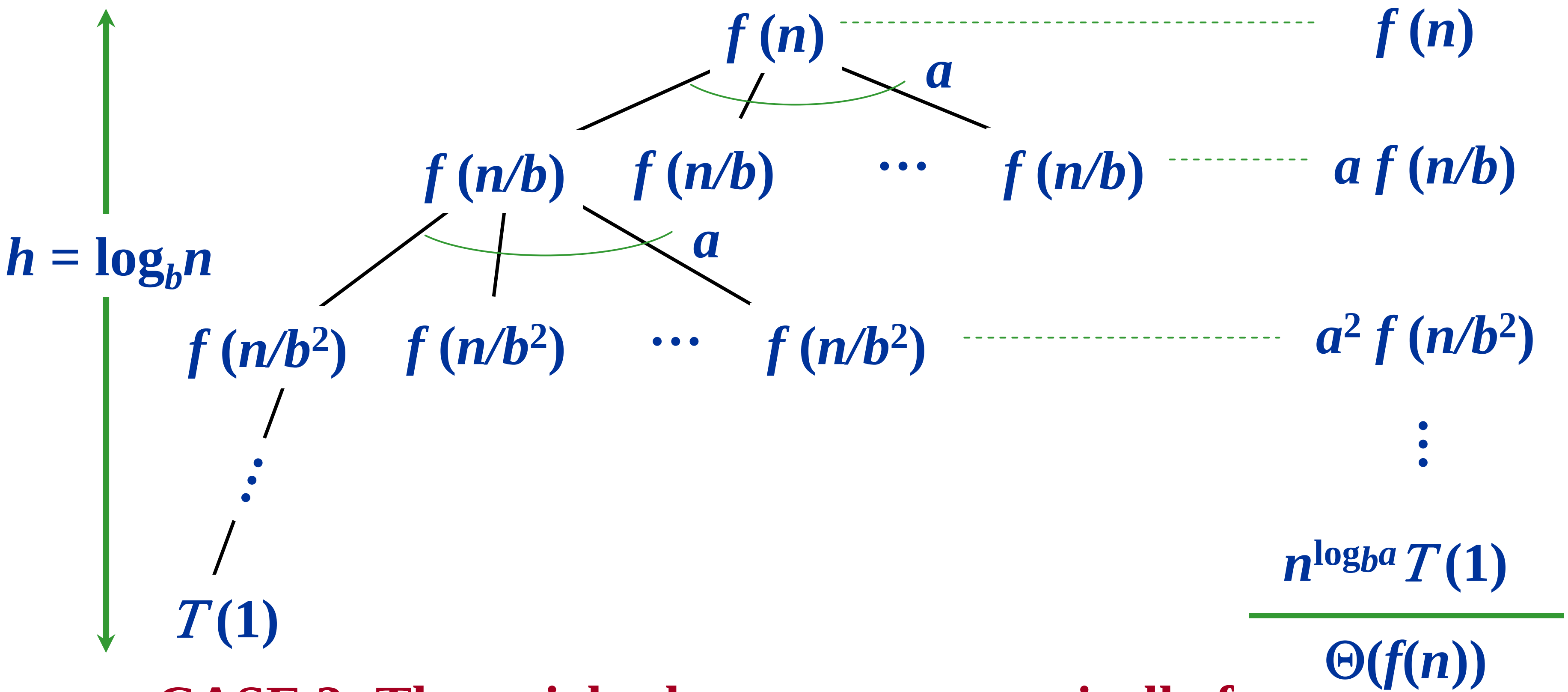
CASE 2: (k = 0)The weight is approximately the same on each of the $\log_b n$ levels.

Three common cases

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

Idea of master theorem

- **Recursion tree**



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

Examples

- $T(n) = 4T(n/2) + n$
  - $a = 4$, $b = 2$, $\Rightarrow n^{\log_b a} = n^2$; $f(n) = n$.
  - CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.
  - $\therefore T(n) = \Theta(n^2)$

- $T(n) = 4T(n/2) + n^2$
  - $a = 4$, $b = 2$, $\Rightarrow n^{\log_b a} = n^2$; $f(n) = n^2$.
  - CASE 2: $f(n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.
  - $\therefore T(n) = \Theta(n^2 \lg n)$

Examples

- $T(n) = 4T(n/2) + n^3$
  - $a = 4$, $b = 2$, $\Rightarrow n^{\log_b a} = n^2$; $f(n) = n^3$.
  - CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$, for $\varepsilon = 1$ and $4(cn/2)^3 \le cn^3$ (regular cond.) for $c = 1/2$.
  - $\therefore\ T(n) = \Theta(n^3)$

- $T(n) = 4T(n/2) + n^2/\lg n$
  - $a = 4$, $b = 2$, $\Rightarrow n^{\log_b a} = n^2$; $f(n) = n^3/\lg n$.
  - Master method does not apply. In particular, for every constant $\varepsilon > 0$, we have $n^{\varepsilon} = \omega(\lg n)$.