Asymptotic Notation

- O-, Ω-, and Θ- notation
- Recursion tree
- Master method

O-notation (upper bounds):

We write $f(n) = O(g(n))$

if there exist constants $c>0$, $n_0 >0$ such that

$0 \leq f(n) \leq cg(n)$

for all $n \geq n_0$

EXAMPLE: $2n^2 = O(n^3)$ $(c = 1, n_0 = 2)$

functions, not values

funny, "one-way" equality

Set definition of O-notation

$O(g(n))$ = { $f(n)$: there exist constants $c>0$, $n_0>0$ such that

$$0 \le f(n) \le cg(n)$$

for all $n \ge n_0$}

Convention: A set in a formula represents an anonymous function in the set.

EXAMPLE: $n^2 + O(n) = O(n^2)$

means

for any $f(n) \in O(n)$:

$n^2 + f(n) = h(n)$

for some $h(n) \in O(n^2)$.

Ω-notation (lower bounds)

O-notation is an upper-bound notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

$\Omega(g(n)) = \{ f(n)$: there exist constants $c>0$, $n_0>0$ such that

$$0 \le cg(n) \le f(n)$$

for all $n \ge n_0\}$

EXAMPLE: $\sqrt{n} = \Omega(\lg n) \quad (c = 1, n_0 = 16)$

Θ-notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n such that

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n) \qquad \text{for all } n \ge n_0$$

EXAMPLE: $\frac{1}{2}n^2 - 2n = \Theta(n^2)$

o-notation and ω-notation

O-notation and Ω-notation are like ≤ and ≥.

o-notation and ω-notation are like < and >.

$o(g(n)) = \{ f(n)$: for any constant $c > 0$, $n_0 > 0$ such that $0 \le f(n) < cg(n)$ for all $n \ge n_0\}$

EXAMPLE: $2n^2 = o(n^3)$ $(n_0 = 2/c)$

$\omega(g(n))$ = { $f(n)$: for any constant $c > 0$, $n_0 > 0$ such that $0 \leq cg(n) < f(n)$ for all $n \geq n_0$}

EXAMPLE: $\sqrt{n} = \omega(\lg n)$ $(n_0 = 1+1/c)$

Solving recurrences

- The analysis of merge sort required us to solve a recurrence.

- Recurrences are like solving integrals, differential equations, etc. Need to learn a few techniques.

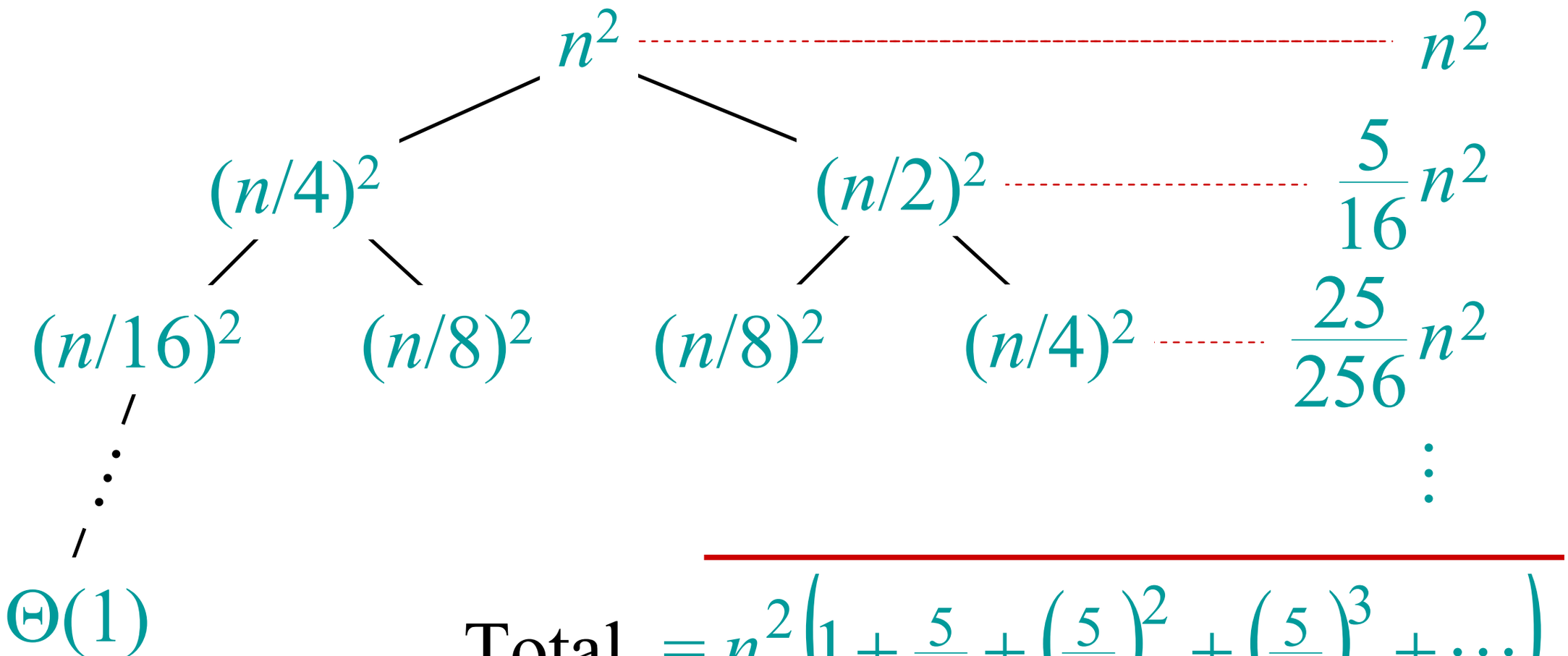
- Applications of recurrences to divide-and-conquer algorithms.

Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (… in a sentence).
- The recursion-tree method promotes intuition, however.
- The recursion tree method is good for generating guesses for the substitution method.

Example of recursion tree

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



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

$$= \Theta(n^2) \qquad \textit{geometric series}$$

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 typical 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. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

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

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

T(n)=2 T(n/2)+ n Lg(n)

a=2, b=2

Log_b(a)=lg(2)=1

is f(n) = Theta(n lg^k(n)) ?

f(n) = n lg(n)

yes with k=1

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: $f(n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.

$\therefore\ T(n) = \Theta(n^2 \lg n).$

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

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

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

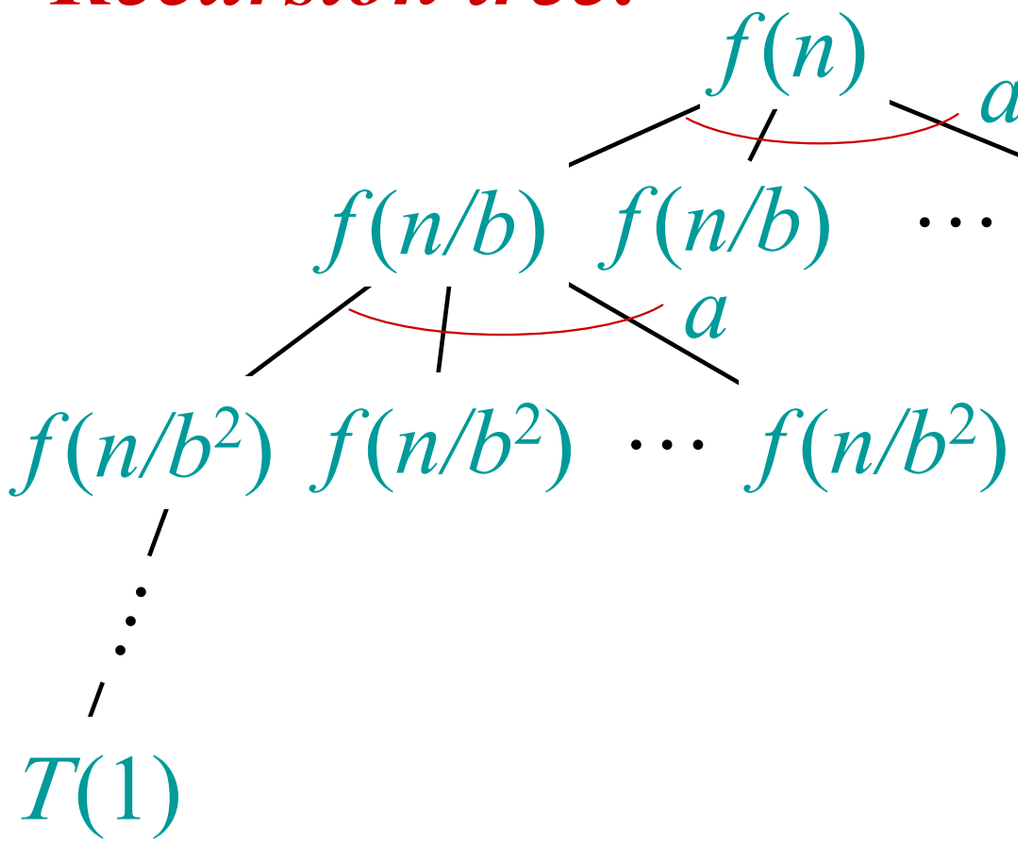
$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^2/\lg n.$

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

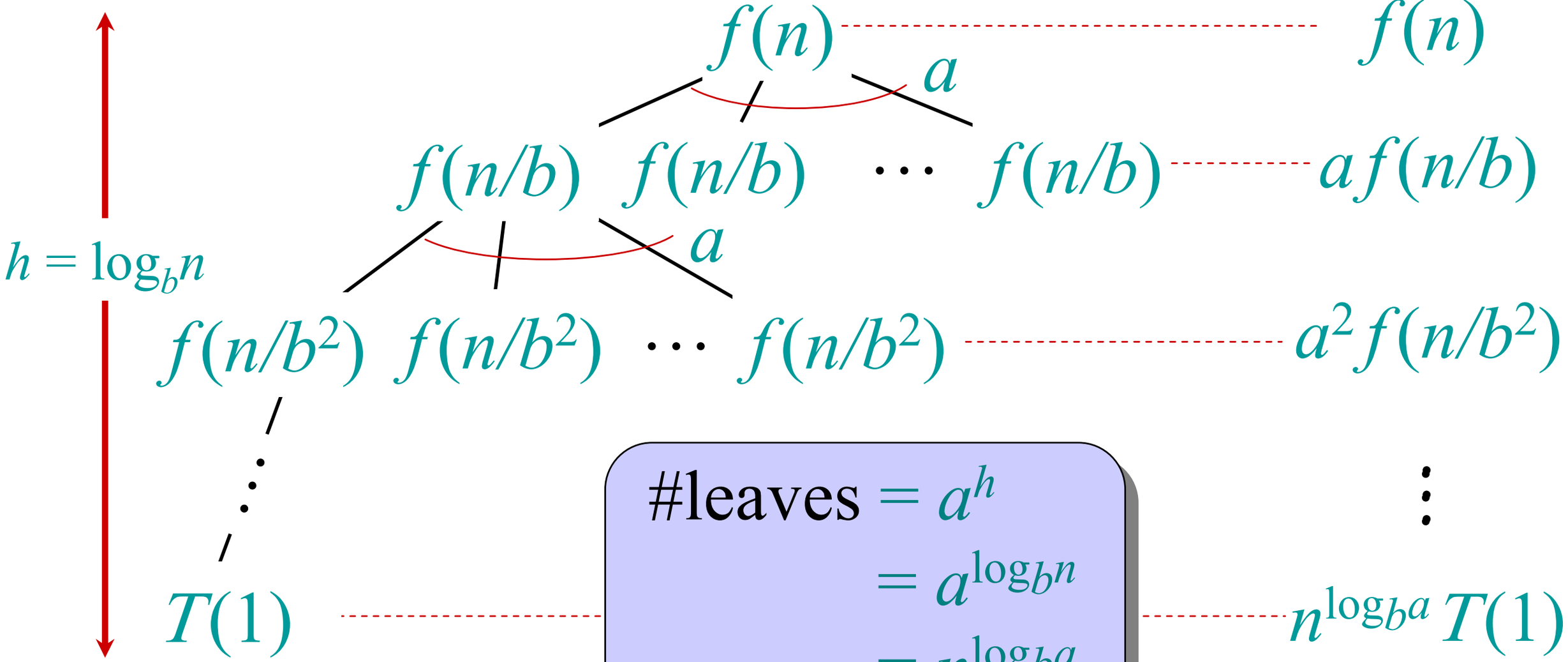
Idea of master theorem

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

Recursion tree:



Recursion tree:

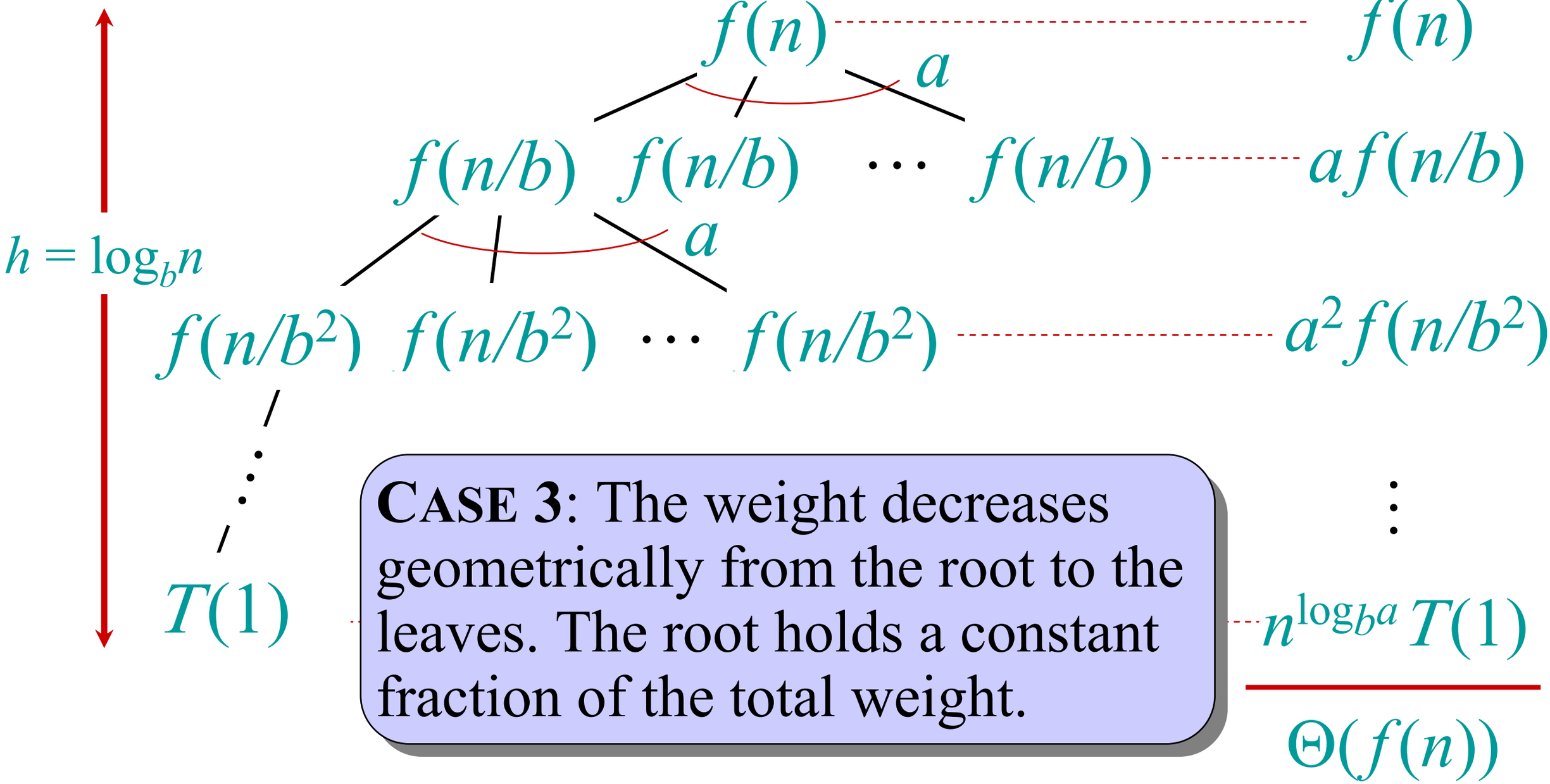


reminder: $\log_b(x) = \dfrac{\ln(x)}{\ln(b)}$

$$a^h = a^{\log_b n} = a^{\frac{\ln n}{\ln b}} = a^{\frac{\ln a}{\ln a}\frac{\ln n}{\ln b}} =$$
$$= a^{\log_a n \log_b a} = (a^{\log_a n})^{\log_b a} = n^{\log_b a}$$

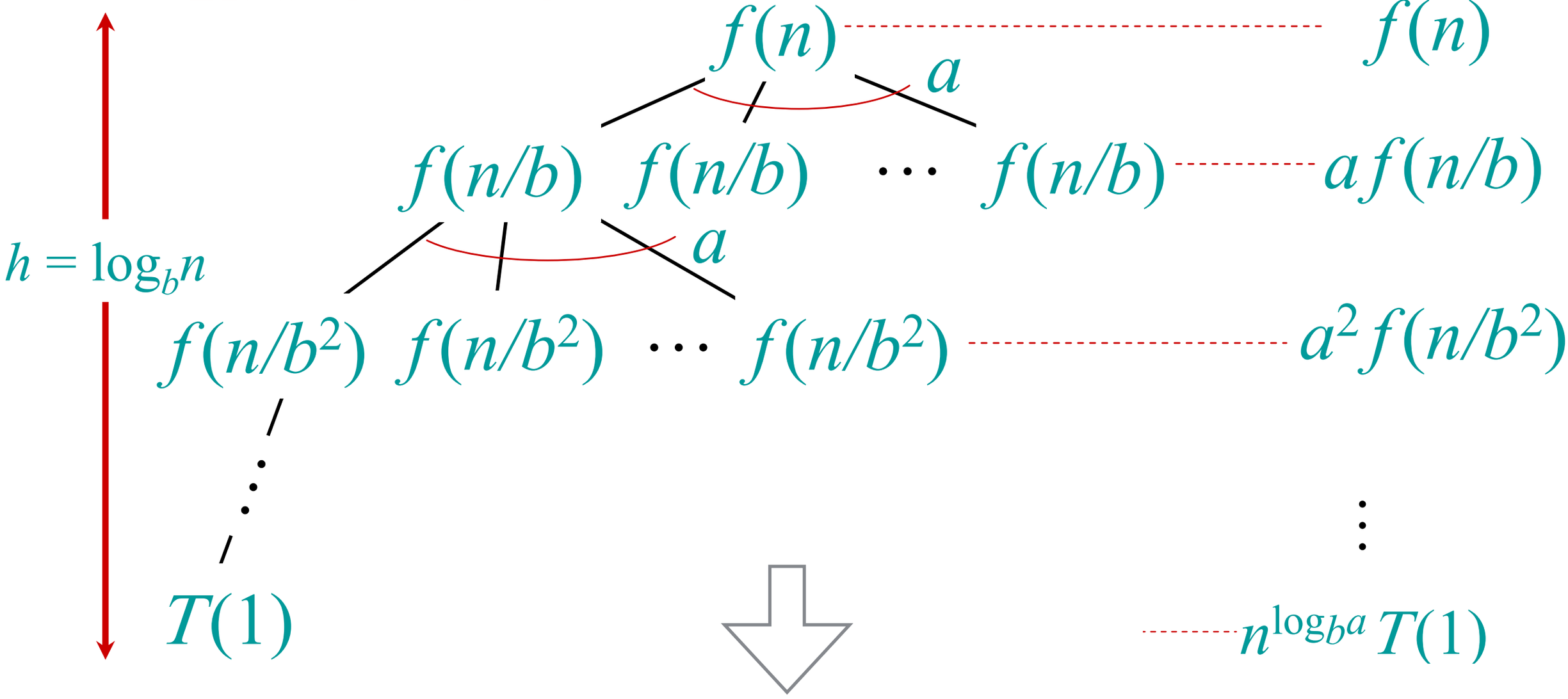
($a^{\log_a n} = n$)

Recursion tree:



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

Recursion tree:



$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a})$$

Three cases:

(1) If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = O(n^{\log_b a})$.

(2) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.

(3) If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if f satisfies the smoothness condition $af(n/b) \leq cf(n)$ for some constant $c < 1$, then $T(n) = \Theta(f(n))$.

Proof of case (1):

$$\text{If } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for some constant } \varepsilon > 0, \text{ then } T(n) = O(n^{\log_b a})$$

Given that we want to prove that T(n) is upper bounded by some quantity, we may take advantage of inequalities.

We know from the analysis of the tree that $T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a})$

Thus, for case (1) we may write:

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) \leq \sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a - \varepsilon} + O(n^{\log_b a})$$

because for this case we have $f(n) = O(n^{\log_b a - \varepsilon})$, i.e. for some $\varepsilon > 0$

For n sufficiently large and for some constant c we have

$$f(n/b^i) \leq \text{ c } (n/b^i)^{\log_b a - \varepsilon}$$

Thus by substitution in the sum we find (we don't need to write the constant in front)

$$\sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a - \varepsilon} = n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} a^i b^{-i \log_b a} b^{i\varepsilon} = n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} a^i a^{-i} b^{i\varepsilon}$$

a^-i

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

n^eps b^eps

$$= n^{\log_b a - \varepsilon} \frac{n^\varepsilon b^\varepsilon - 1}{b^\varepsilon - 1} \leq n^{\log_b a - \varepsilon} \frac{n^\varepsilon b^\varepsilon}{b^\varepsilon - 1} = n^{\log_b a} \frac{b^\varepsilon}{b^\varepsilon - 1}$$

$$= O(n^{\log_b a}).$$

This results, together with $T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a})$ gives

$$T(n) = O(n^{\log_b a})$$

□

Little Remark: O(f(n))+O(f(n)) is still O(f(n))

Proofs of case-2 and case-3 are similar. I leave them to you if you are interested (not required for the exam).
