CS473-Algorithms I

Lecture 3

Solving Recurrences

Solving Recurrences

- The analysis of merge sort Lecture 1 required us to solve a recurrence.
- Recurrences are like solving integrals, differential equations, etc.
  - Learn a few tricks.
- Lecture 4 : Applications of recurrences.

Recurrences

- Function expressed recursively

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(\lceil n/2 \rceil) + 1 & \text{if } n>1 \end{cases}$$

- Solve for $n = 2^k$

Recurrences

- Claimed answer: T(n) = lg$n$+1 = Θ (lg$n$)
  - Substitute claimed answer for T in the recurrence
  - Note: resulting equations are true when $n = 2^k$

i.e. $$\lg n + 1 = \begin{cases} 1 & \text{if } n=2^0=1 \\ (\lg(\lceil n/2 \rceil) + 1) & \text{if } n=2^k>1 \end{cases}$$

Tedious technicality: haven't shown T(n) = Θ (lg$n$)

– But, since T(n) is monotonically non-decreasing function of n

$$n \le 2^{\lg n} \Rightarrow T(n) \le T(2^{\lceil \lg n \rceil})$$

$$= \lg(2^{\lceil \lg n \rceil}) + 1 = \lceil \lg n \rceil + 1 = \Theta(\lg n)$$

– Thus, ceiling didn't matter much

Recurrences

- Technically, should be careful about floors and ceilings (as in the book)

- But, usually it is okay
  - To ignore floor/ceiling
  - Just solve for exact powers of 2 (or b)

Boundary Conditions

- Usually assume $T(n) = \Theta(1)$ for small n
  - Does not usually affect soln. (if polynomially bounded)
- Example: Initial condition affects soln.
  - Exponential $T(n) = (T(n/2))^2$

    If $T(1) = c$ for a constant $c > 0$, then

    $T(2) = (T(1))^2 = c^2$, $T(4) = (T(2))^2 = c^4$,

    $T(n) = \Theta(c^n)$

E.g.,

$$\left.\begin{array}{l} T(1) = 2 \Rightarrow T(n) = \Theta(2^n) \\ T(1) = 3 \Rightarrow T(n) = \Theta(3^n) \end{array}\right\} \textit{However} \quad 2^n \neq \Theta(3^n)$$

Difference in soln. is more dramatic with

$$T(1) = 1 \Rightarrow T(n) = \Theta(1^n) = \Theta(1)$$

Substitution Method

- The most general method:

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

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

Example of Substitution

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

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

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

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

$\leq cn^3$

whenever $(c/2)\,n^3 - n \geq 0$, for example,

if $c \geq 2$ and $n \geq 1$ *residual*

Example (Continued)

- We must also handle the initial 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 $1 \leq n < n_0$, we have "$\Theta(1)$" $\leq cn^3$, 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 $T(k) \leq ck^2$ for $k < n$:

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\leq cn^2 + n \\
&= \cancel{O(n^2)} \quad \text{Wrong ! We must prove the I.H.} \\
&= cn^2 - (-n) \\
&\leq cn^2
\end{aligned}
$$

for no choice of $c > 0$. Lose!

A Tighter Upper Bound!

- IDEA: Strengthen the inductive hypothesis.
- *Subtract* a low-order term.

Inductive hypothesis: $T(k) \le c_1 k^2 - c_2 k$ for $k < n$

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4\,(c_1\,(n/2)^2 - c_2\,(n/2)) + n \\
&= c_1\,n^2 - 2\,c_2\,n + n \\
&= c_1\,n^2 - c_2\,n - (c_2\,n - n) \\
&\le c_1\,n^2 - c_2\,n \quad \text{if } c_2 > 1
\end{aligned}
$$

Pick c_1 big enough to handle the initial conditions

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.

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

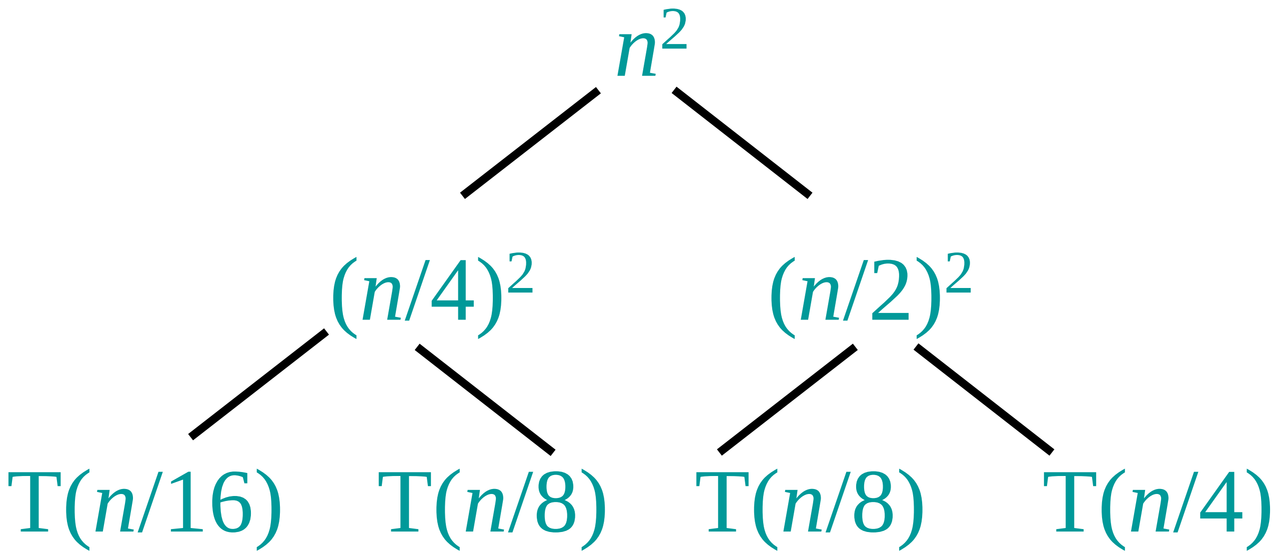
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$n^2$$

$$T(n/4) \qquad\qquad T(n/2)$$

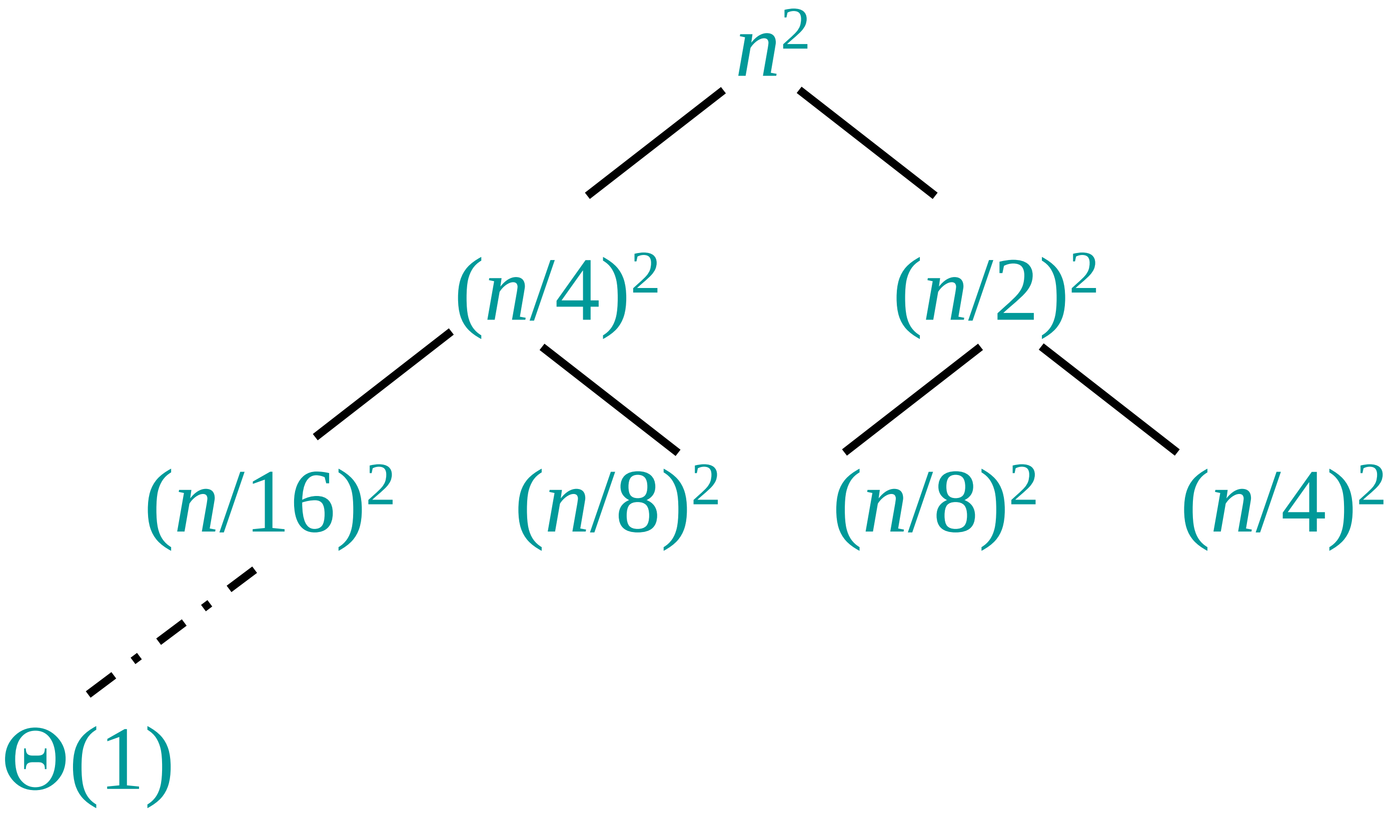
Example of Recursion Tree

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



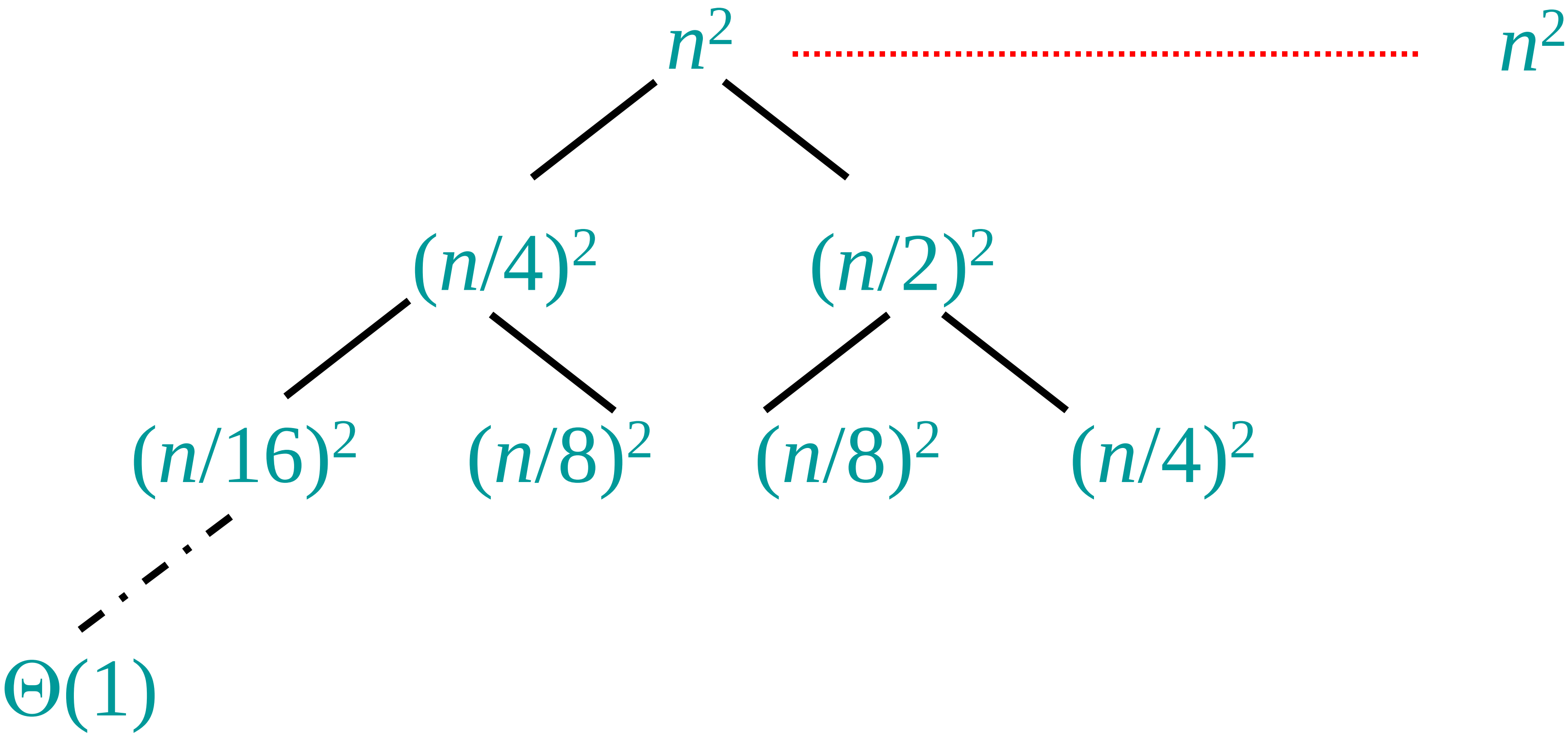
Example of Recursion Tree

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



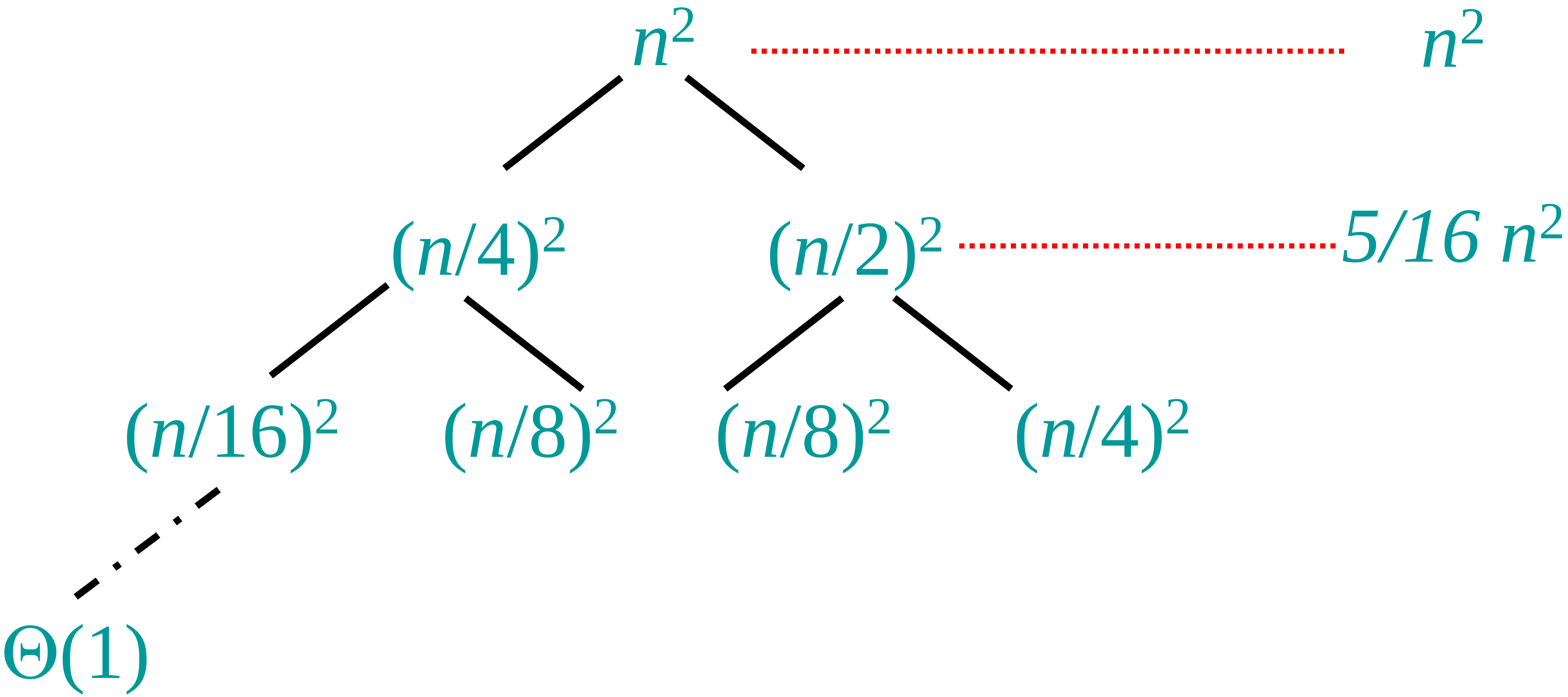
Example of Recursion Tree

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



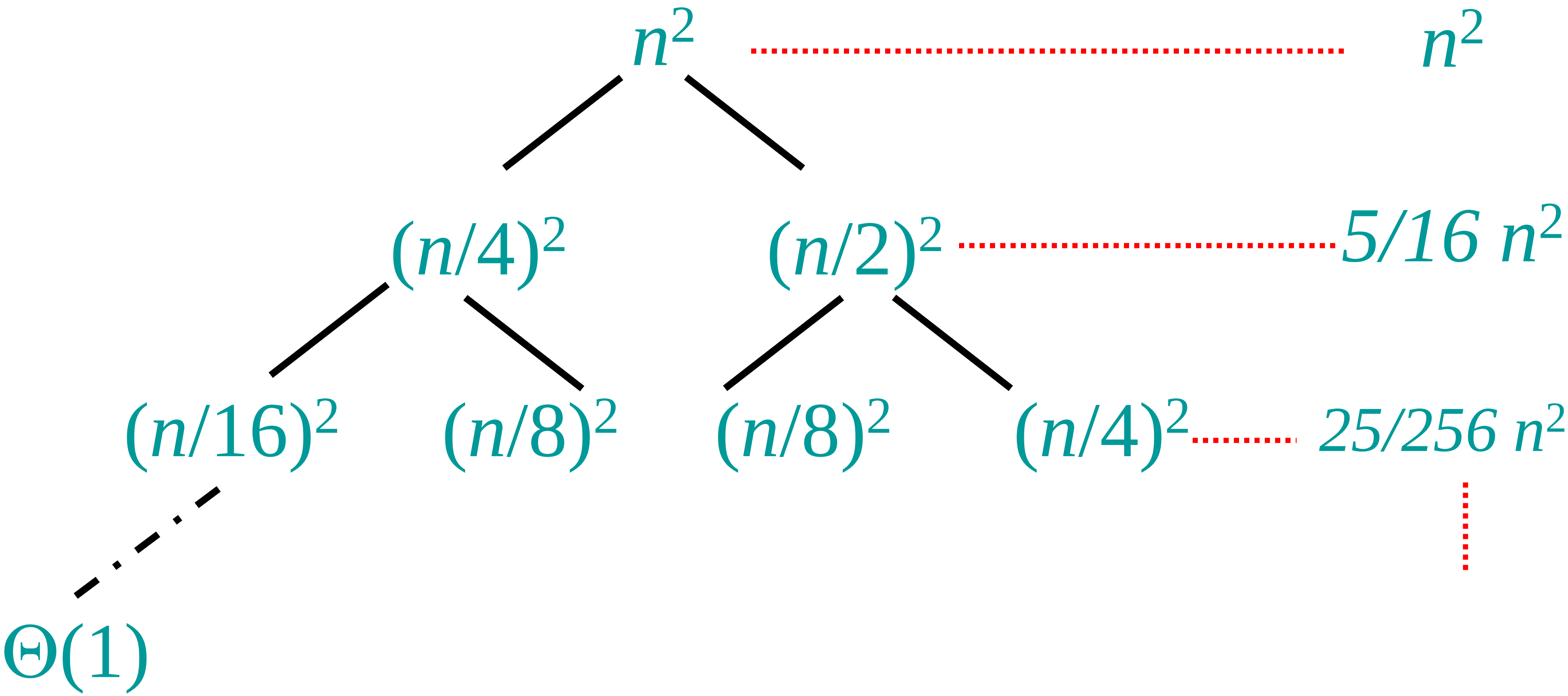
Example of Recursion Tree

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

n^2 .... n^2

$(n/4)^2$ $\quad$ $(n/2)^2$ .... $5/16\, n^2$

$(n/16)^2$ $\quad$ $(n/8)^2$ $\quad$ $(n/8)^2$ $\quad$ $(n/4)^2$ .... $25/256\, n^2$

$\Theta(1)$

Total $= n^2 \left(1 + 5/16 + (5/16)^2 + (5/16)^2 + \ldots\right)$

$= \Theta(n^2)$ geometric series

The Master Method

- The master method applies to recurrences of the form

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

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

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

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, $k=0$

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

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 $4c(n/2)^3 \le cn^3$ (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 $n^{\varepsilon} = \omega(\lg n)$

General Method (Akra-Bazzi)

$$T(n) = \sum_{i=1}^{k} a_i T(n / b_i) + f(n)$$

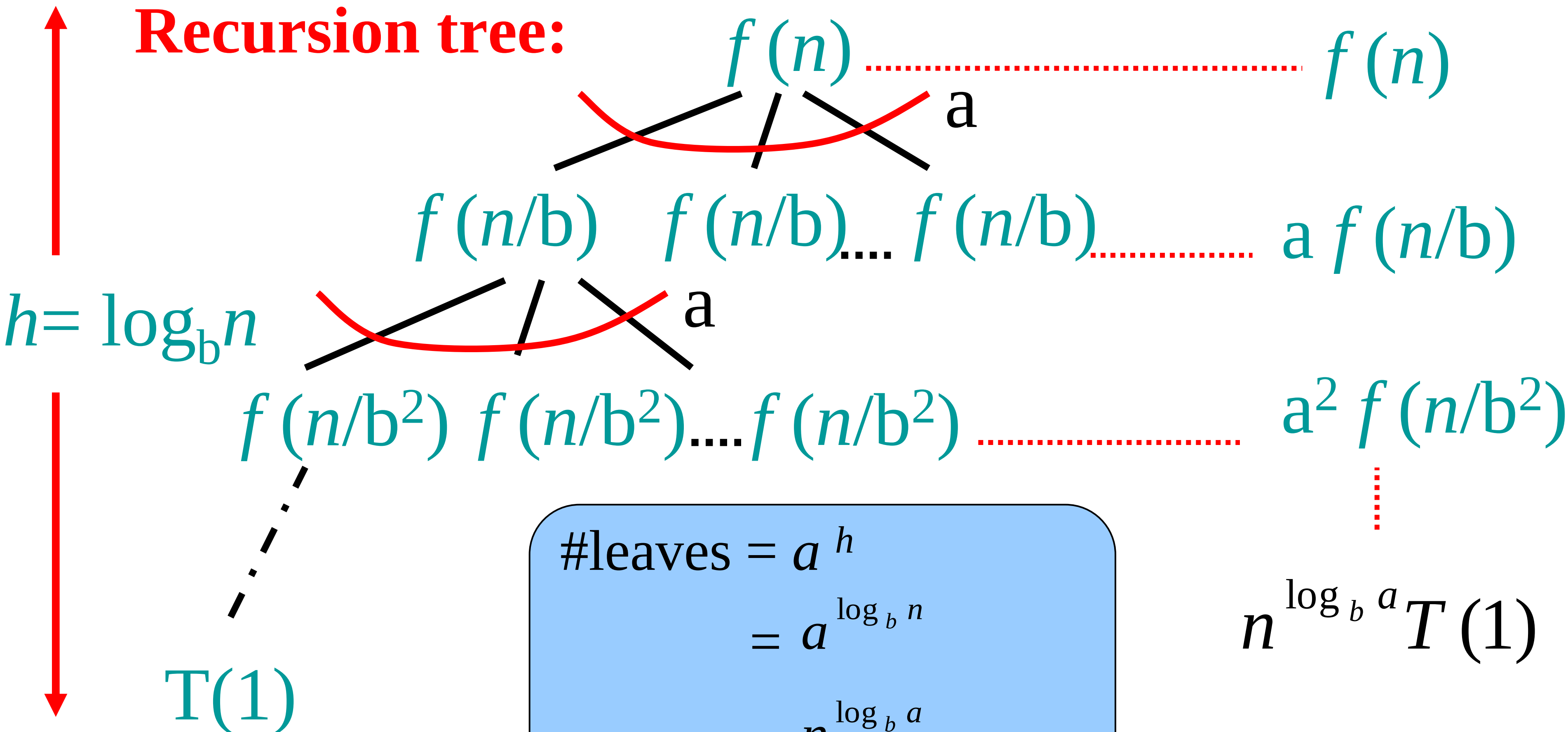
Let p be the unique solution to

$$\sum_{i=1}^{k} (a_i / b^p{}_i) = 1$$

Then, the answers are the same as for the master method, but with n^p instead of $n^{\log_b a}$

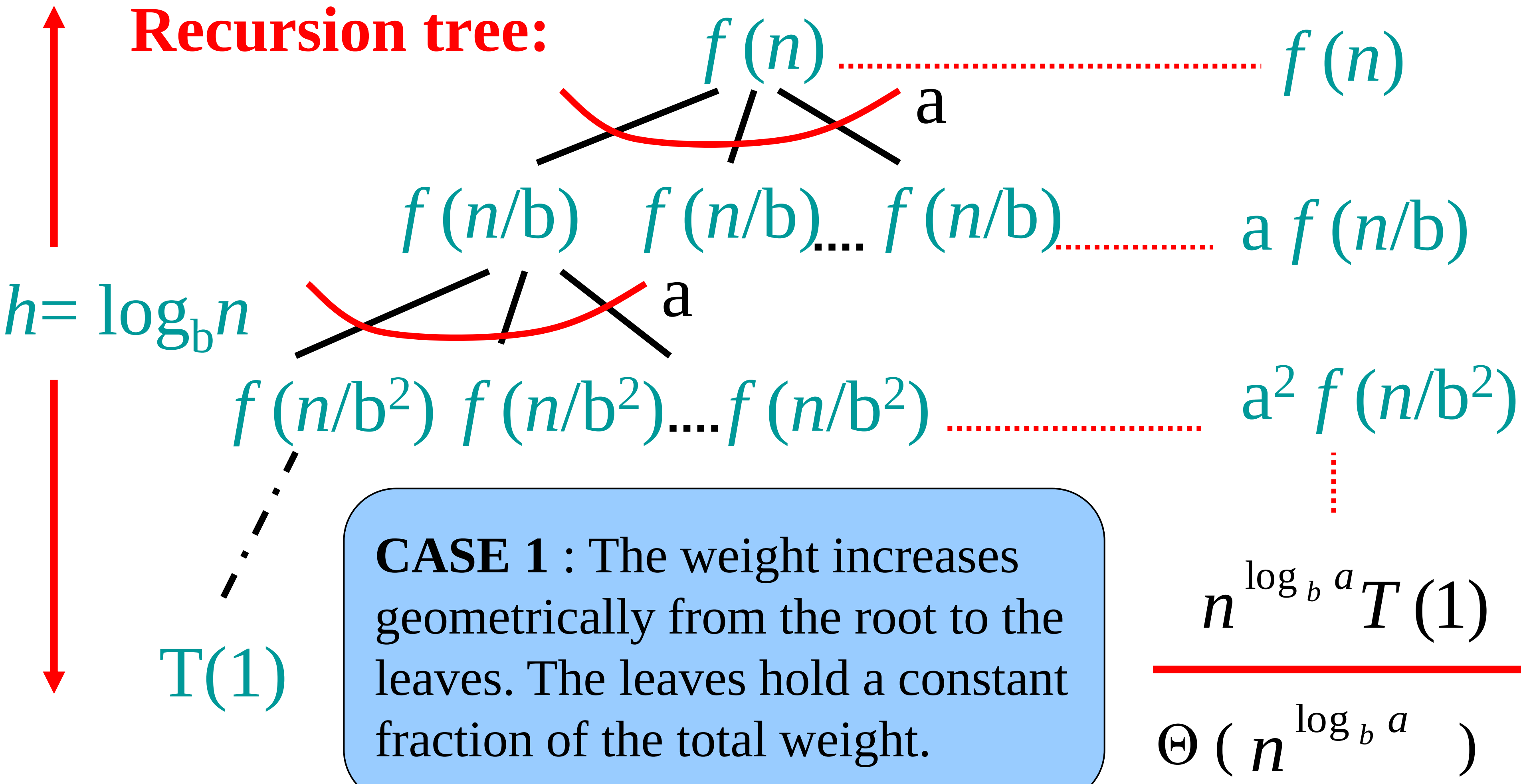
(*Akra and Bazzi also prove an even more general result*.)

Idea of Master Theorem



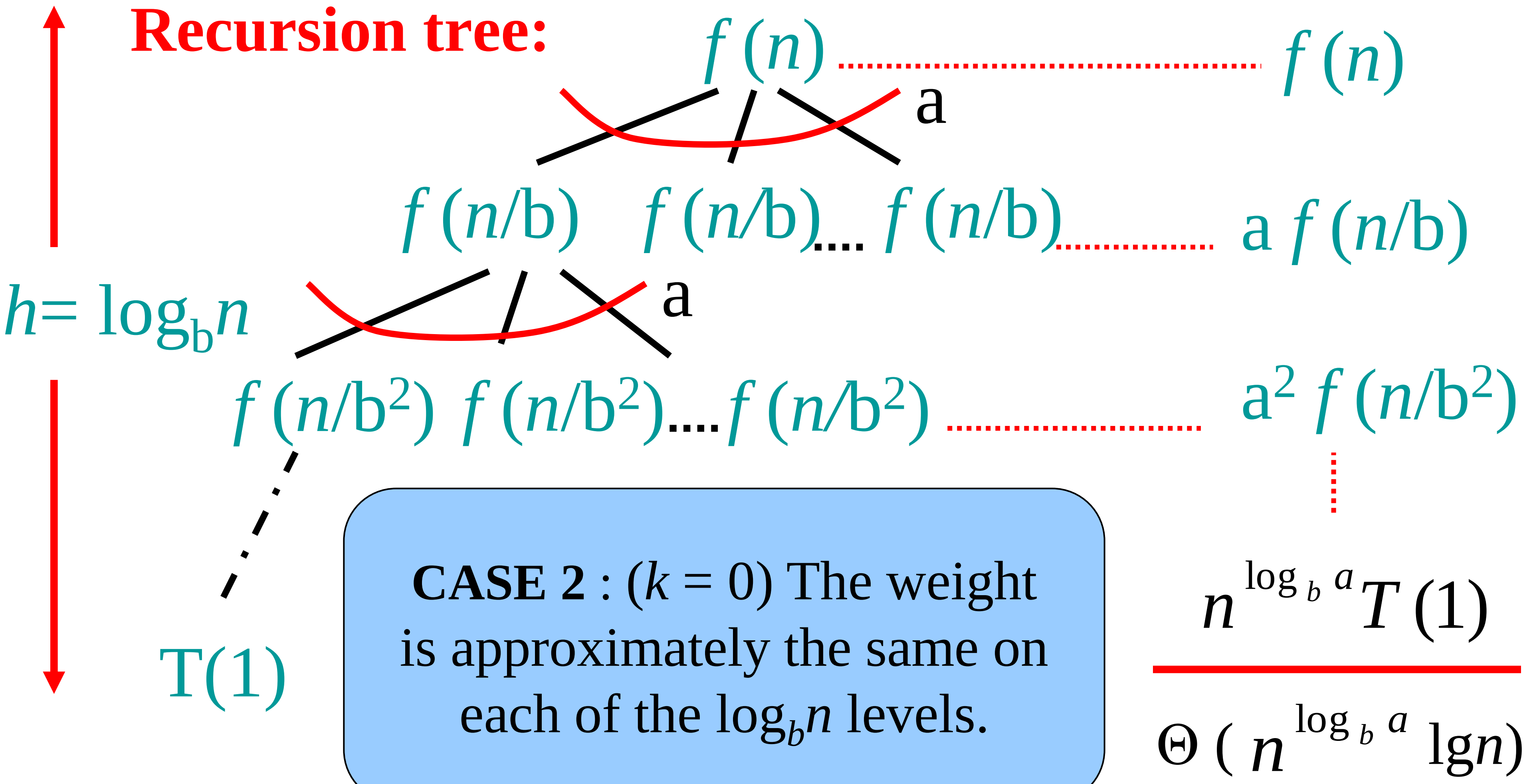
$$\#\text{leaves} = a^{h}$$
$$= a^{\log_b n}$$
$$= n^{\log_b a}$$

Idea of Master Theorem



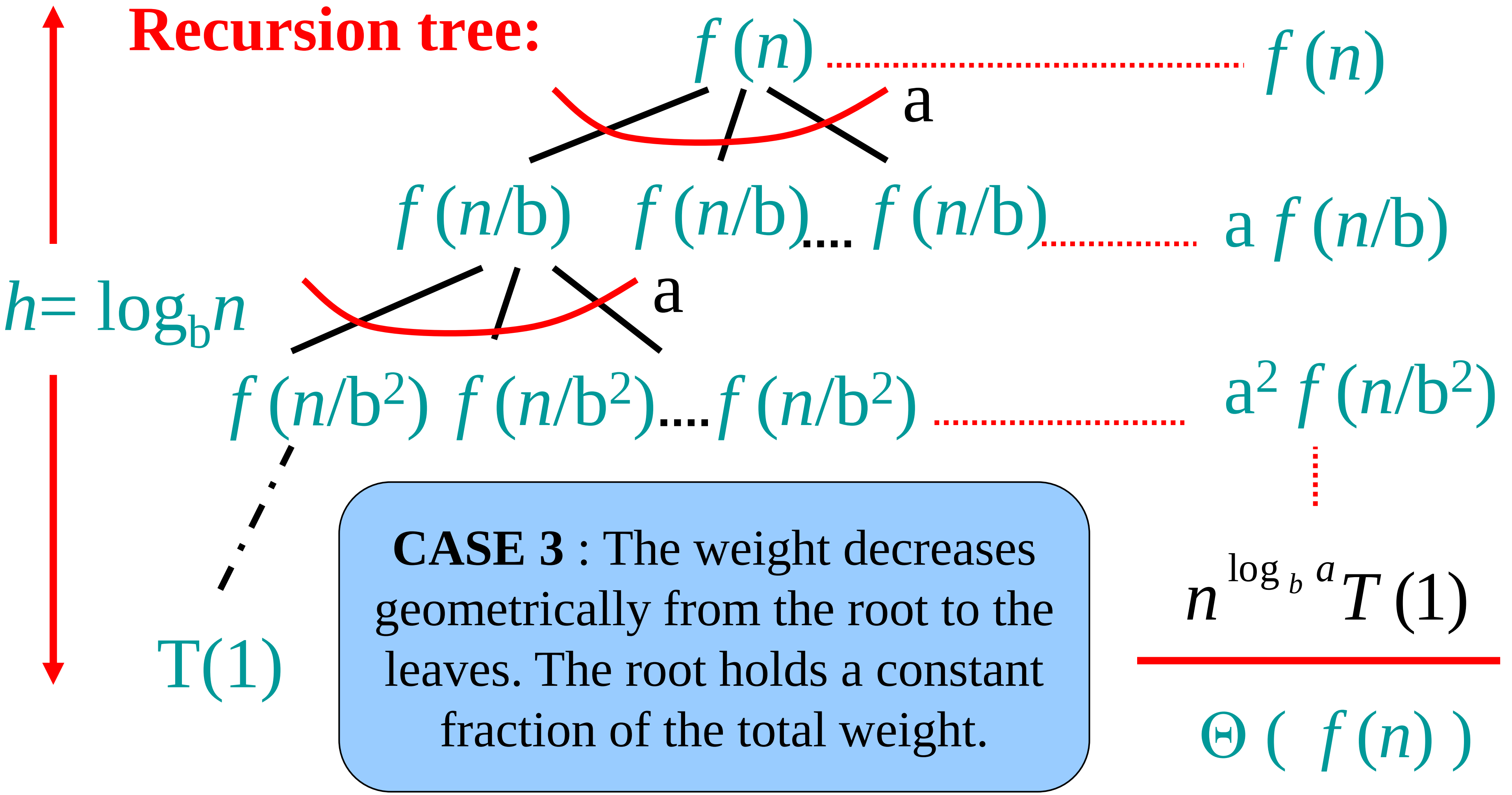
CASE 1 : The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

Idea of Master Theorem



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.

Proof of Master Theorem: Case 1 and Case 2

- Recall from the recursion tree (note $h = \lg_b n$=tree height)

$$T(n) = \underbrace{\Theta(n^{\log_b a})}_{\text{Leaf cost}} + \underbrace{\sum_{i=0}^{h-1} a^i f(n/b^i)}_{\text{Non-leaf cost} = g(n)}$$

Proof of Case 1

- $$\frac{n^{\log_b a}}{f(n)} = \Omega(n^{\varepsilon}) \qquad \text{for some } \varepsilon > 0$$
- $$\frac{n^{\log_b a}}{f(n)} = \Omega(n^{\varepsilon}) \Rightarrow \frac{f(n)}{n^{\log_b a}} = O(n^{-\varepsilon}) \Rightarrow f(n) = O(n^{\log_b a - \varepsilon})$$
- $$g(n) = \sum_{i=0}^{h-1} a^i O\left((n/b^i)^{\log_b a - \varepsilon}\right) = O\left(\sum_{i=0}^{h-1} a^i (n/b^i)^{\log_b a - \varepsilon}\right)$$
- $$= O\left(n^{\log_b a - \varepsilon} \sum_{i=0}^{h-1} a^i b^{i\varepsilon} / b^{i \log_b a}\right)$$

Case 1 (cont')

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

= An increasing geometric series since b > 1

$$= \frac{b^{\varepsilon h} - 1}{b^{\varepsilon} - 1} = \frac{(b^h)^{\varepsilon} - 1}{b^{\varepsilon} - 1} = \frac{(b^{\log_b n})^{\varepsilon} - 1}{b^{\varepsilon} - 1} = \frac{n^{\varepsilon} - 1}{b^{\varepsilon} - 1} = O(n^{\varepsilon})$$

Case 1 (cont')

- $g(n) = O\left(n^{\log_b a-\varepsilon} O(n^{\varepsilon})\right) = O\left(\dfrac{n^{\log_b a}}{n^{\varepsilon}} O(n^{\varepsilon})\right)$

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

- $T(n) = \Theta(n^{\log_b a}) + g(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a})$

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

Q.E.D.

Proof of Case 2 (limited to $k=0$)

- $$\frac{f(n)}{n^{\log_b a}} = \Theta(\lg^0 n) = \Theta(1) \Rightarrow f(n) = \Theta(n^{\log_b a}) \Rightarrow f(n/b^i) = \Theta\left((\frac{n}{b^i})^{\log_b a} \right)$$
- $$\therefore g(n) = \sum_{i=0}^{h-1} a^i \Theta\left((n/b^i)^{\log_b a} \right)$$
$$= \Theta\left(\sum_{i=0}^{h-1} a^i \frac{n^{\log_b a}}{b^{i \log_b a}} \right) = \Theta\left(n^{\log_b a} \sum_{i=0}^{h-1} a^i \frac{1}{(b^{\log_b a})^i} \right) = \Theta\left(n^{\log_b a} \sum_{i=0}^{h-1} a^i \frac{1}{a^i} \right)$$
$$= \Theta\left(n^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1 \right) = \Theta\left(n^{\log_b a} \log_b n \right) = \Theta\left(n^{\log_b a} \lg n \right)$$
- $$T(n) = n^{\log_b a} + \Theta(n^{\log_b a} \lg n)$$
$$= \Theta\left(n^{\log_b a} \lg n \right)$$

Q.E.D.

Conclusion

- Next time: applying the master method.