CS473 - Algorithms I

Lecture 3
Solving Recurrences

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Solving Recurrences

- Reminder: Runtime ($T(n)$) of *MergeSort* was expressed as a recurrence

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

- Solving recurrences is like solving differential equations, integrals, etc.
  - *Need to learn a few tricks*

Recurrences

- Recurrence: *An equation or inequality that describes a function in terms of its value on smaller inputs.*

Example:

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

Recurrence - Example

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

- *Simplification: Assume $n = 2^k$*
- Claimed answer: $T(n) = \lg n + 1$
- Substitute claimed answer in the recurrence:

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

True when $n = 2^k$

Technicalities: Floor/Ceiling

- Technically, should be careful about the floor and ceiling functions (as in the book).
- e.g. For merge sort, the recurrence should in fact be:

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

- But, it's usually ok to:
  - ignore floor/ceiling
  - solve for exact powers of 2 (or another number)

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Technicalities: Boundary Conditions

- Usually assume: $T(n) = \Theta(1)$ for sufficiently small n
  - Changes the exact solution, but usually the asymptotic solution is not affected (e.g. if polynomially bounded)

- For convenience, the boundary conditions generally implicitly stated in a recurrence

$$T(n) = 2T(n/2) + \Theta(n)$$

assuming that

$$T(n) = \Theta(1) \text{ for sufficiently small n}$$

Example: When Boundary Conditions Matter

- Exponential function: $T(n) = (T(n/2))^2$
- Assume $T(1) = c$ (where c is a positive constant).

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

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

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

- e.g. $$\left.\begin{aligned} T(1) = 2 &\Rightarrow T(n) = \Theta(2^n) \\ T(1) = 3 &\Rightarrow T(n) = \Theta(3^n) \end{aligned}\right\} \textit{However} \quad \Theta(2^n) \neq \Theta(3^n)$$

- Difference in solution more dramatic when:

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

Solving Recurrences

- We will focus on 3 techniques in this lecture:

  1. Substitution method

  2. Recursion tree approach

  3. Master method

Substitution Method

- The most general method:
  1. Guess
  2. Prove by induction
  3. Solve for constants

Substitution Method: Example

Solve $T(n) = 4T(n/2) + n$ *(assume $T(1) = \Theta(1)$)*

1. Guess $T(n) = O(n^3)$ (need to prove O and Ω separately)

2. Prove by induction that $T(n) \leq cn^3$ for large n (i.e. $n \geq n_0$)

Inductive hypothesis: $T(k) \leq ck^3$ for any $k < n$

Assuming ind. hyp. holds, prove $T(n) \leq cn^3$

Substitution Method: Example – cont'd

Original recurrence: $T(n) = 4T(n/2) + n$

From inductive hypothesis: $T(n/2) \leq c(n/2)^3$

Substitute this into the original recurrence:

$$
\begin{aligned}
T(n) &\leq 4c\,(n/2)^3 + n \\
&= (c/2)\,n^3 + n \\
&= cn^3 - ((c/2)n^3 - n) \\
&\leq cn^3
\end{aligned}
$$

desired - residual

when $((c/2)n^3 - n) \geq 0$

Substitution Method: Example – cont'd

- So far, we have shown:

  $T(n) \leq cn^3$ when $((c/2)n^3 - n) \geq 0$

- We can choose $c \geq 2$ and $n_0 \geq 1$
- But, the proof is not complete yet.
- Reminder: Proof by induction:
  1. Prove the base cases → *haven't proved the base cases yet*
  2. Inductive hypothesis for smaller sizes
  3. Prove the general case

Substitution Method: Example – cont'd

- We need to prove the base cases

Base: $T(n) = \Theta(1)$ for small n (e.g. for $n = n_0$)

- We should show that:

  $$\text{“}\Theta(1)\text{”} \le cn^3 \quad \text{for } n = n_0$$

  This holds if we pick c big enough

- So, the proof of $T(n) = O(n^3)$ is complete.
- But, is this a tight bound?

Example: A tighter upper bound?

- Original recurrence: $T(n) = 4T(n/2) + n$
- Try to prove that $T(n) = O(n^2)$,

  i.e. $T(n) \le cn^2$ for all $n \ge n_0$

- Ind. hyp: Assume that $T(k) \le ck^2$ for $k < n$
- Prove the general case: $T(n) \le cn^2$

Example (cont'd)

- Original recurrence: $T(n) = 4T(n/2) + n$
- Ind. hyp: Assume that $T(k) \le ck^2$ for $k < n$
- Prove the general case: $T(n) \le cn^2$

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4c(n/2)^2 + n \\
&= cn^2 + n \\
&\cancel{= O(n^2)}
\end{aligned}
$$

Wrong! We must prove exactly

Example (cont'd)

- Original recurrence: $T(n) = 4T(n/2) + n$
- Ind. hyp: Assume that $T(k) \leq ck^2$ for $k < n$
- Prove the general case: $T(n) \leq cn^2$

- So far, we have:

$$T(n) \leq cn^2 + n$$

No matter which positive c value we choose,

this does not show that $T(n) \leq cn^2$

Proof failed?

Example (cont'd)

- What was the problem?
  - *The inductive hypothesis was not strong enough*
- Idea: Start with a stronger inductive hypothesis
  - *Subtract* a low-order term
- Inductive hypothesis: $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$
- Prove the general case: $T(n) \leq c_1 n^2 - c_2 n$

Example (cont'd)

- Original recurrence: $T(n) = 4T(n/2) + n$
- Ind. hyp: Assume that $T(k) \leq c_1k^2 - c_2k$ for $k < n$
- Prove the general case: $T(n) \leq c_1n^2 - c_2n$

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\leq 4\,(c_1(n/2)^2 - c_2(n/2)) + n \\
&= c_1n^2 - 2c_2n + n \\
&= c_1n^2 - c_2n - (c_2n - n) \\
&\leq c_1n^2 - c_2n \qquad \text{for } n(c_2 - 1) \geq 0 \\
& \qquad\qquad\qquad\quad\;\; \text{choose } c_2 \geq 1
\end{aligned}
$$

Example (cont'd)

- We now need to prove

$$T(n) \leq c_1 n^2 - c_2 n$$

for the base cases.

$T(n) = \Theta(1)$ for $1 \leq n \leq n_0$ (implicit assumption)

"$\Theta(1)$" $\leq c_1 n^2 - c_2 n$ for n small enough (e.g. $n = n_0$)

We can choose c_1 large enough to make this hold

- We have proved that $T(n) = O(n^2)$

Substitution Method: Example 2

- For the recurrence $T(n) = 4T(n/2) + n$, prove that $T(n) = \Omega(n^2)$

  i.e. $T(n) \geq cn^2$ for any $n \geq n_0$
- Ind. hyp: $T(k) \geq ck^2$ for any $k < n$
- Prove general case: $T(n) \geq cn^2$

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\geq 4c\,(n/2)^2 + n \\
&= cn^2 + n \\
&\geq cn^2 \qquad \text{since } n > 0
\end{aligned}
$$

Proof succeeded – no need to strengthen the ind. hyp as in the last example

Example 2 (cont'd)

- We now need to prove that

  $$T(n) \geq cn^2$$

  for the base cases

  $T(n) = \Theta(1)$ for $1 \leq n \leq n_0$ (implicit assumption)

  "$\Theta(1)$" $\geq cn^2$ for $n = n_0$

  n_0 is sufficiently small (i.e. constant)

  We can choose c small enough for this to hold

- We have proved that $T(n) = \Omega(n^2)$

Substitution Method - Summary

1. Guess the asymptotic complexity

1. Prove your guess using induction
    1. Assume inductive hypothesis holds for $k < n$
    2. Try to prove the general case for n

       Note: MUST prove the EXACT inequality

       CANNOT ignore lower order terms

       If the proof fails, strengthen the ind. hyp. and try again
    3. Prove the base cases (usually straightforward)

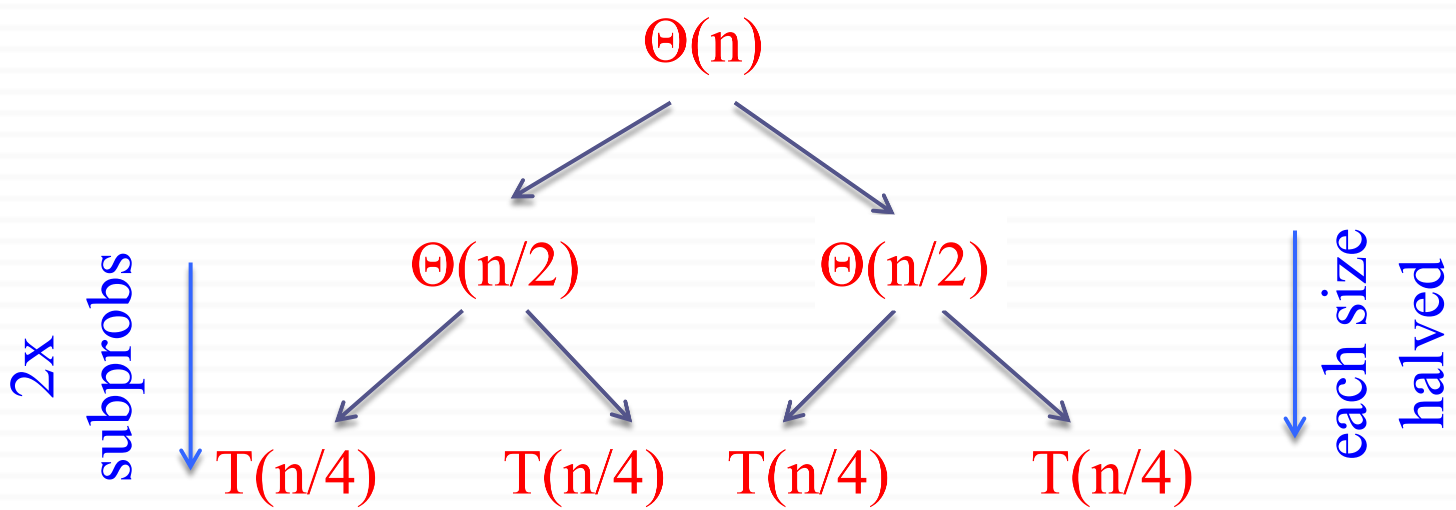
Recursion Tree Method

- A recursion tree models the runtime costs 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.
  - Not suitable for formal proofs
- The recursion-tree method promotes intuition, however.

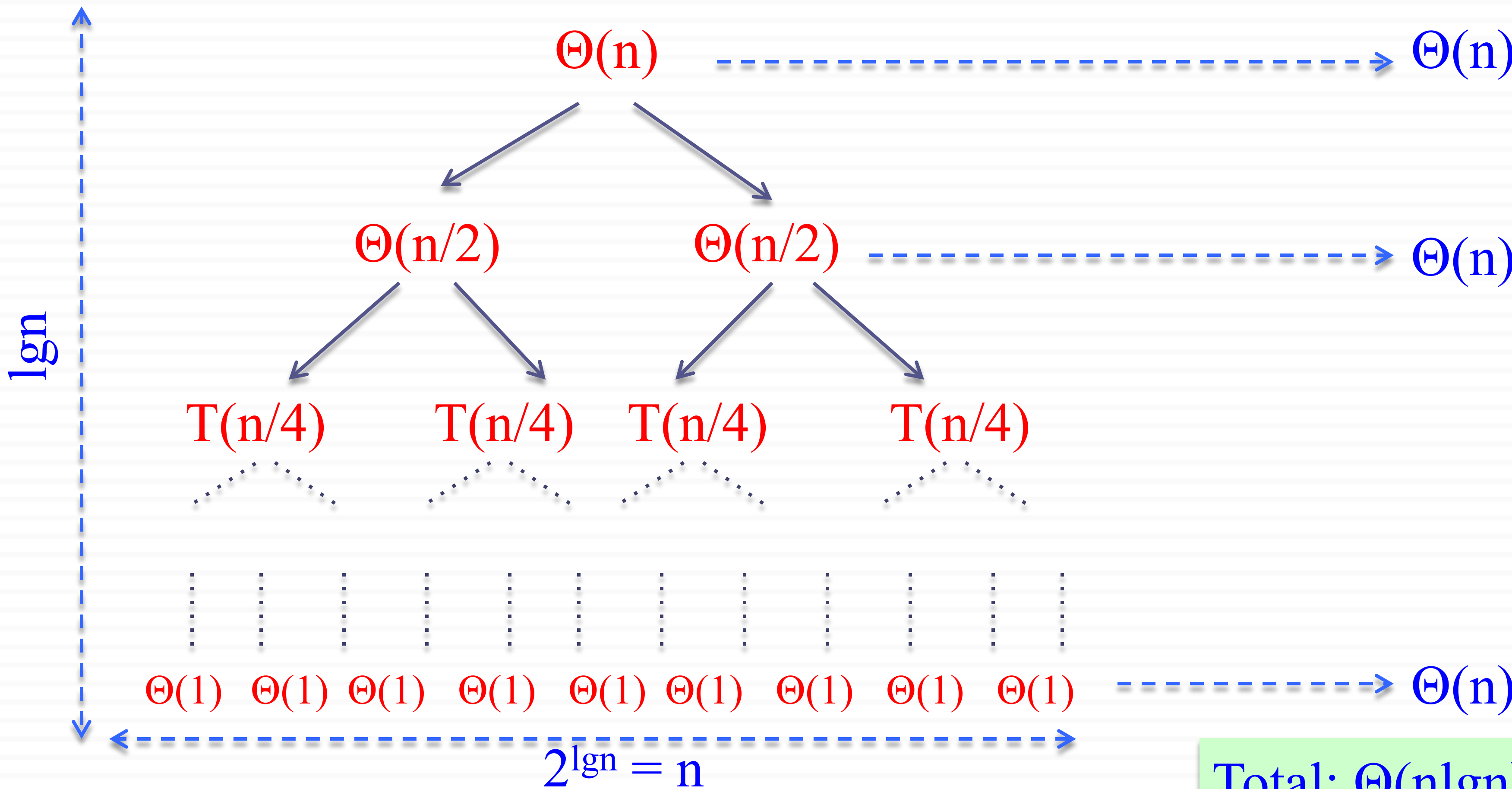
Solve Recurrence: $T(n) = 2T(n/2) + \Theta(n)$

$$
\begin{tikzcd}
& \Theta(n) \arrow[dl] \arrow[dr] & \\
T(n/2) & & T(n/2)
\end{tikzcd}
$$

Solve Recurrence: $T(n) = 2T(n/2) + \Theta(n)$



Solve Recurrence: $T(n) = 2T(n/2) + \Theta(n)$



Total: $\Theta(n\lg n)$

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

$$n^2$$

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

Example of Recursion Tree

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

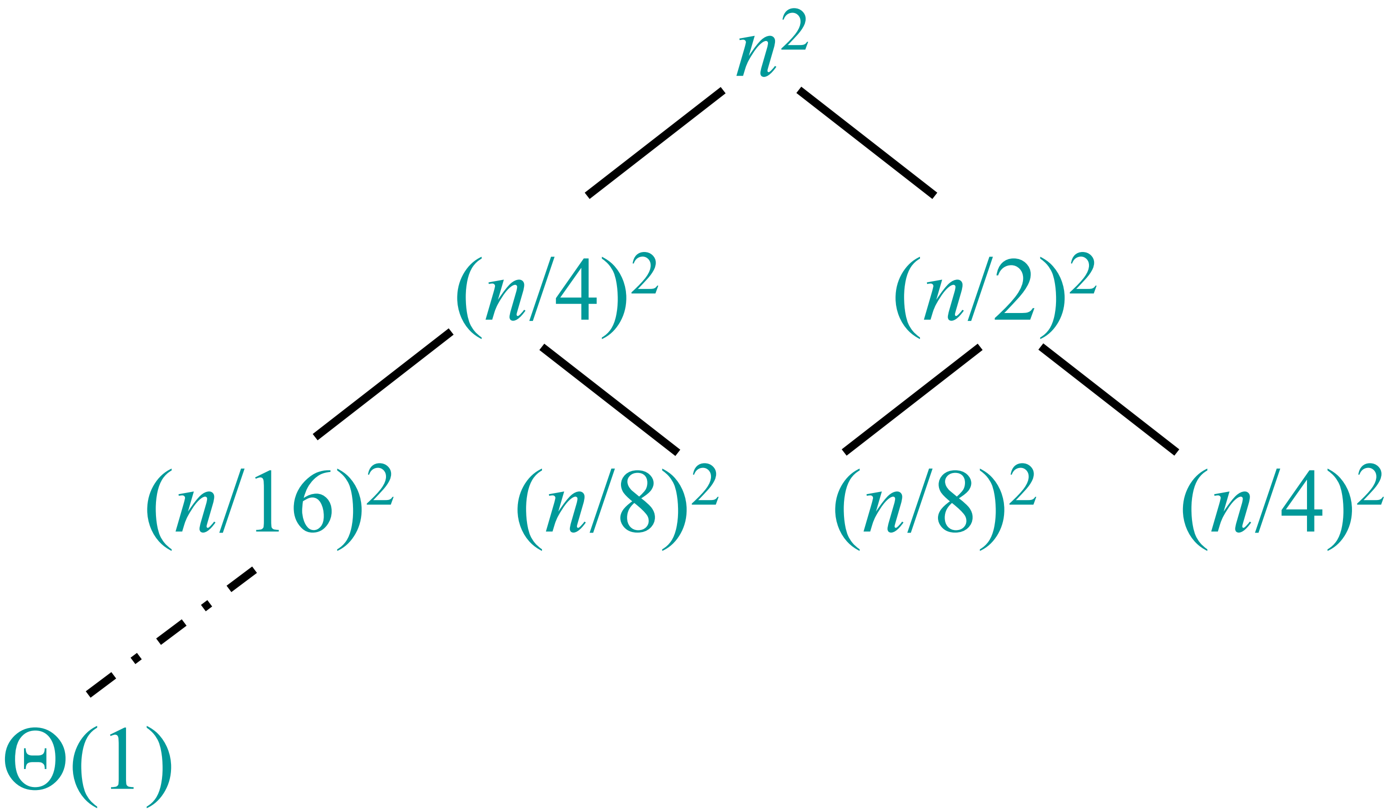
$$n^2$$

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

$$T(n/16) \quad T(n/8) \quad T(n/8) \quad T(n/4)$$

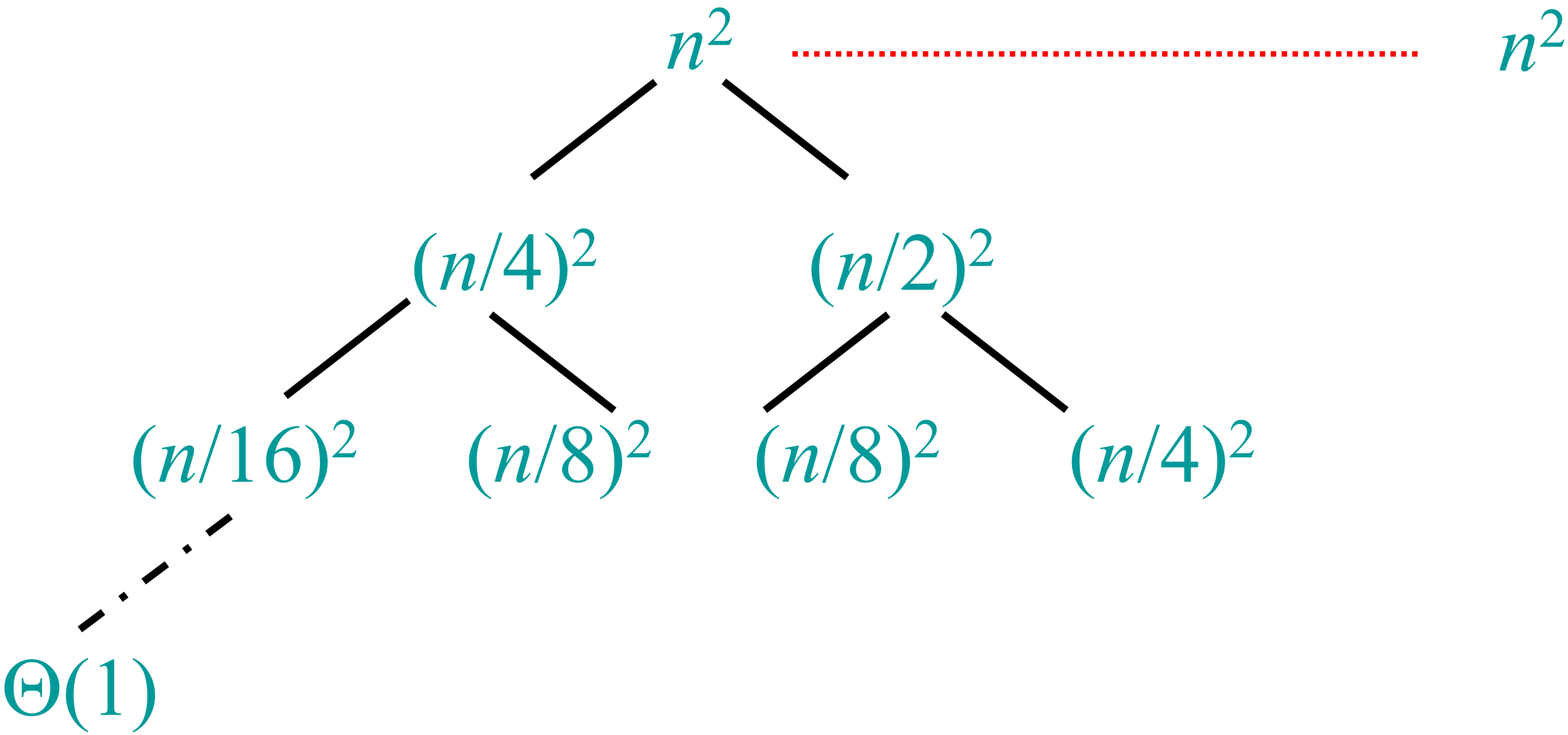
Example of Recursion Tree

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



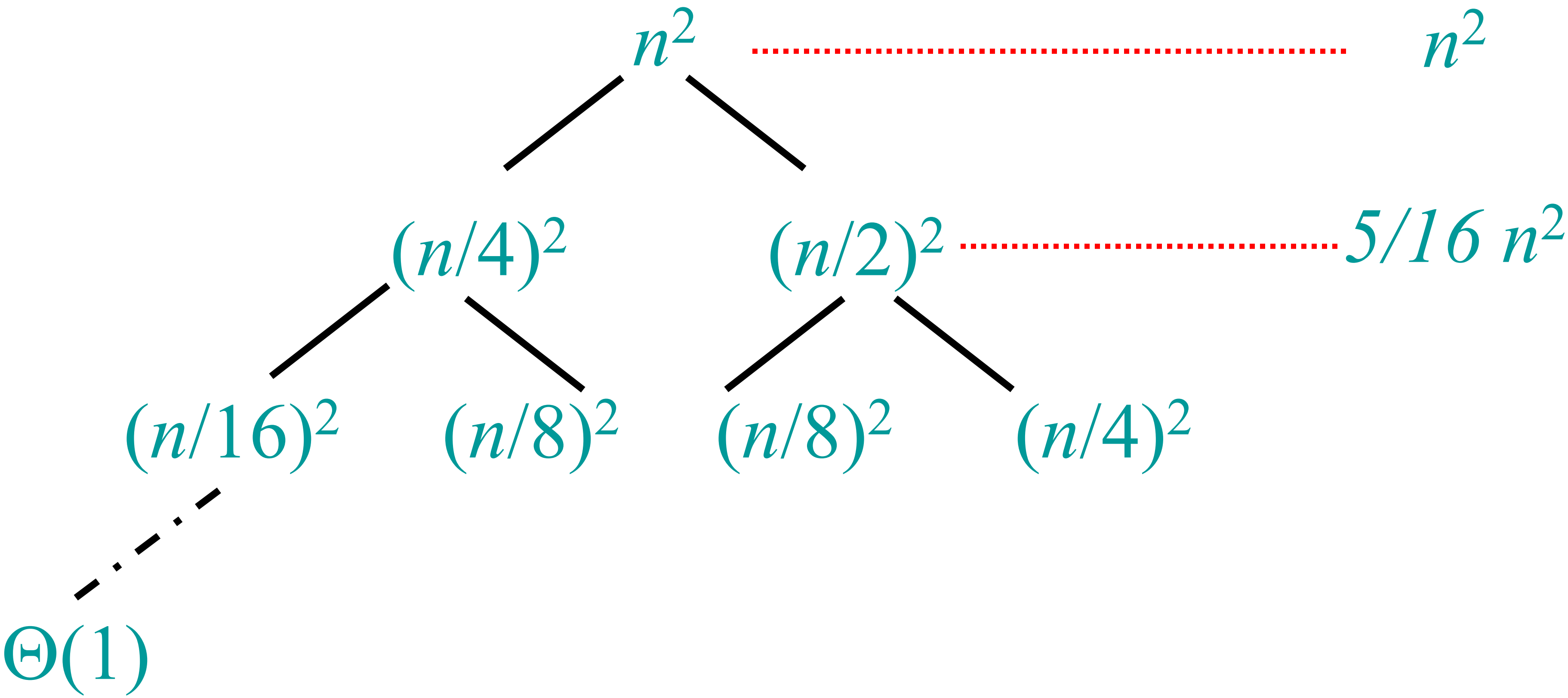
Example of Recursion Tree

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



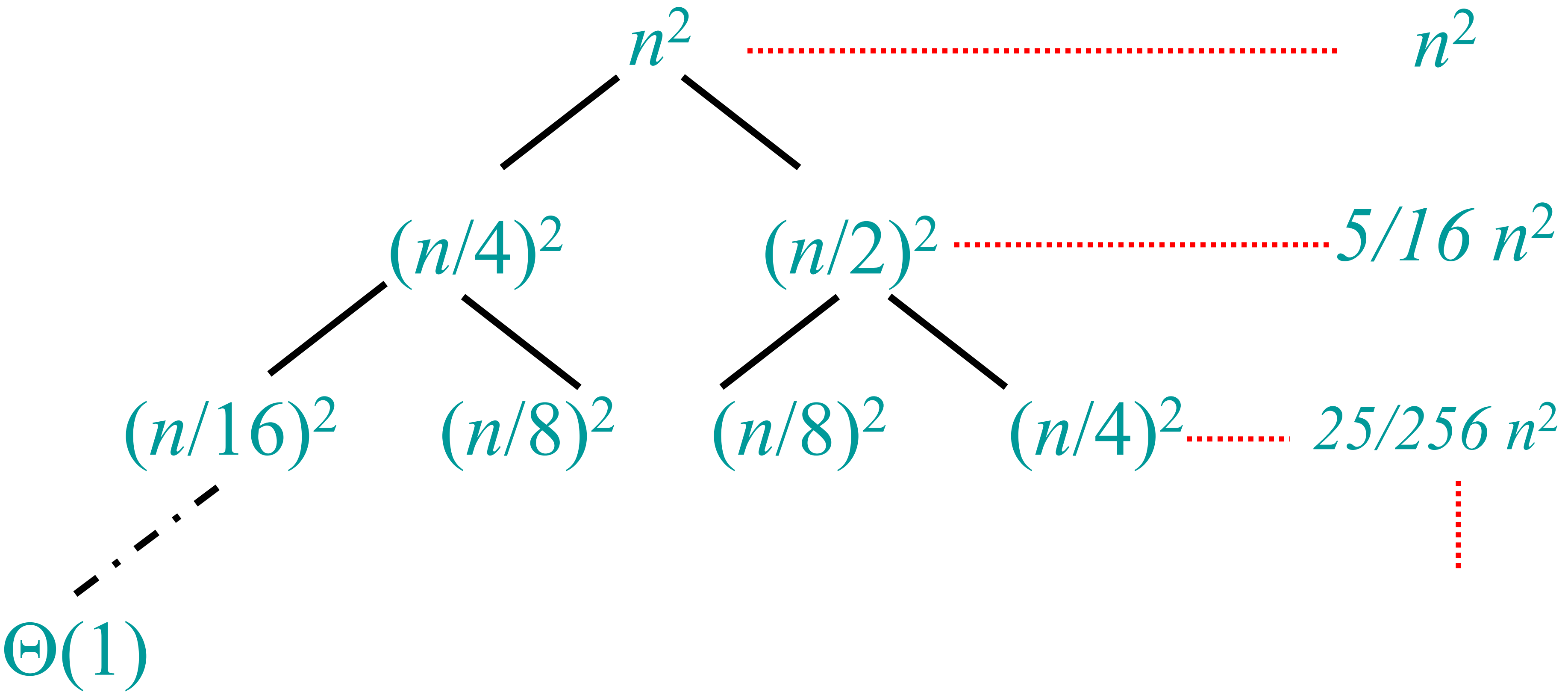
Example of Recursion Tree

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



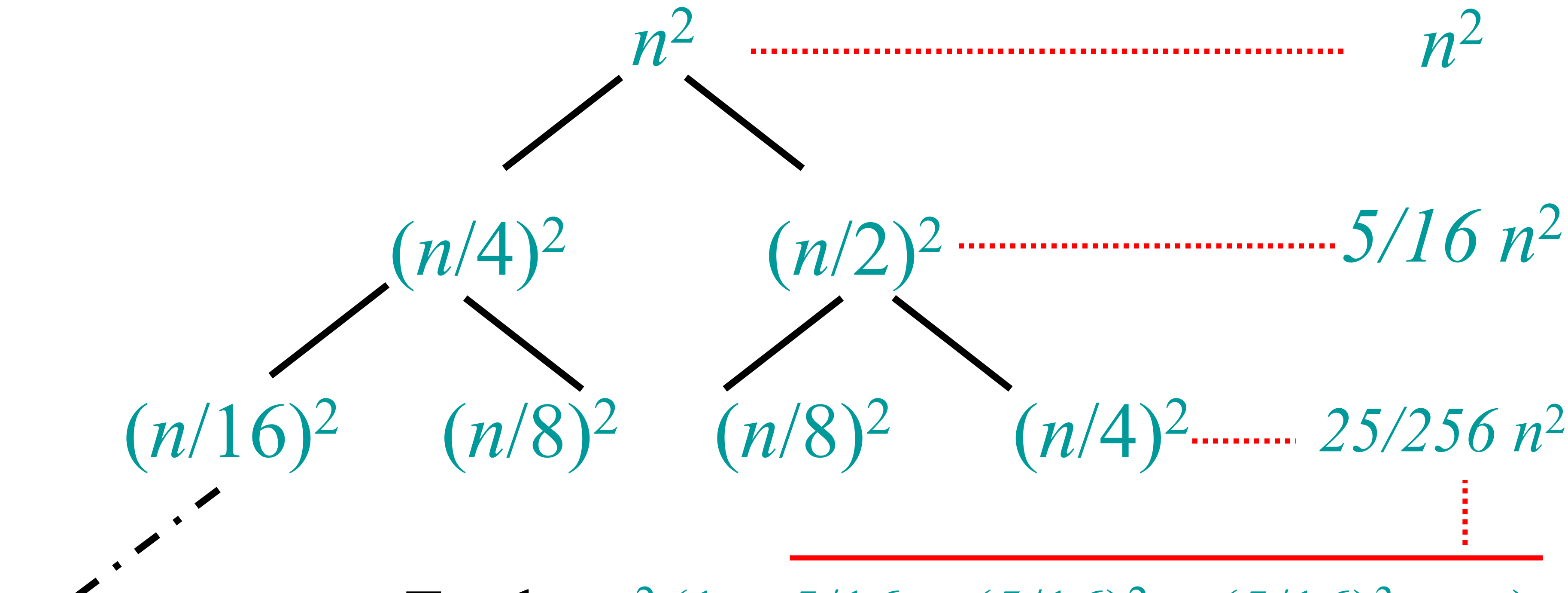
Example of Recursion Tree

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



Example of Recursion Tree

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



$\Theta(1)$

Total $= n^2 (1 + 5/16 + (5/16)^2 + (5/16)^3 + \ldots)$

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

The Master Method

- A powerful black-box method to solve recurrences.

- 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.

The Master Method: 3 Cases

- Recurrence: $T(n) = aT(n/b) + f(n)$

- Compare $f(n)$ with $n^{\log_b a}$
- Intuitively:

Case 1: $f(n)$ grows *polynomially slower* than $n^{\log_b a}$

Case 2: $f(n)$ grows *at the same rate* as $n^{\log_b a}$

Case 3: $f(n)$ grows *polynomially faster* than $n^{\log_b a}$

The Master Method: Case 1

- Recurrence: $T(n) = aT(n/b) + f(n)$

<u>Case 1</u>: $$\frac{n^{\log_b a}}{f(n)} = \Omega(n^{\varepsilon})$$ for some constant $\varepsilon > 0$

i.e., $f(n)$ grows polynomialy slower than $n^{\log_b a}$ (by an n^{ε} factor).

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

The Master Method: Case 2 (simple version)

- Recurrence: $T(n) = aT(n/b) + f(n)$

Case 2: $\dfrac{f(n)}{n^{\log_b a}} = \Theta(1)$

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

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

The Master Method: Case 3

Case 3: $\dfrac{f(n)}{n^{\log_b a}} = \Omega(n^{\varepsilon})$ for some constant $\varepsilon > 0$

i.e., $f(n)$ grows polynomialy faster than $n^{\log_b a}$ (by an n^{ε} factor).

and the following regularity condition holds:

$a\,f(n/b) \leq c\,f(n)$ for some constant $c < 1$

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

Example: T(n) = 4T(n/2) + n

a = 4

b = 2

f(n) = n

$$n^{\log_b a} = n^2$$

f(n) grows *polynomially* slower than $n^{\log_b a}$

$$\frac{n^{\log_b a}}{f(n)} = \frac{n^2}{n} = n = \Omega(n^{\varepsilon})$$

for ε = 1

CASE 1

T(n) = Θ($n^{\log_b a}$)

T(n) = Θ(n^2)

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

$a = 4$

$b = 2$

$f(n) = n^2$

$n^{\log_b a} = n^2$

$\Rightarrow$ $f(n)$ grows at similar rate as $n^{\log_b a}$

$f(n) = \Theta(n^{\log_b a}) = n^2$

$\Rightarrow$ CASE 2

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

$$\boxed{T(n) = \Theta(n^2 \lg n)}$$

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

$a = 4$

$b = 2$

$f(n) = n^3$

$n^{\log_b a} = n^2$

f(n) grows *polynomially* faster than $n^{\log_b a}$

$$\frac{f(n)}{n^{\log_b a}} = \frac{n^3}{n^2} = n = \Omega(n^{\varepsilon})$$

for $\varepsilon = 1$

seems like CASE 3, but need to check the regularity condition

Regularity condition: $a\,f(n/b) \le c\,f(n)$ for some constant $c < 1$

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

CASE 3

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

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

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

$a = 4$

$b = 2$

$f(n) = n^2/\lg n$

$$n^{\log_b a} = n^2$$

f(n) grows slower than $n^{\log_b a}$

but is it polynomially slower?

$$\frac{n^{\log_b a}}{f(n)} = \frac{n^2}{\dfrac{n^2}{\lg n}} = \lg n \neq \Omega(n^{\varepsilon})$$

for any $\varepsilon > 0$

is **not** CASE 1

Master method does not apply!

The Master Method: Case 2 (general version)

- Recurrence: $T(n) = aT(n/b) + f(n)$

Case 2: $$\frac{f(n)}{n^{\log_b a}} = \Theta(\lg^k n)$$ for some constant $k \geq 0$

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

General Method (Akra-Bazzi)

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

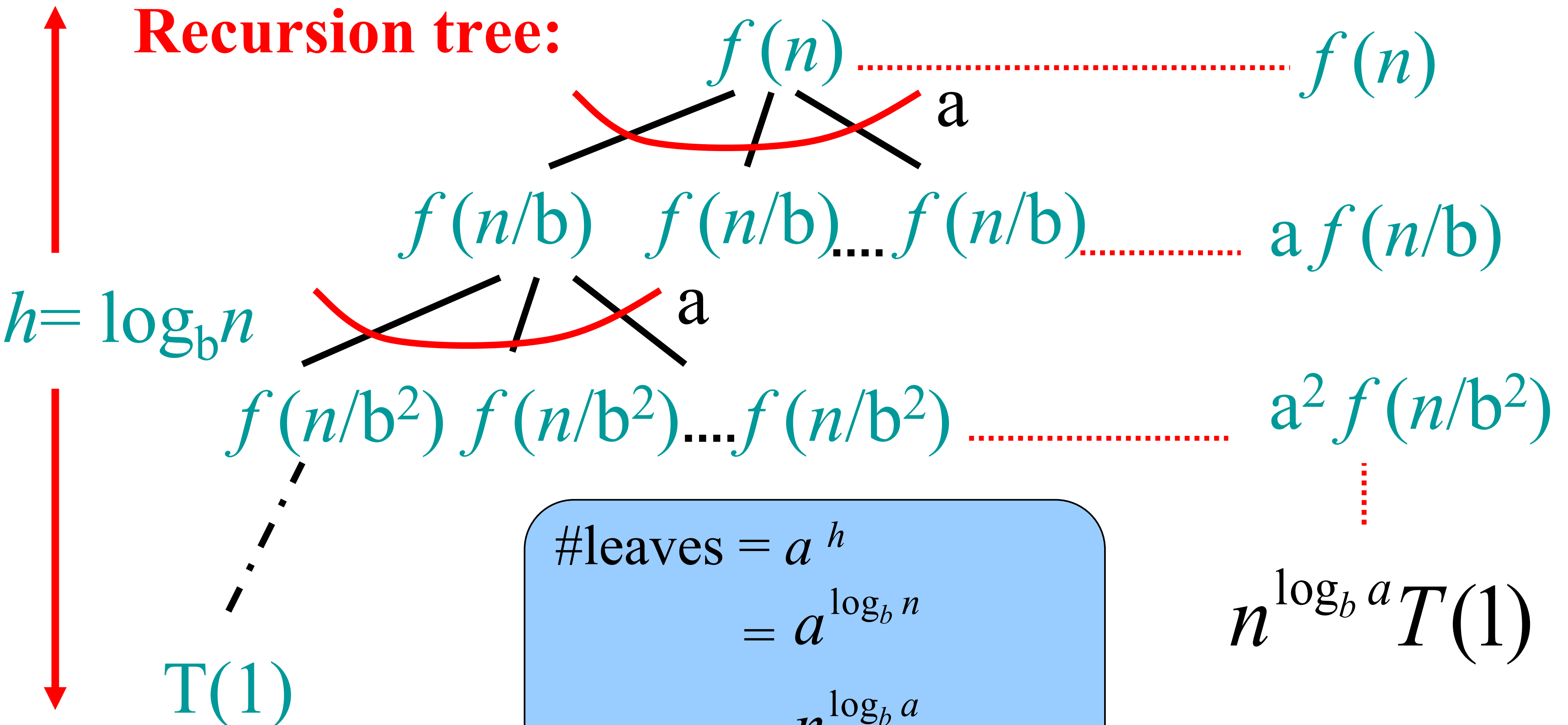
Let p be the unique solution to

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

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

Recursion tree:

$f(n)$ .... $f(n)$

a

$f(n/\text{b})$ $f(n/\text{b})$ .... $f(n/\text{b})$ .... $\text{a} f(n/\text{b})$

a

$h = \log_{\text{b}} n$

$f(n/\text{b}^2)$ $f(n/\text{b}^2)$ .... $f(n/\text{b}^2)$ .... $\text{a}^2 f(n/\text{b}^2)$

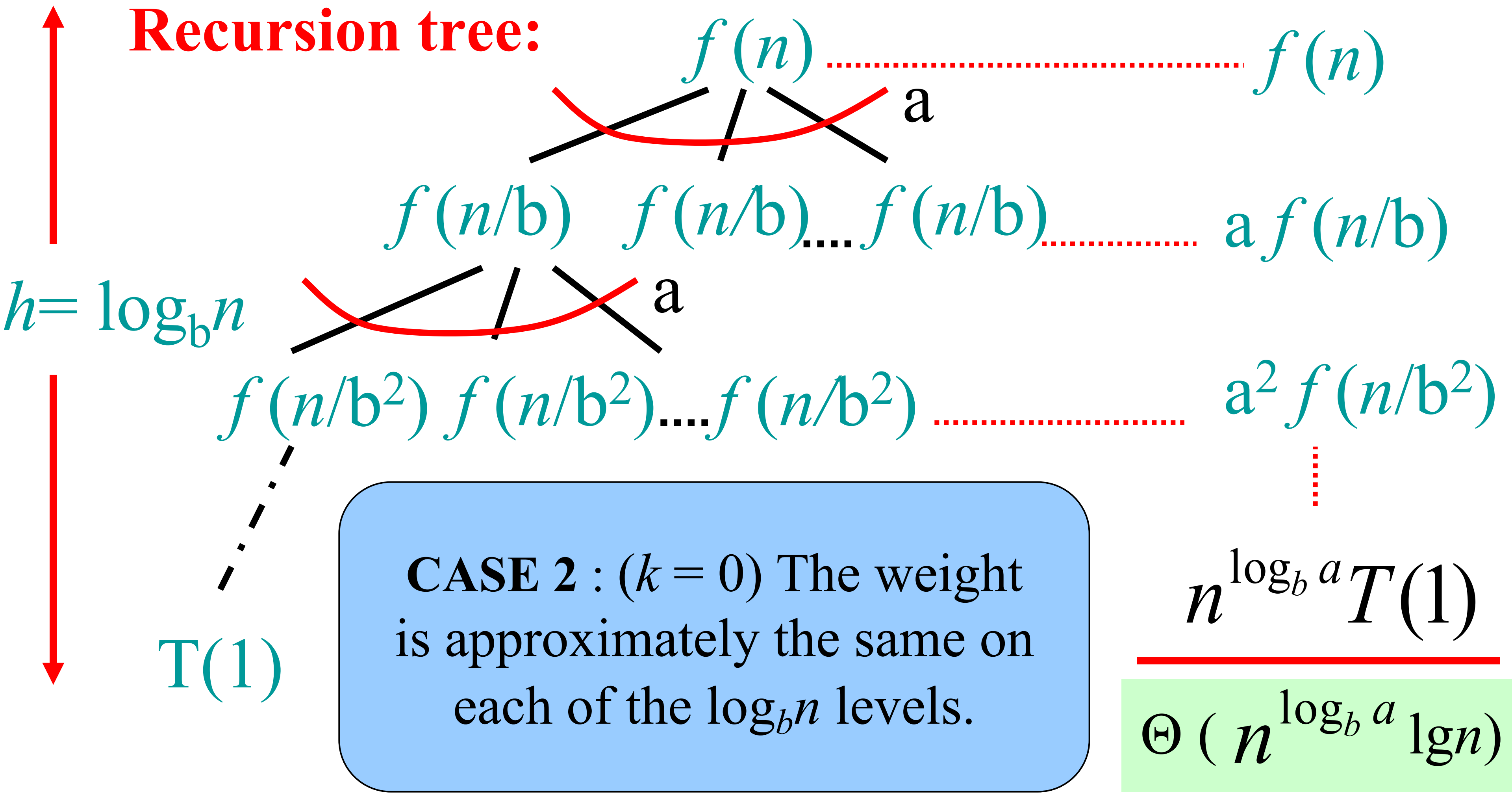
T(1)

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

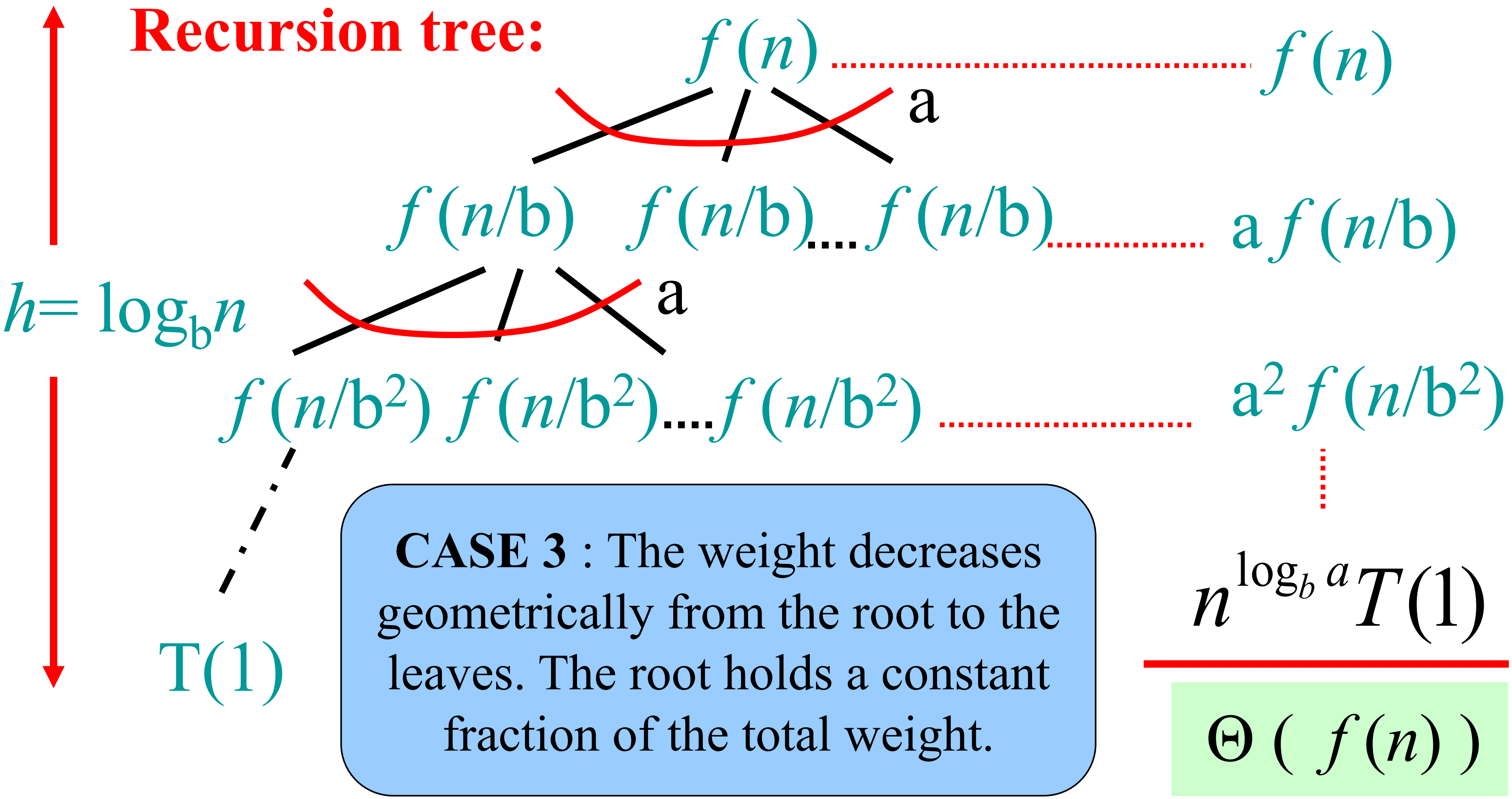
$$n^{\log_b a} T(1)$$

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

Idea of Master Theorem



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)

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

Proof of Case 1

- $$\frac{n^{\log_b a}}{f(n)} = \Omega(n^{\varepsilon})$$ 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')

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

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

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.