

CSE214 – Analysis of Algorithms

PhD Furkan Gözükar, Toros University

https://github.com/FurkanGozukara/CSE214_2018

Lecture 3

Solving Recurrences

*Based on Cevdet Aykanat's and Mustafa Ozdal's Lecture
Notes - Bilkent*

Solving Recurrences

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

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

Kod Satırı	Maliyet
<code>mergeSort(A, left, right) {</code>	$T(n)$
<code>if (left < right) {</code>	$\Theta(1)$
<code>mid = floor((left + right) / 2);</code>	$\Theta(1)$
<code>MergeSort(A, left, mid);</code>	$T(n/2)$
<code>MergeSort(A, mid+1, right);</code>	$T(n/2)$
<code>Merge(A, left, mid, right);</code>	$\Theta(n)$
<code>}</code>	

- $T(n) = \Theta(1)$, $n = 1$ için

$$T(n) = 2T(n/2) + f(n) , n > 1 \text{ için}$$

- Özyineli bir denklemdir

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
-

Remembering How Mathematical Induction Works

Consider an infinite sequence of dominoes, labeled $1, 2, 3, \dots$, where each domino is standing.

Let $P(n)$ be the proposition that the n th domino is knocked over.



We know that the first domino is knocked down, i.e., $P(1)$ is true .

We also know that if whenever the k th domino is knocked over, it knocks over the $(k + 1)$ st domino, i.e, $P(k) \rightarrow P(k + 1)$ is true for all positive integers k .

Hence, all dominos are knocked over.

$P(n)$ is true for all positive integers n .

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:

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

Substitution Method: Example – cont'd

- To prove

$$T(n) \leq cn^3$$

- 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
2. Inductive hypothesis for smaller sizes
3. Prove the general case

*haven't proved
the base cases yet*

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:

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

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. $T(n) \leq cn^2$ for all $n \geq n_0$
 - Ind. hyp: Assume that $T(k) \leq ck^2$ for $k < n$
 - Prove the general case: $T(n) \leq cn^2$
-

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

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

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

$$cn^2 + n \leq cn^2$$

$$= O(n^2)$$

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: $T(n) \leq c_1n^2 - c_2n$

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

$$\leq 4(c_1(n/2)^2 - c_2(n/2)) + n$$

$$= c_1n^2 - 2c_2n + n$$

$$c_1n^2 - 2c_2n + n \leq c_1n^2 - c_2n$$

$$n - c_2n \leq 0$$

$$\text{for } n(c_2 - 1) \geq 0$$

$$n(1 - c_2) \leq 0$$

$$\text{choose } c_2 \geq 1$$

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 (i.e. $n = n_0$)

We can choose c_1 large enough to make this hold

e.g. $c_1 = 2, c_2 = 1, n = 1$

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

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

since $n > 0$

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. $c=1, n=1$)

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)

$$T(1) = 4 \quad T(\lfloor n \rfloor) = 2T(\lfloor n/2 \rfloor) + 4\lfloor n \rfloor$$

Build Solution

$$\begin{aligned} T(n) &= 2T(n/2) + 4n \\ &= 2\left[2T(n/2^2) + 4\frac{n}{2}\right] + 4n \\ &= 2^2 T(n/2^2) + 4n + 4n \\ &= 2^2 \left[2T(n/2^3) + 4\frac{n}{2^2}\right] + 4n + 4n \\ &= 2^3 T(n/2^3) + 4n + 4n + 4n \\ &= 2^3 \left[2T(n/2^4) + 4\frac{n}{2^3}\right] + 4n + 4n + 4n \\ &= 2^4 T(n/2^4) + 4n + 4n + 4n + 4n \\ &= 2^i T(n/2^i) + i(4n) \end{aligned}$$

$$\frac{n}{2^i} = 1 \Rightarrow n = 2^i \Rightarrow \log_2 n = i$$

Expand Scratch

$$T(\lfloor n/2 \rfloor) = 2T(\frac{n}{2^2}) + 4(\frac{n}{2})$$

$$T(\frac{n}{2^2}) = 2T(\frac{n}{2^3}) + 4(\frac{n}{2^2})$$

$$T(\frac{n}{2^3}) = 2T(\frac{n}{2^4}) + 4\frac{n}{2^3}$$

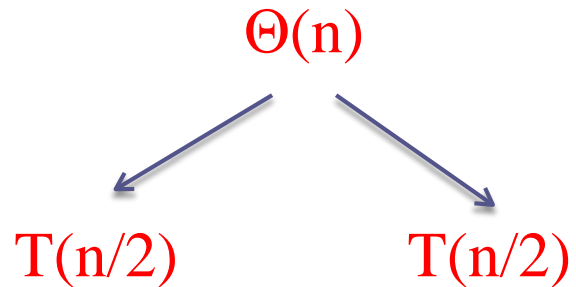
$$2^{\log_2 n} = n^{\log_2 2} = n$$

$$\begin{aligned} &\Rightarrow 2^{\log_2 n} T(1) + (\log_2 n)(4n) \\ &= n(4) + 4n \log_2 n \\ &= 4n + 4n \log_2 n = \Theta(n \log n) \end{aligned}$$

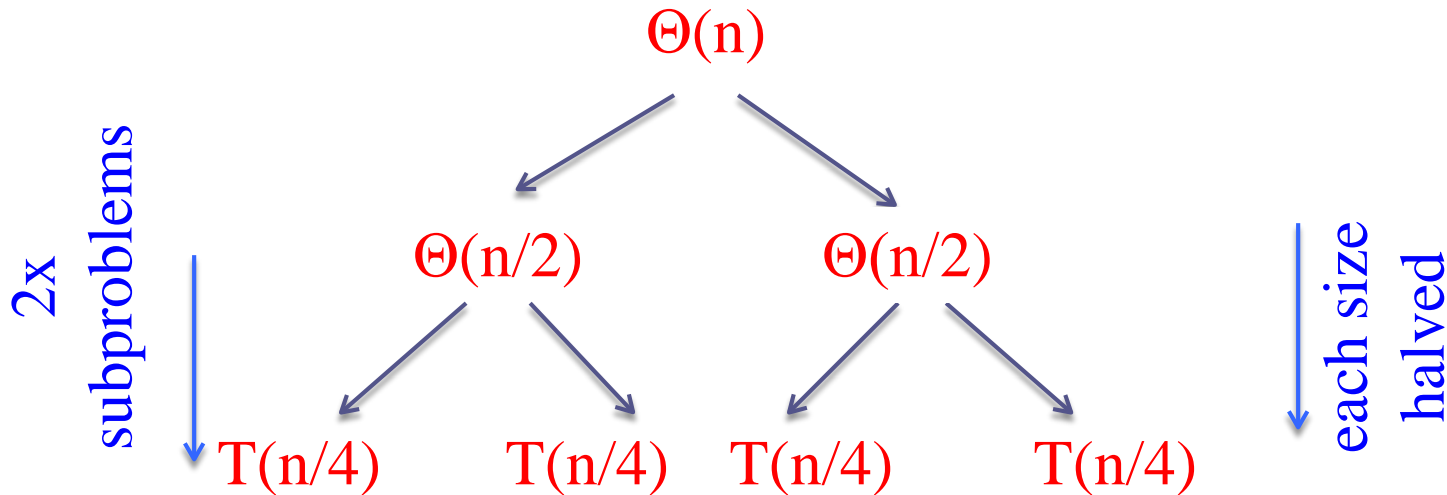
Recursion Tree Method

- 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.
-

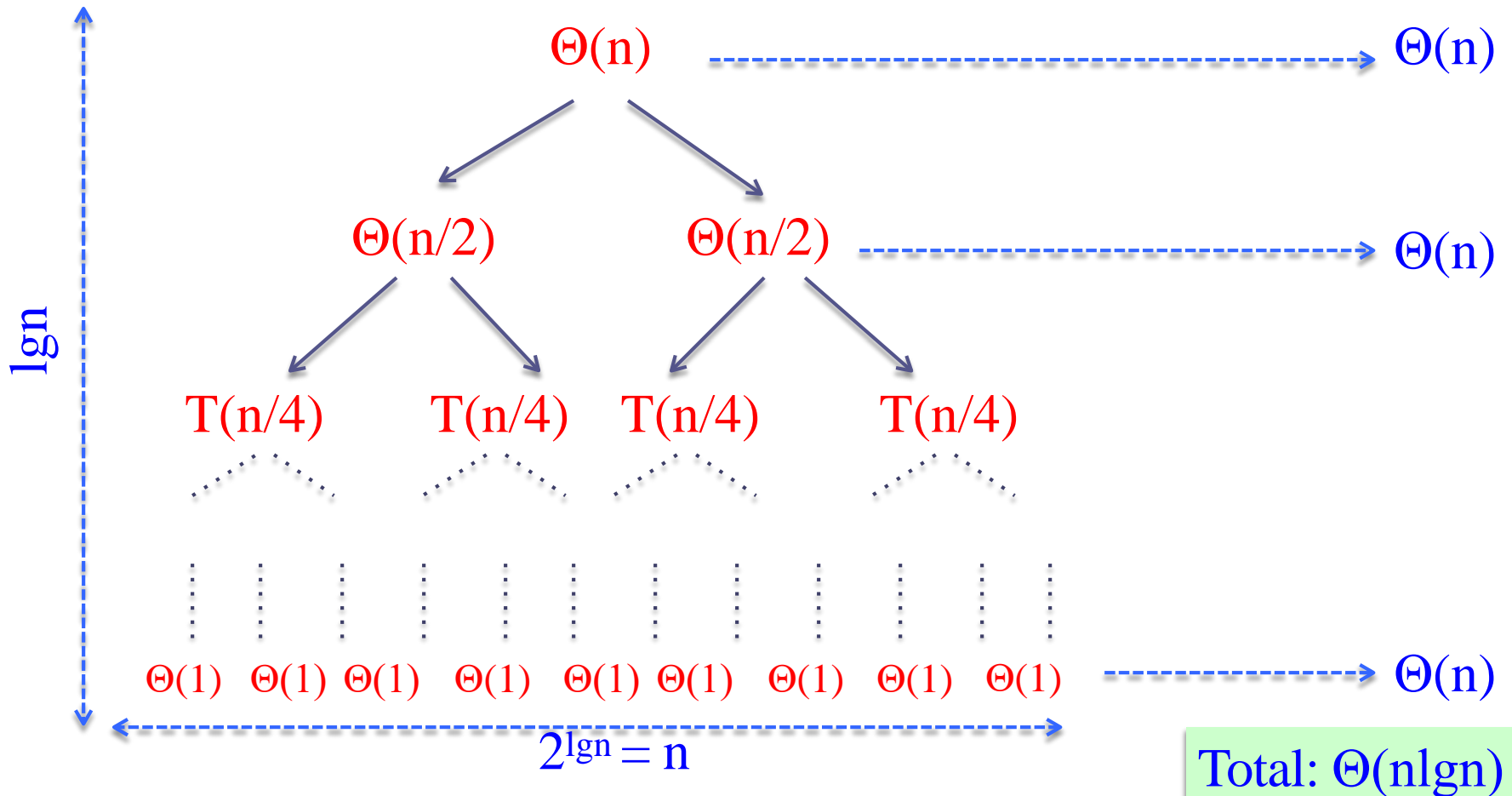
Solve Recurrence: $T(n) = 2T(n/2) + \Theta(n)$



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



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



Example of Recursion Tree

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

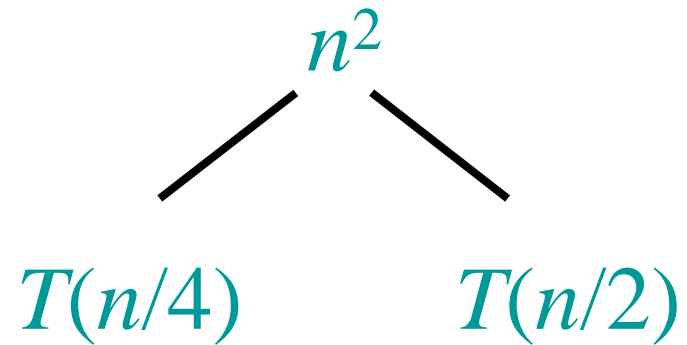
Example of Recursion Tree

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

$$T(n)$$

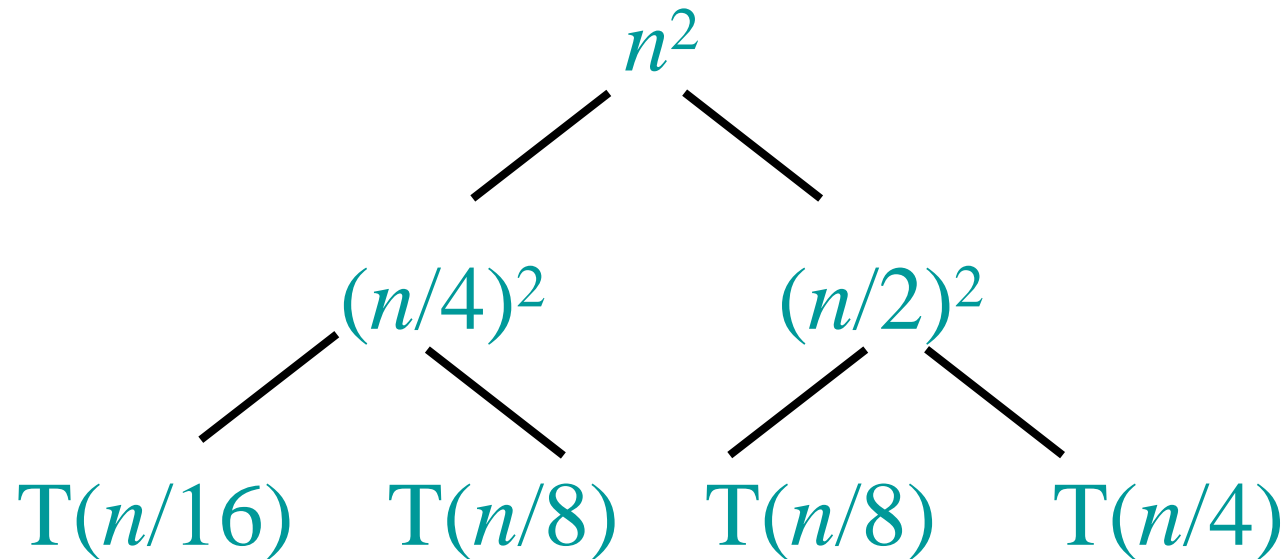
Example of Recursion Tree

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



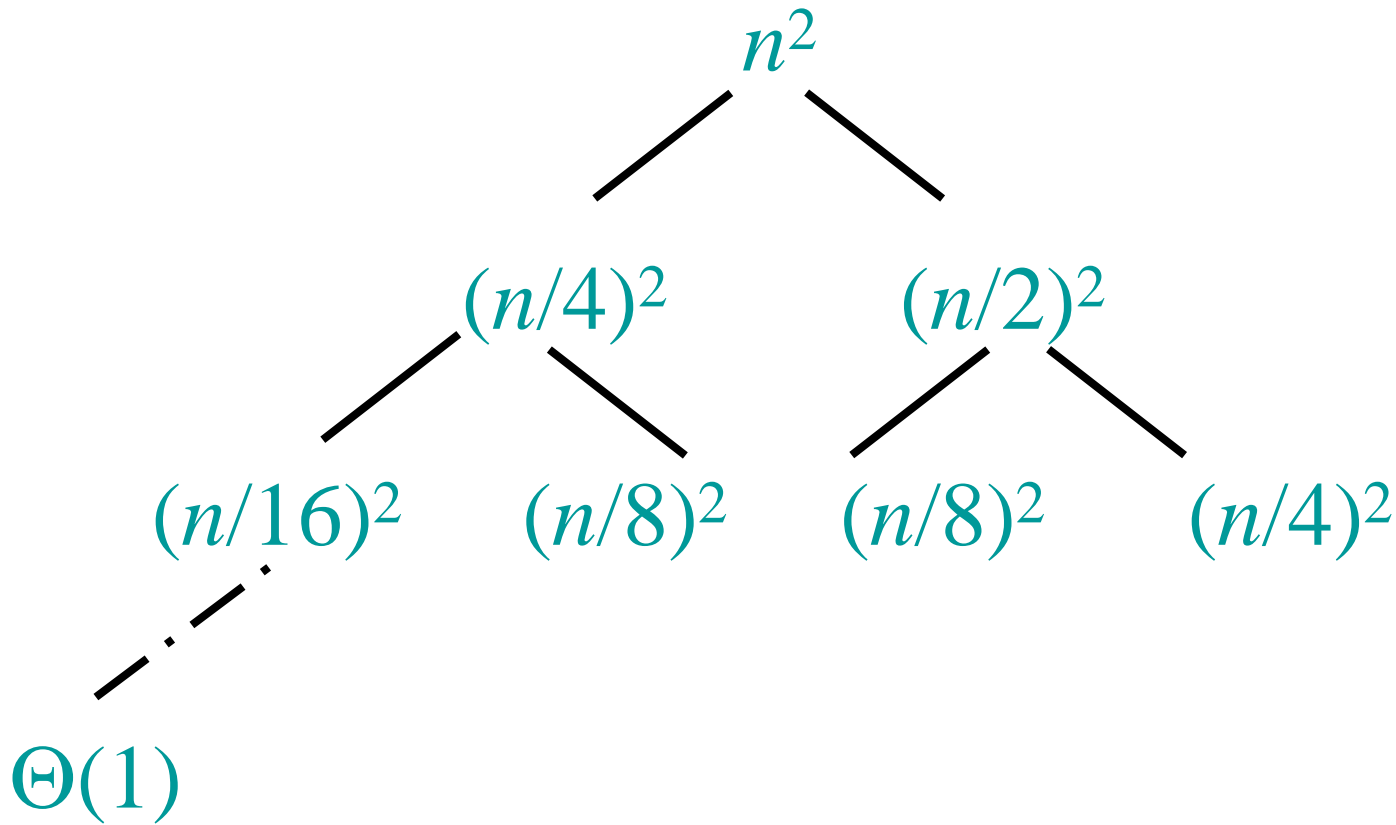
Example of Recursion Tree

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



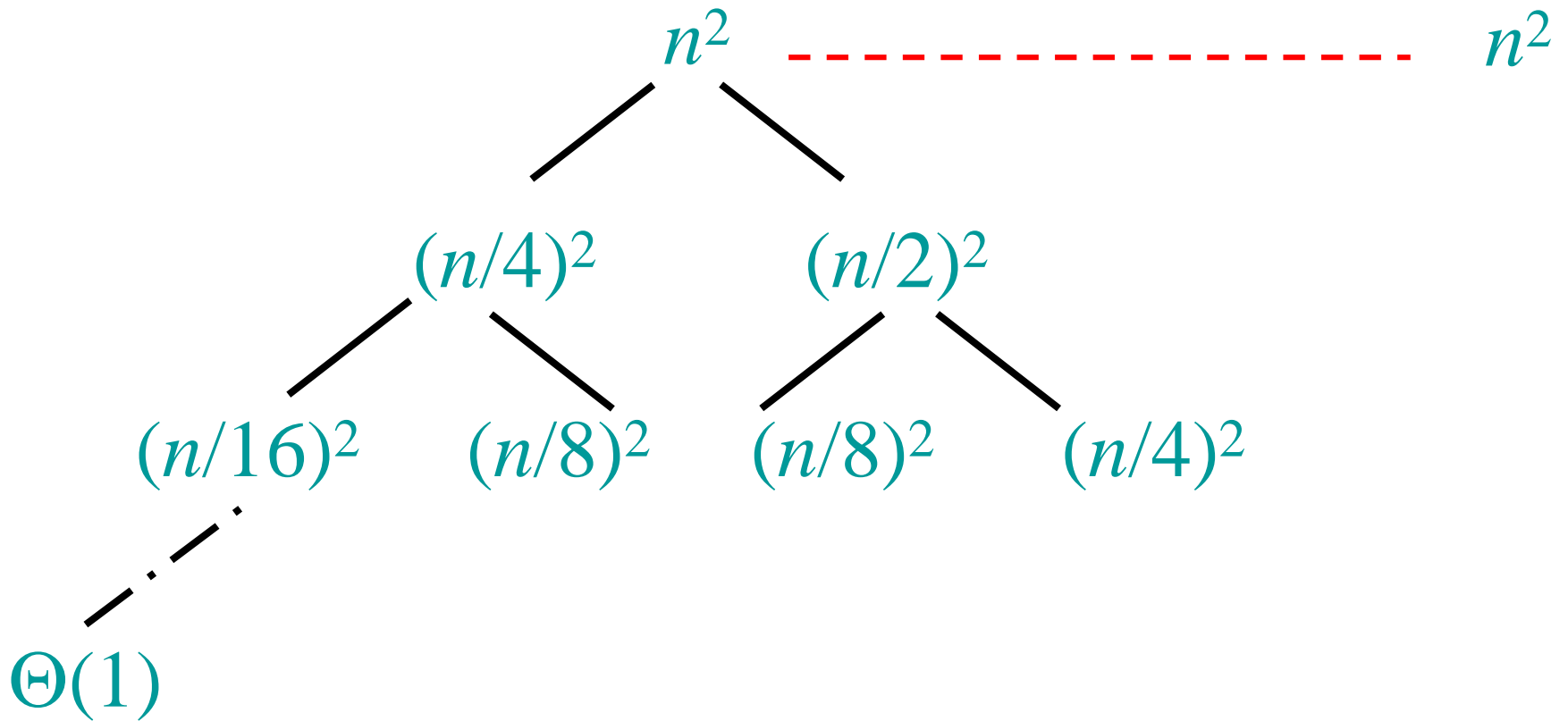
Example of Recursion Tree

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



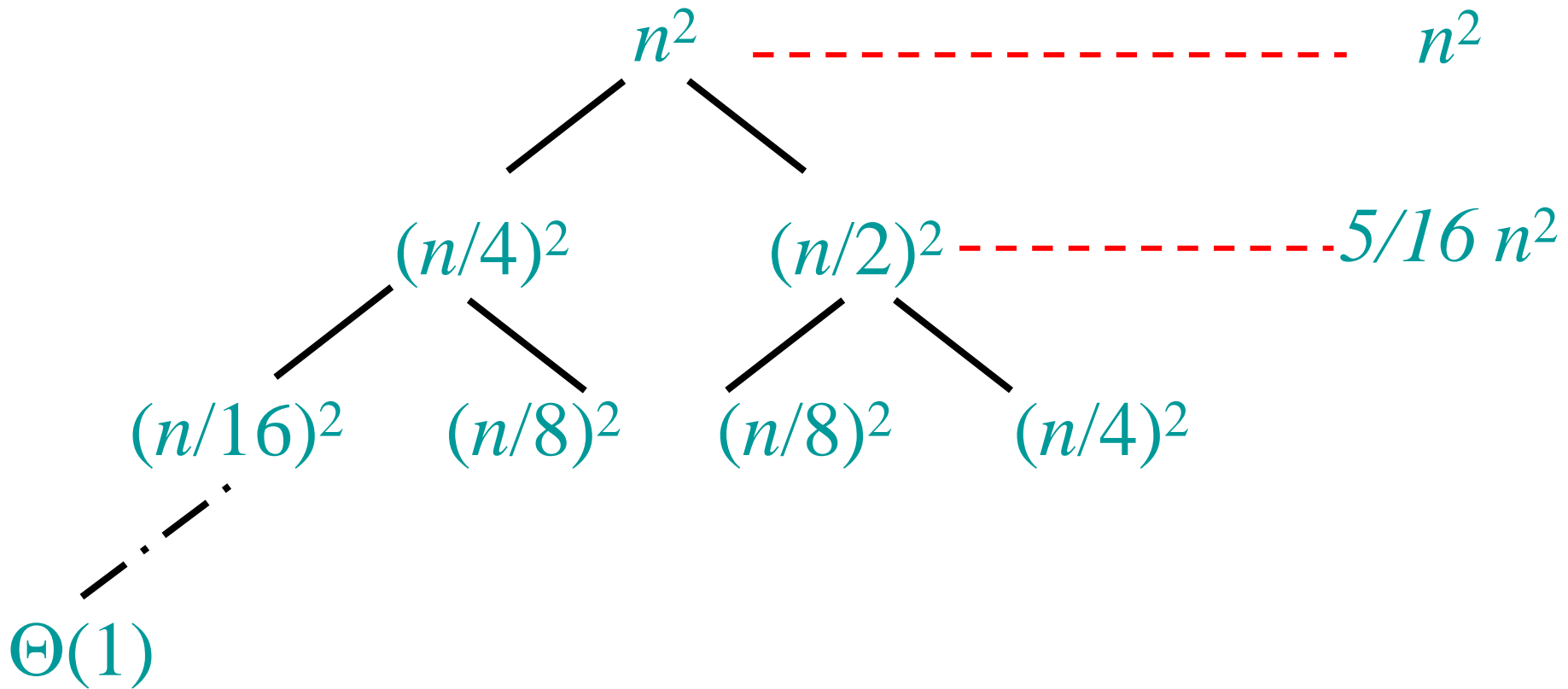
Example of Recursion Tree

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



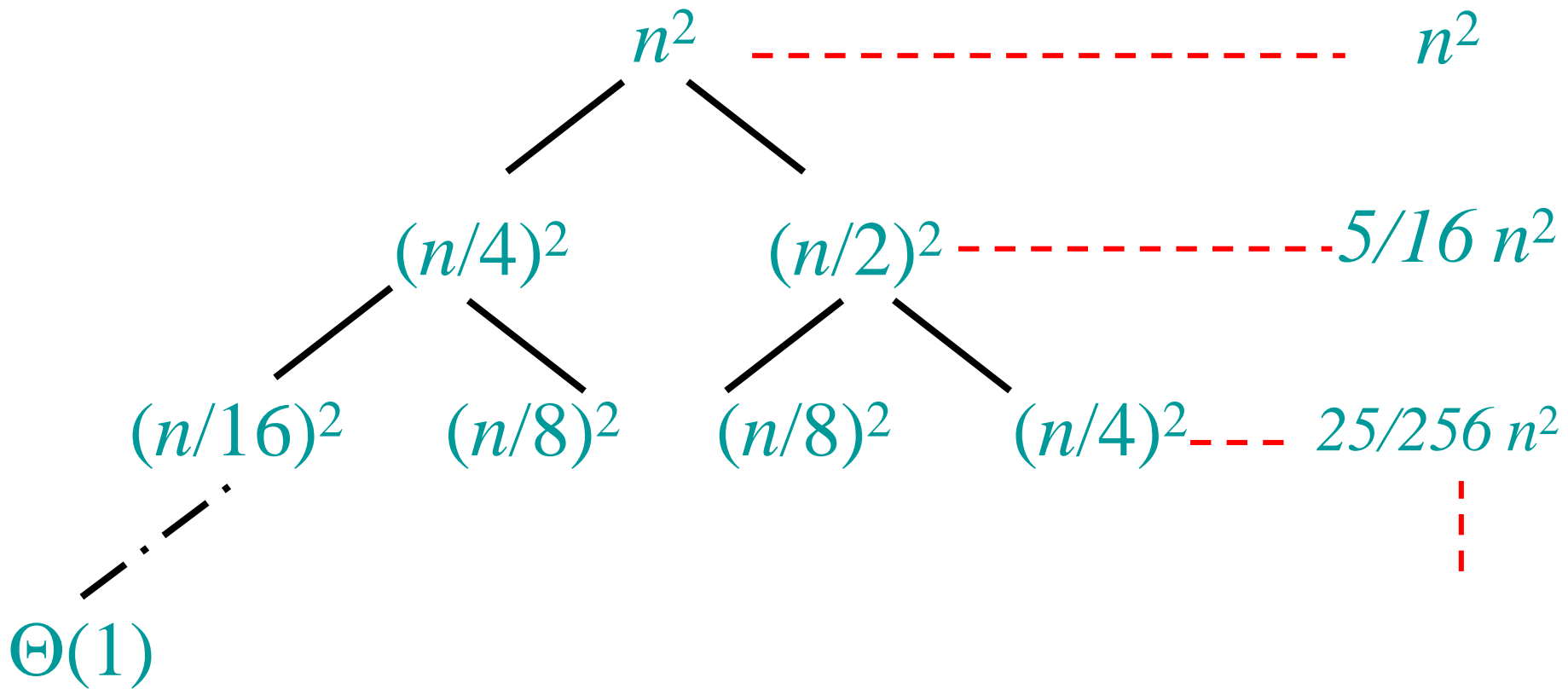
Example of Recursion Tree

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



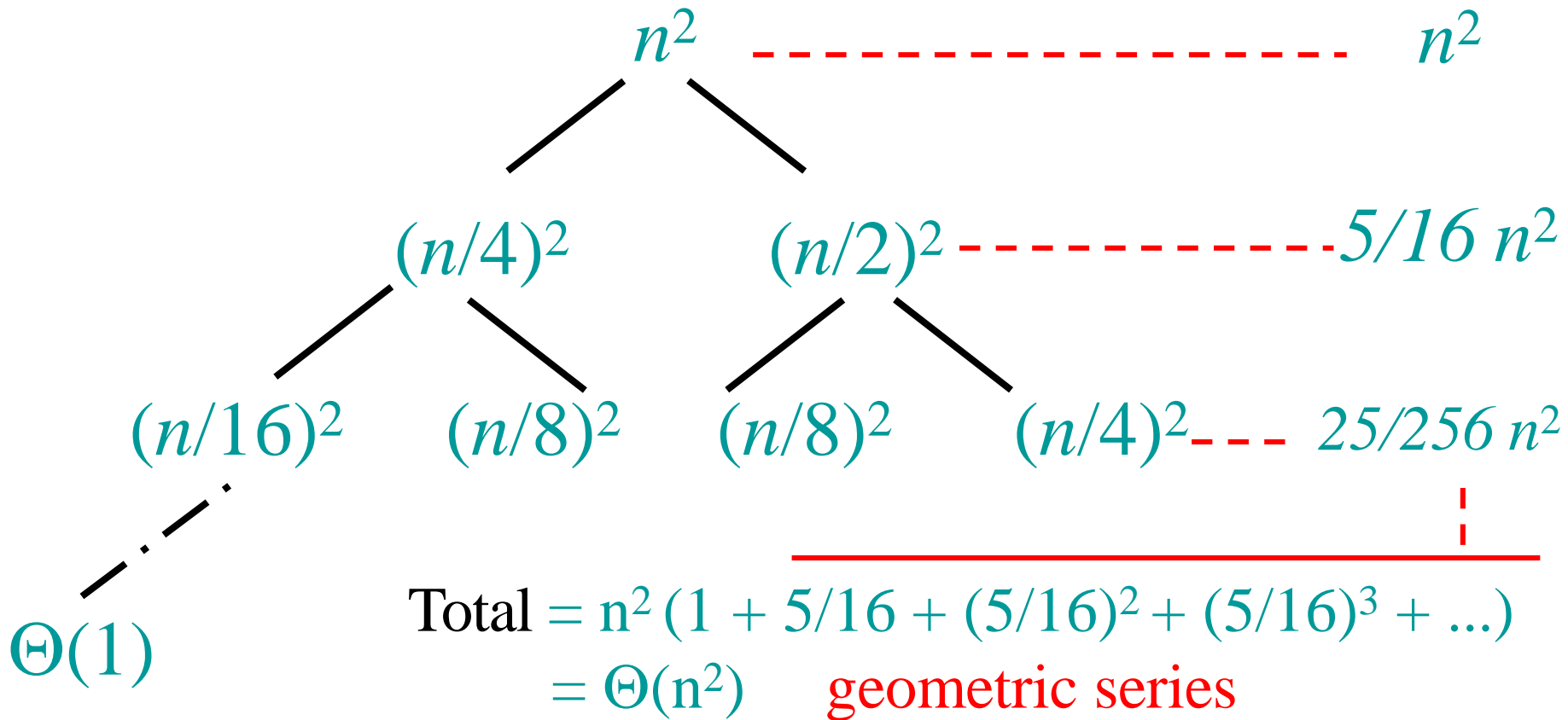
Example of Recursion Tree

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



Example of Recursion Tree

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



Geometric Series Reminder

$$1 + x + x^2 + \cdots = \frac{1}{1-x} \quad \text{for } |x| < 1$$

$$1 + x + x^2 + \cdots + x^n = \frac{1 - x^{n+1}}{1-x} \quad \text{for } x \neq 1$$

The Master Method

- A powerful black-box method to solve recurrences (solving divide and conquer type problems).
- The master method is utilized for algorithms that divides problem into a times n/b size pieces

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

where $a \geq 1$, $b > 1$, and f is asymptotically positive.

The Master Method

- The cost of every step is calculated (e.g. dividing cost, merge cost of the pieces, etc.) and sum of these costs are shown as $f(n)$
 - Then the master method formula is used to have guess of the run time cost of the algorithm
-

Logaritma Nedir? Logaritma Formülleri Özellikleri

Logaritma Tanımı: $a, b \in \mathbb{R}^+$ ve $a \neq 1$ olmak üzere $a^x = b$ denklemini sağlayan x sayısına $\log_a b$ denir ve b 'nin a tabanında logaritması diye okunur.

1) $\log_a x = b$ ise $x = a^b$ $\log_2 8 = 3 \mid 8 = 2^3$

2) $\log_a (A \cdot B) = \log_a A + \log_a B$ $\log_2 (4 \cdot 8) = \log_2 (32) = 5 = \log_2 (4) + \log_2 (8) = 2 + 3$

3) $\log_a (A/B) = \log_a A - \log_a B$ $\log_2 (16/4) = \log_2 (4) = 2 = \log_2 (16) - \log_2 (4) = 4 - 2 = 2$

4) $\log_a A^n = n \cdot \log_a A$ $\log_2 8^2 = \log_2 64 = 6 = 2 \cdot \log_2 8 = 2 \cdot 3 = 6$

5) $\log_{a^m} A^n = \frac{n}{m} \log_a A$ $\log_{2^3} 8^2 = \log_8 64 = 2 = (2/3) \cdot \log_2 8 = 2/3 \cdot 3 = 2$

6) $\log_{(a^n)} x = \frac{1}{n} \cdot \log_a x$ $\log_{2^3} 8 = \log_8 8 = 1 = (1/3) \cdot \log_2 8 = 1/3 \cdot 3 = 1$

7) $\log_a x = (\log_b x) / (\log_b a)$ [taban değiştirme] $\log_4 16 = 2 = \log_2 16 - \log_2 4 = 4 - 2 = 2$

8) $a^{\log_a x} = x$ $2^{\log_2 8} = 2^3 = 8 = 8$

9) $\log_a \sqrt[n]{A} = \frac{1}{n} \log_a A$ $\log_2 \sqrt[3]{8} = \log_2 2 = 1 = \left(\frac{1}{3}\right) \cdot \log_2 8 = \frac{1}{3} \cdot 3 = 1$

10) $\log_{1/a} x = -\log_a x$

$$\log_{1/2} 8 = -\log_2 8 = -3 \mid 8 = \left(\frac{1}{2}\right)^{-3}$$

11) $\log_a b \cdot \log_b c \cdot \log_c d = \log_a d$ $\log_2 4 \cdot \log_4 16 \cdot \log_{16} 256 = 2 \cdot 2 \cdot 2 = 8 = \log_2 256$

12) $\log_a b = 1 / \log_b a$ veya $\log_a b \cdot \log_b a = 1$

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

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

*i.e., $f(n)$ grows polynomially 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: $\frac{f(n)}{n^{\log_b a}} = \Theta(1)$

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: $\frac{f(n)}{n^{\log_b a}} = \Omega(n^\epsilon)$ for some constant $\epsilon > 0$

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

and the following regularity condition holds:

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

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 $n^{\log_b a}$


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

for $\epsilon = 1$

 CASE 1

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

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

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

$$a = 4$$

$$b = 2$$

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

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



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

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



CASE 2



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

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

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 $n^{\log_b a}$

$$\frac{f(n)}{n^{\log_b a}} = \frac{n^3}{n^2} = n = \Omega(n^\epsilon) \quad \text{for } \epsilon = 1$$

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

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

$$4 (n/2)^3 \leq c n^3 \text{ for } c = 1/2$$

CASE 3

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

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?

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

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

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

$$a = 4$$

$$b = 2$$

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

Solution:

$$\frac{\frac{n^2}{\log n}}{n^{\log_2 4}} = \frac{\frac{n^2}{\log n}}{n^2} = \frac{1}{\log n} = \log_n 2 = \log^{-1} n$$

$$T(n) = \Theta (n^{\log_2 4} \lg^{1+1} n)$$

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

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

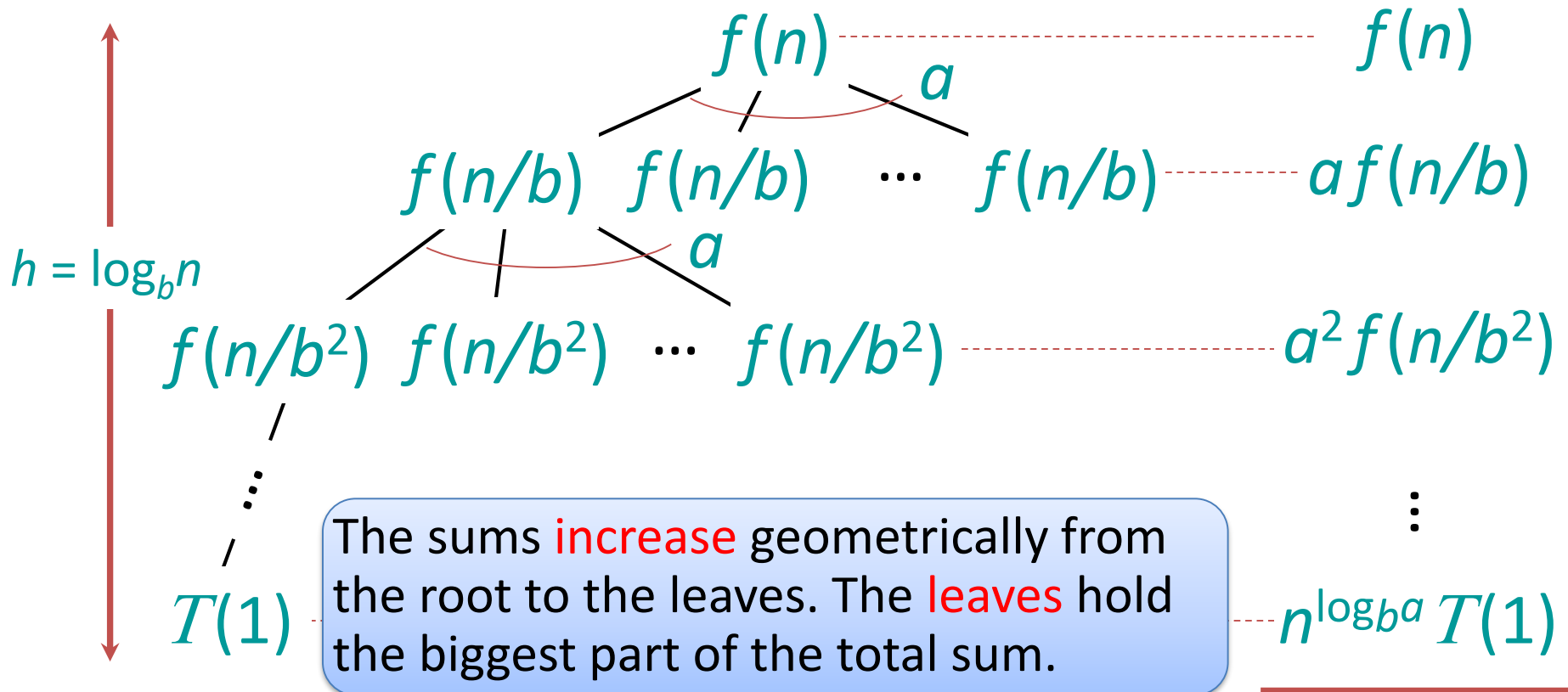
$$\sum_{i=1}^k (a_i / b_i^p) = 1$$

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.)

$f(n)$ is smaller than leaf sum.

Case 1 Explained

$$f(n) < O(n^{\log_b a})$$

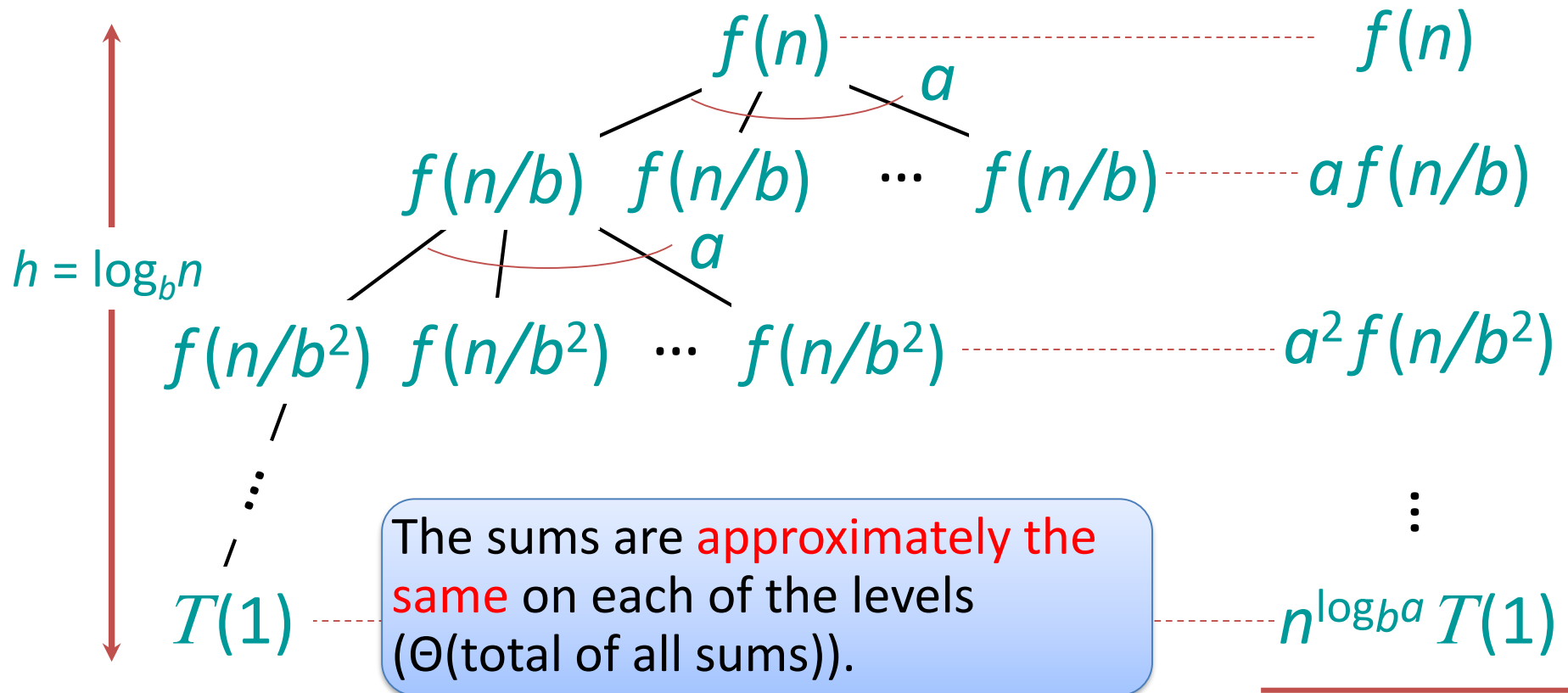


$$\Theta(n^{\log_b a})$$

$f(n)$ is roughly equal to the leaf sum.

Case 2 Explained

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

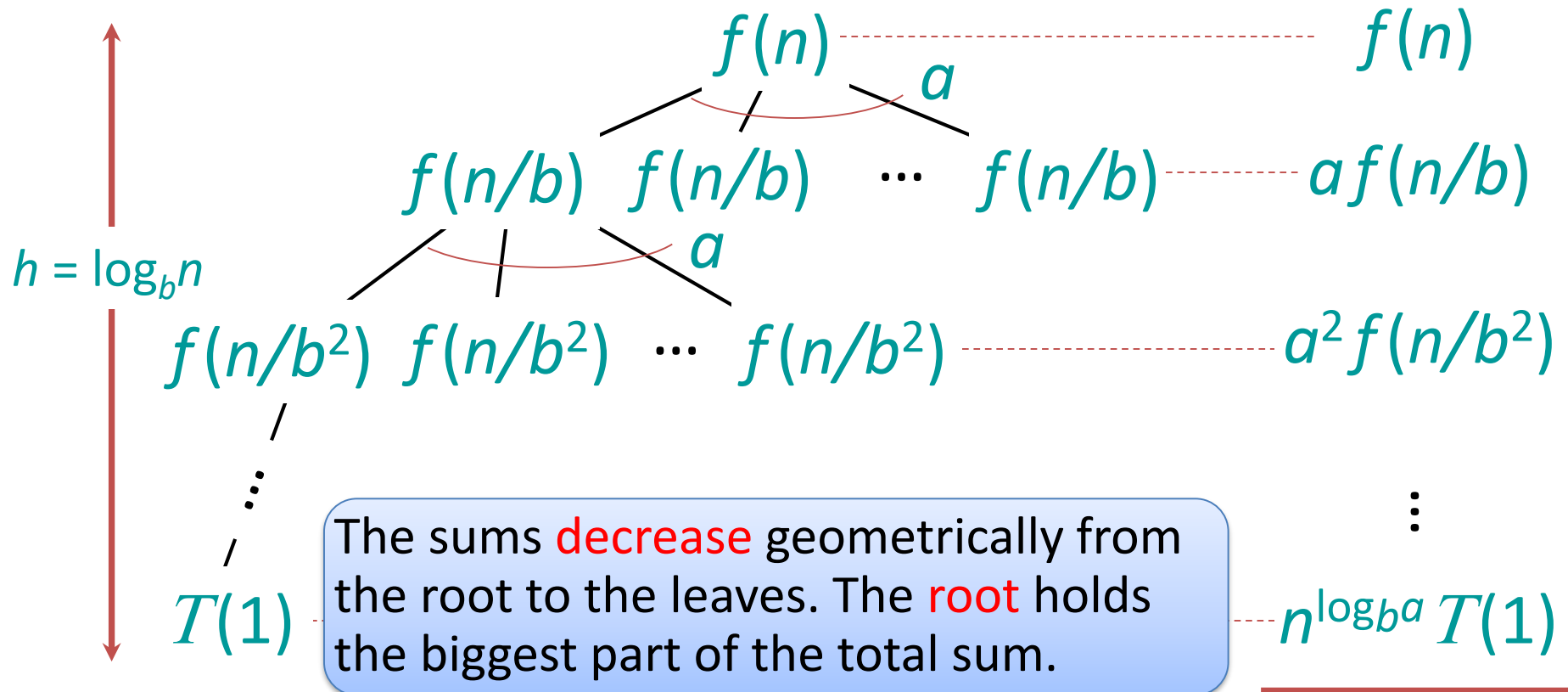


$$\Theta(n^{\log_b a} * \log n)$$

$a f(n/b)$ is getting smaller
at lower levels (see defⁿ)

Case 3 Explained

$$f(n) > O(n^{\log_b a})$$



$$\Theta(f(n))$$

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

$$\blacktriangleright \frac{n^{\log_b a}}{f(n)} = \Omega(n^\varepsilon) \quad \text{for some } \varepsilon > 0$$

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

$$\blacktriangleright g(n) = \sum_{i=0}^{h-1} a^i O\left((n / b^i)^{\log_b a - \varepsilon}\right) = O\left(\sum_{i=0}^{h-1} a^i (n / b^i)^{\log_b a - \varepsilon}\right)$$

$$\blacktriangleright = O\left(n^{\log_b a - \varepsilon} \sum_{i=0}^{h-1} a^i b^{i\varepsilon} / b^{i \log_b a}\right)$$

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')

$$\begin{aligned} - \quad g(n) &= O\left(n^{\log_b a - \varepsilon} O(n^\varepsilon)\right) = O\left(\frac{n^{\log_b a}}{n^\varepsilon} O(n^\varepsilon)\right) \\ &= O(n^{\log_b a}) \end{aligned}$$

$$\begin{aligned} - \quad T(n) &= \Theta(n^{\log_b a}) + g(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a}) \\ &= \Theta(n^{\log_b a}) \end{aligned}$$

Q.E.D.

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

$$\frac{f(n)}{n^{\log_b a}} = \Theta(\lg^0 n) = \Theta(1) \Rightarrow f(n) = \Theta(n^{\log_b a}) \Rightarrow f(n/b^i) = \Theta\left(\left(\frac{n}{b^i}\right)^{\log_b a}\right)$$

$$\therefore g(n) = \sum_{i=0}^{\log_b n - 1} a^i \Theta\left(\left(\frac{n}{b^i}\right)^{\log_b a}\right)$$

$$= \Theta\left(\sum_{i=0}^{\log_b n - 1} a^i \frac{n^{\log_b a}}{b^{i \log_b a}}\right) = \Theta\left(n^{\log_b a} \sum_{i=0}^{\log_b n - 1} a^i \frac{1}{(b^{\log_b a})^i}\right) = \Theta\left(n^{\log_b a} \sum_{i=0}^{\log_b n - 1} a^i \frac{1}{a^i}\right)$$

$$= \Theta\left(n^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1\right) = \Theta\left(n^{\log_b a} \log_b n\right) = \Theta\left(n^{\log_b a} \lg n\right)$$

$$T(n) = n^{\log_b a} + \Theta\left(n^{\log_b a} \lg n\right)$$

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

Q.E.D.