

Algorithms

Kiho Choi

Fall, 2022

Department of AI·Software
Gachon University

A decorative graphic in the bottom right corner consisting of a blue curved shape filled with various-sized circles in different shades of blue, creating a bubble-like or cellular pattern.

2. Growth of Functions & Divide-and-Conquer

Contents

- Asymptotic notation
 - Asymptotic notation in equations
 - Comparing functions
- Recursion-tree method
- The master method

- Problem 3: Thanksgiving trip

Growth of Functions

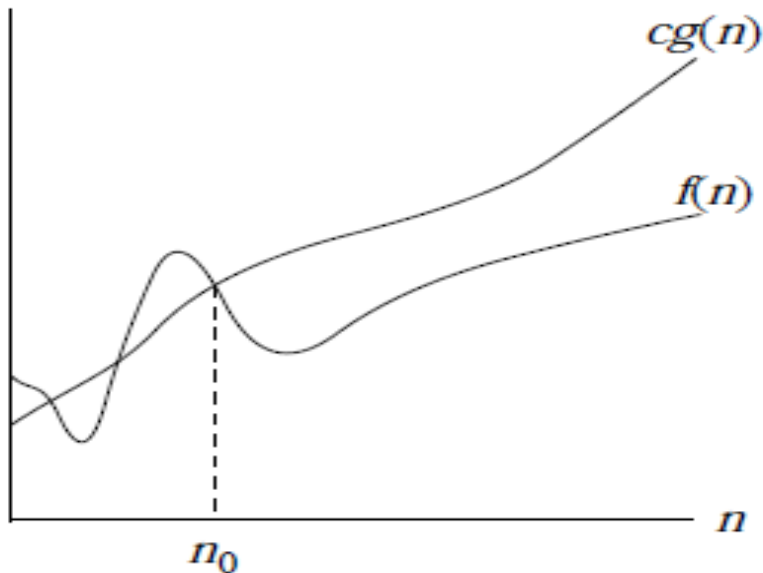
overview

- A way to describe behavior of functions in the limit.
We're studying asymptotic efficiency.
- Describe growth of functions.
- Focus on what's important by abstracting away low-order terms and constant factors.
- How we indicate running times of algorithms.
- A way to compare "sizes" of functions:

$$\begin{array}{lll} O & \approx & \leq \\ \Omega & \approx & \geq \\ \Theta & \approx & = \\ o & \approx & < \\ \omega & \approx & > \end{array}$$

O -notation

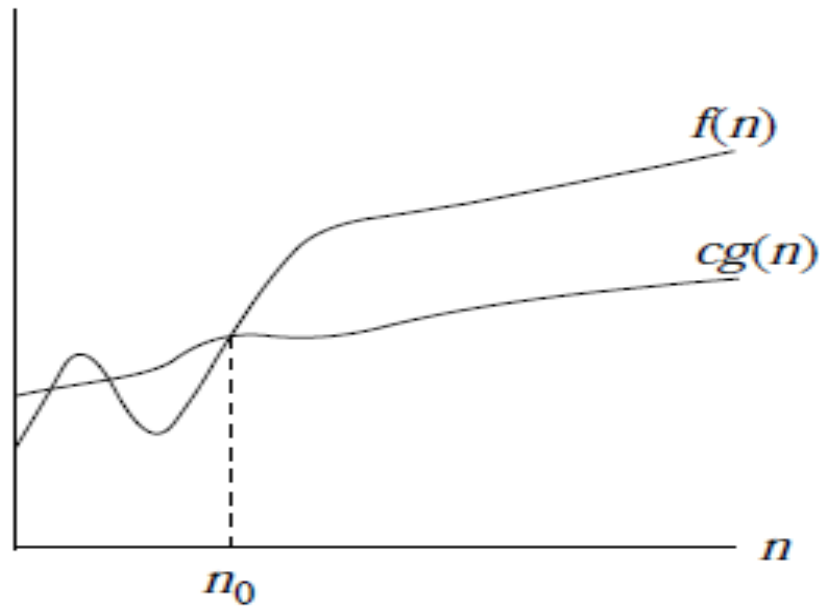
$O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c g(n) \text{ for all } n \geq n_0 \}$



$g(n)$ is an asymptotic **upper bound** for $f(n)$.

Ω - notation

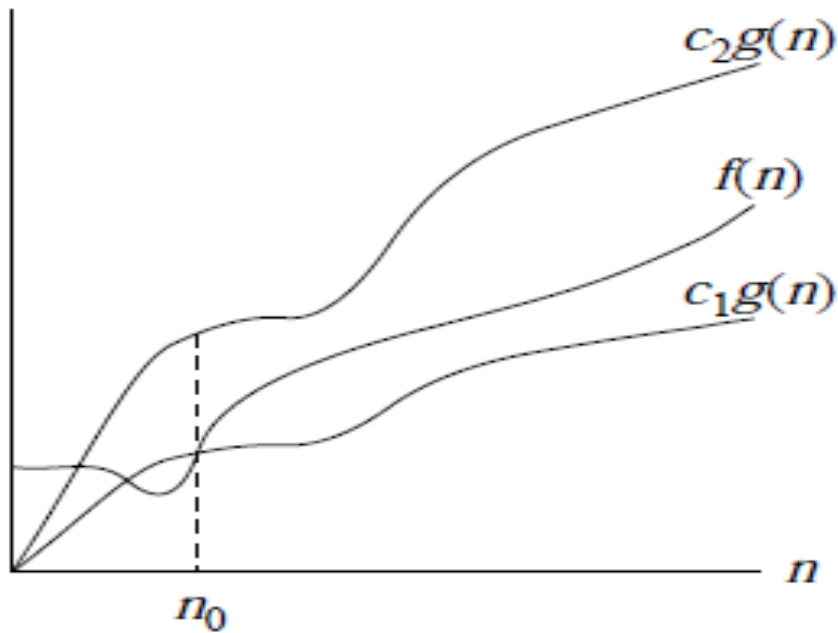
$\Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c$
and n_0 such that $0 \leq c g(n) \leq f(n)$
for all $n \geq n_0 \}$



*$g(n)$ is an asymptotic
lower bound for $f(n)$.*

Θ - notation

$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}$



$g(n)$ is an asymptotically **tight bound** for $f(n)$.

Θ - notation

- ***Engineering:***

- Drop low-order terms; ignore leading constants.

- Example : $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

o - notation

$o(g(n))$: $\{f(n) : \text{for all constants } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}$.

Another view, probably easier to use: $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$.

$$n^{1.9999} = o(n^2)$$

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

$$n^2 \neq o(n^2) \text{ (just like } 2 \not\leq 2)$$

$$n^2 / 1000 \neq o(n^2)$$

ω - notation

$\omega(g(n)) = \{f(n) : \text{for all constants } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0\}.$

Another view, again, probably easier to use: $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty.$

$$n^{2.0001} = \omega(n^2)$$

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

$$n^2 \neq \omega(n^2)$$

Comparing functions

Relational properties:

Transitivity:

$f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$.
Same for O , Ω , o , and ω .

Reflexivity:

$f(n) = \Theta(f(n))$.
Same for O and Ω .

Symmetry:

$f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$.

Transpose symmetry:

$f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$.
 $f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$.

Divide-and-Conquer

Three methods for solving recurrences

- Substitution method
 - guess the bound and then use mathematical induction to prove our guess correct.
- **Recursion-tree method**
 - converts the recurrence into a tree whose nodes represent the costs incurred at various levels of the recursion.
- Master method
 - provides for recurrences of the form.

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

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 (...).
- The recursion-tree method promotes intuition, however.

Example of recursion tree

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

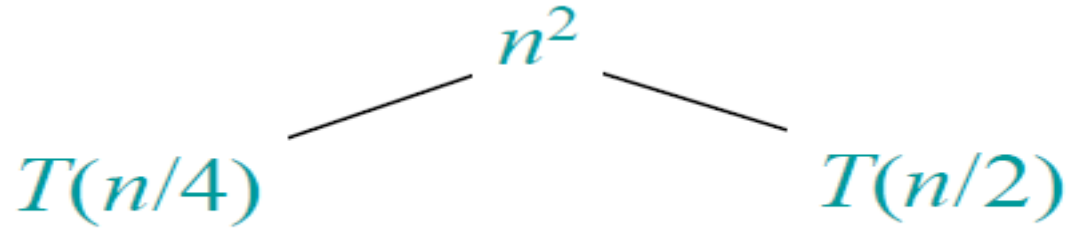
Example of recursion tree

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

$$T(n)$$

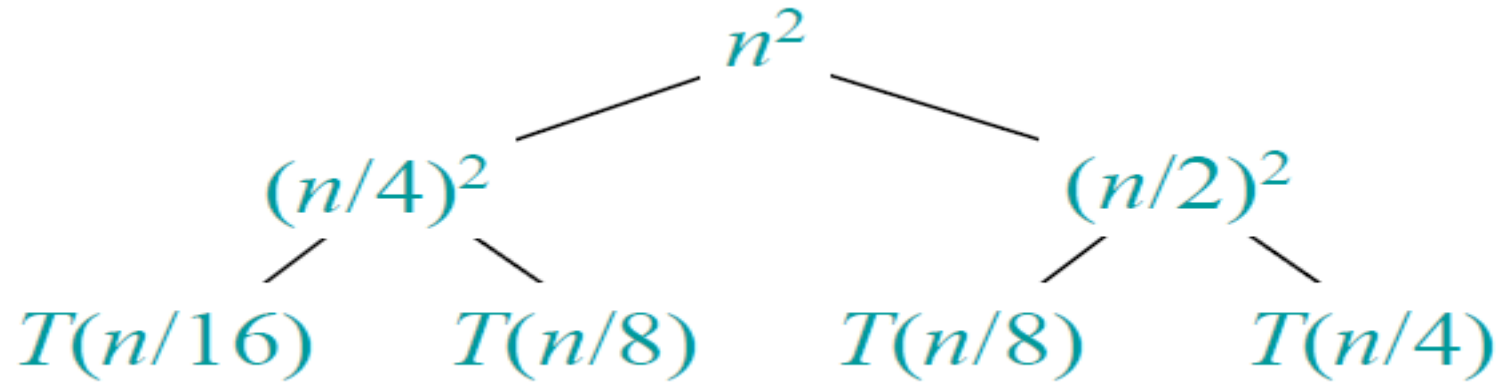
Example of recursion tree

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



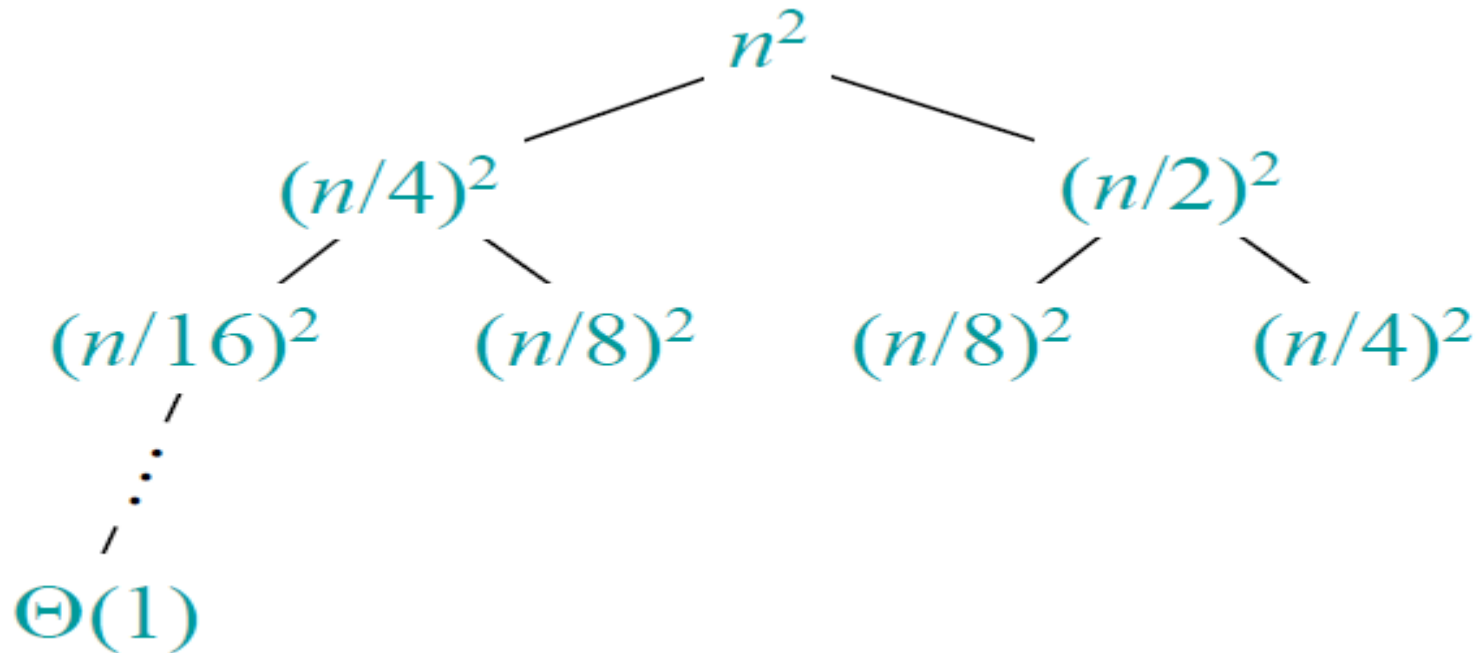
Example of recursion tree

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



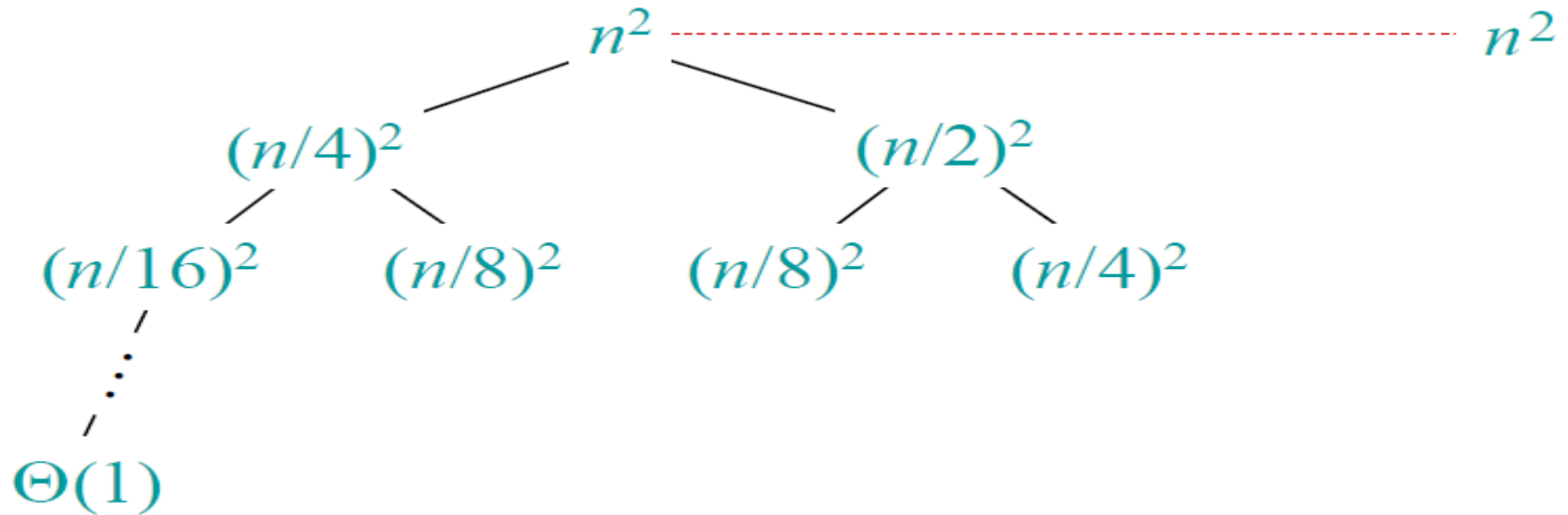
Example of recursion tree

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



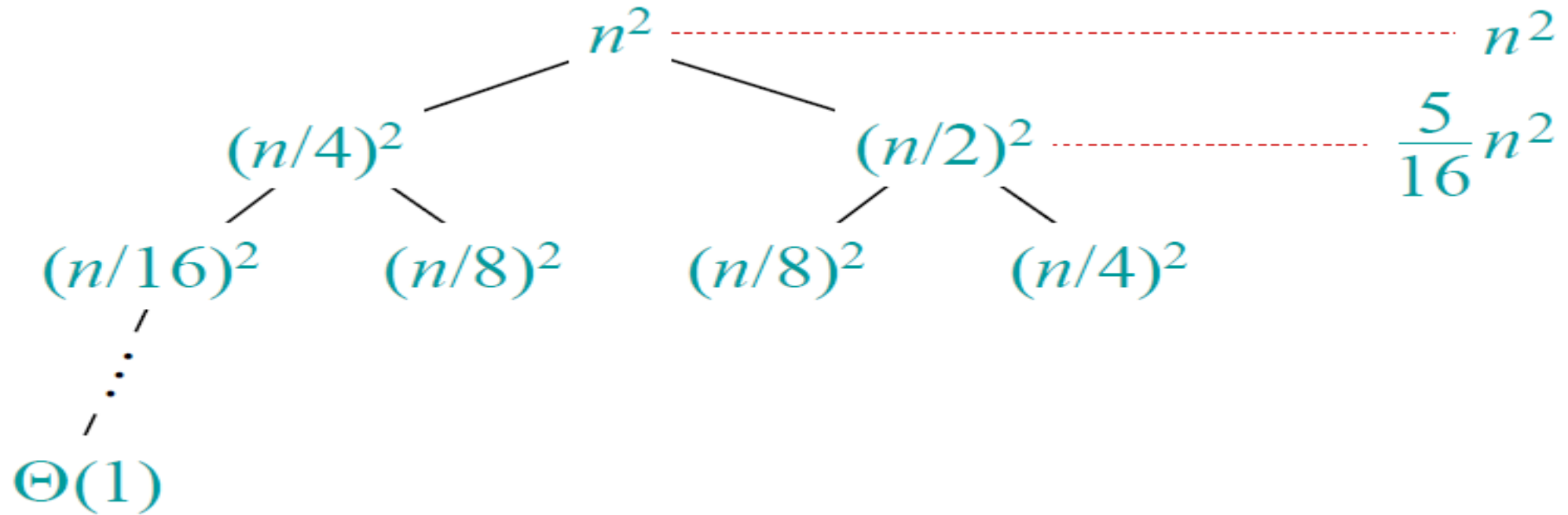
Example of recursion tree

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



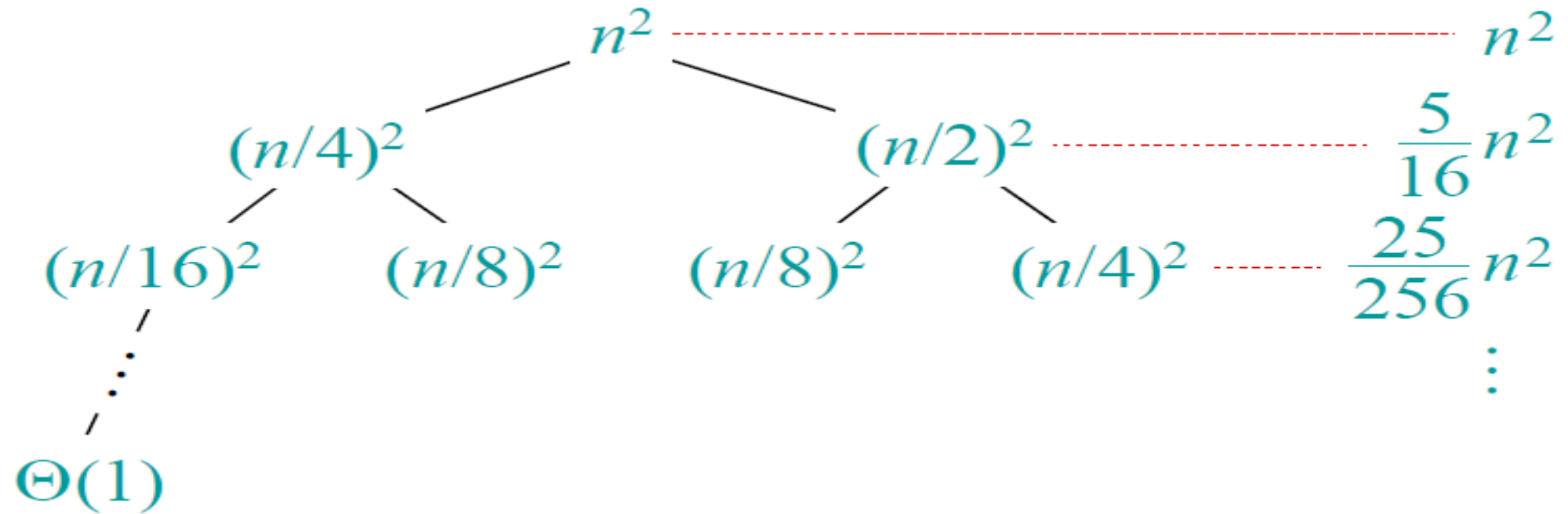
Example of recursion tree

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



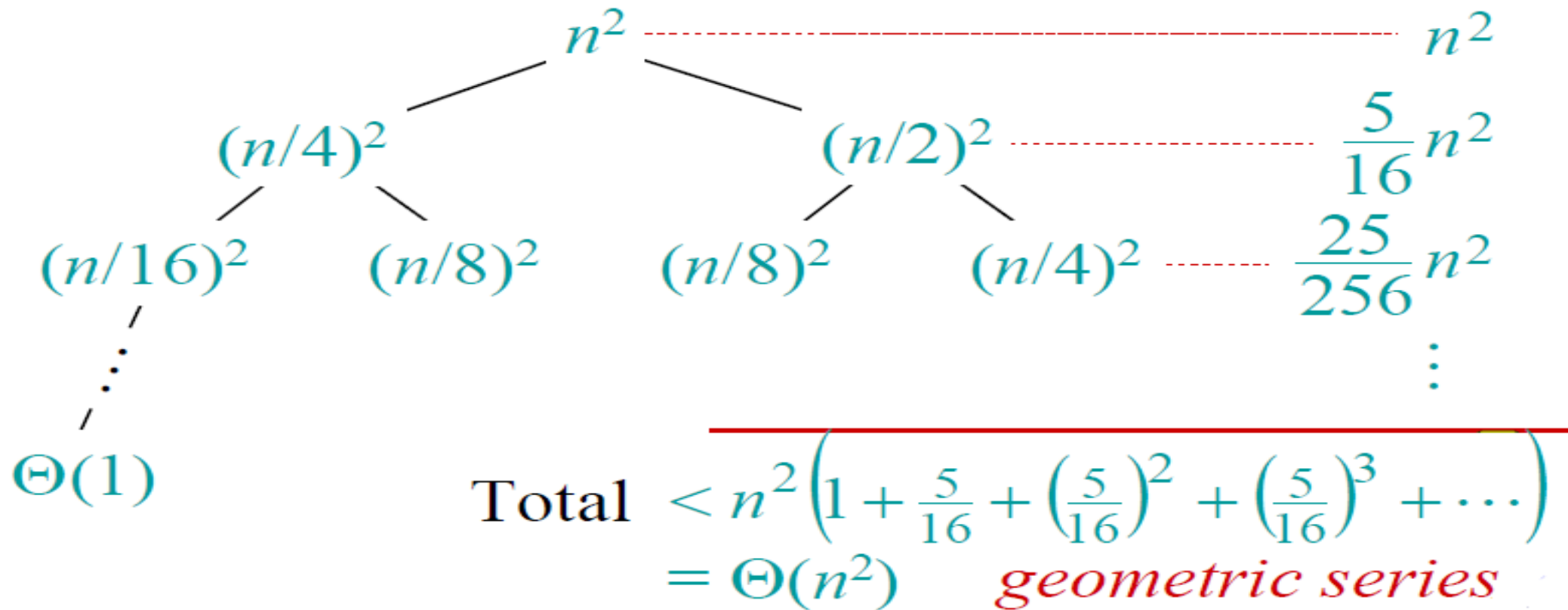
Example of recursion tree

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



Example of recursion tree

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



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 $n^{\log_b a}$: \rightarrow # of leaves

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

Three common cases (cont'd)

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

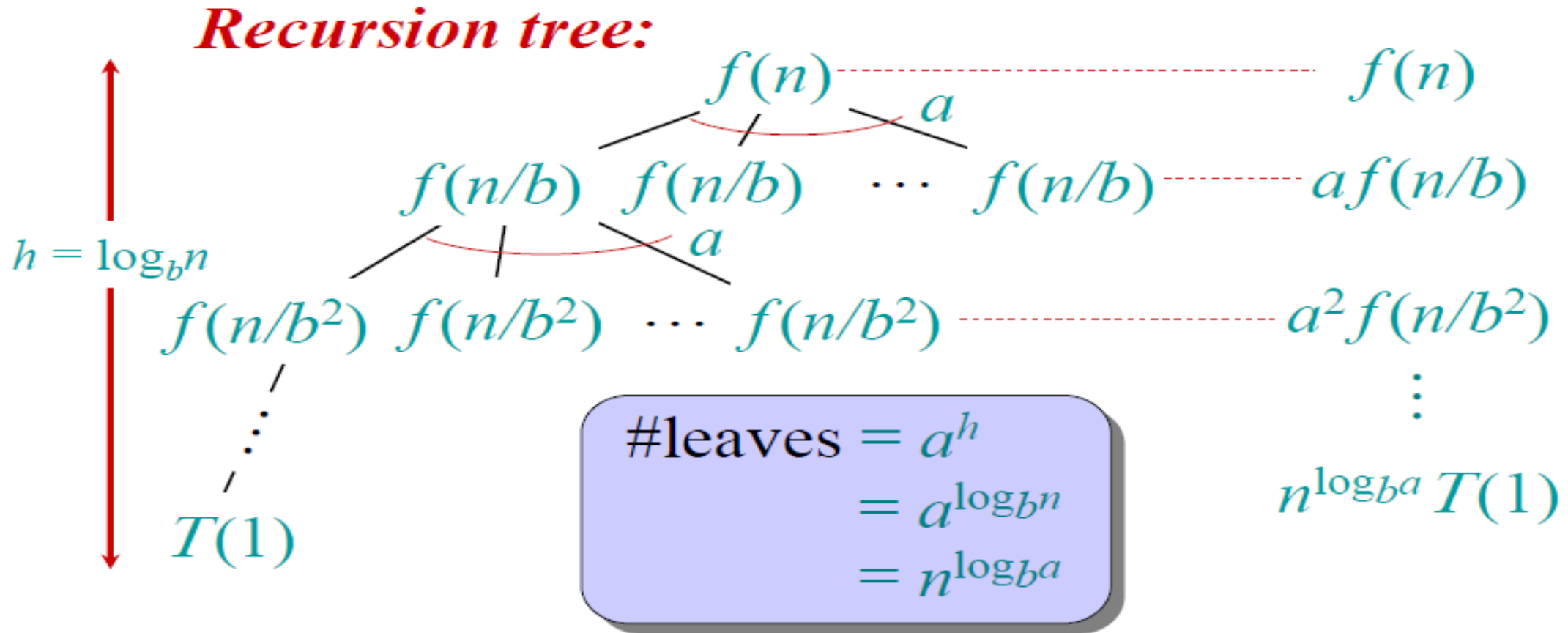
$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^3.$

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

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

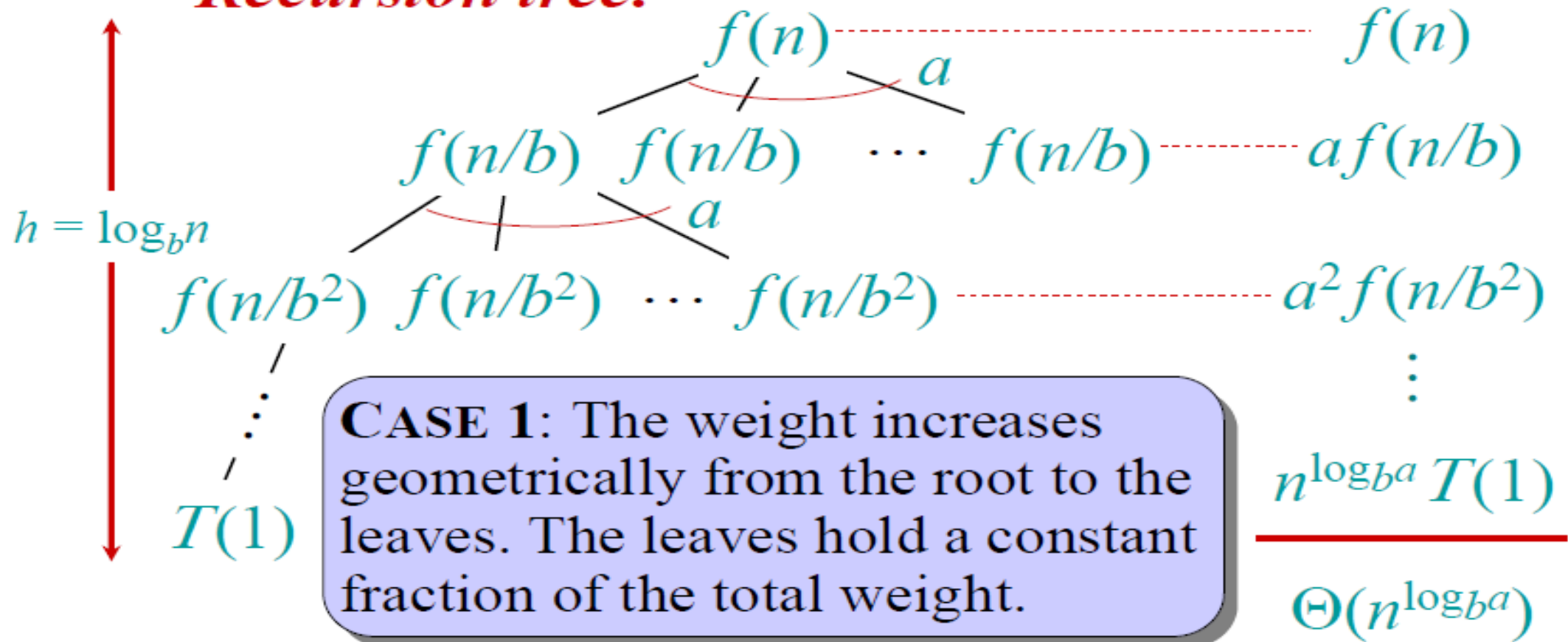
$\therefore T(n) = \Theta(n^3).$

Idea of master method

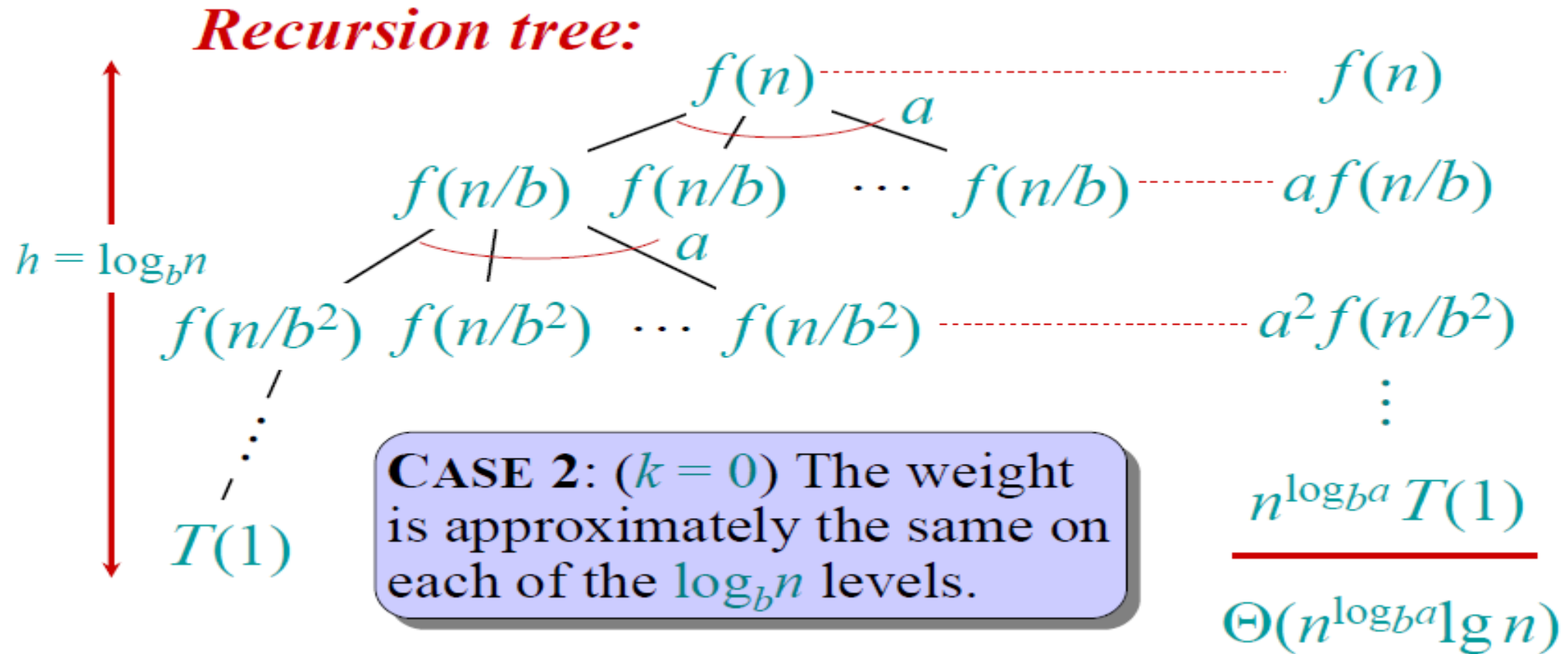


Idea of master method

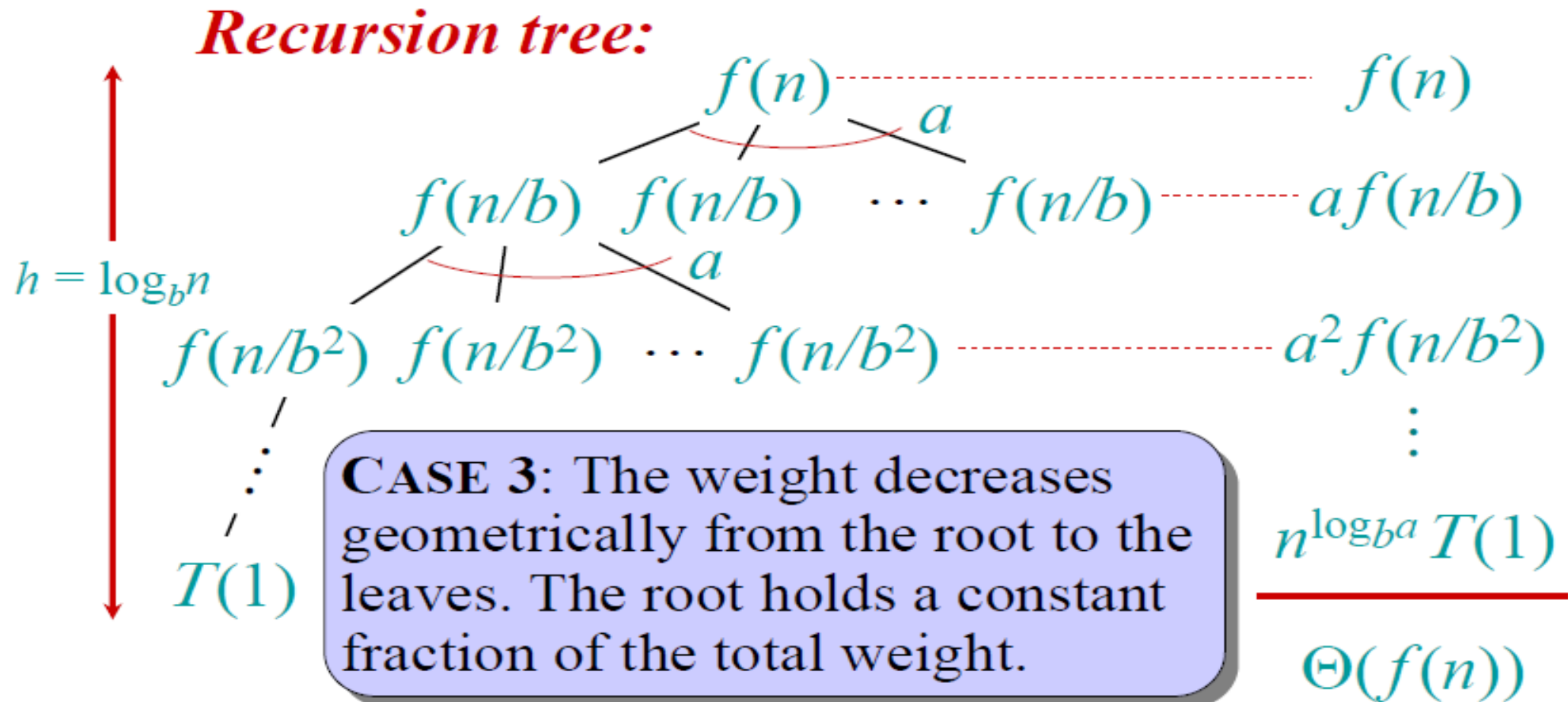
Recursion tree:



Idea of master method



Idea of master method



Appendix: geometric series

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

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

Implementation

Implementation of quick sort

QUICKSORT(A, p, r)

if $p < r$

$q = \text{PARTITION}(A, p, r)$

 QUICKSORT($A, p, q - 1$)

 QUICKSORT($A, q + 1, r$)

Initial call is QUICKSORT($A, 1, n$).

PARTITION(A, p, r)

$x = A[r]$

$i = p - 1$

for $j = p$ **to** $r - 1$

if $A[j] \leq x$

$i = i + 1$

 exchange $A[i]$ with $A[j]$

 exchange $A[i + 1]$ with $A[r]$

return $i + 1$



Definition of quick sort

```
def quickSort(A, p:int, r:int):
```

```
    if p < r:
```

```
        q = partition(A, p, r)
```

```
        quickSort(A, p, q-1)
```

```
        quickSort(A, q+1, r)
```

```
def partition(A, p:int, r:int):
```

```
    x = A[r]
```

```
    i = p-1
```

```
    for j in range(p, r):
```

```
        if A[j] < x:
```

```
            i += 1
```

```
            A[i], A[j] = A[j], A[i]
```

```
    A[i+1], A[r] = A[r], A[i+1]
```

```
    return i+1
```

Example code test

- Code test: <https://www.acmicpc.net/problem/11004>
- Solving the problem using quick sort
- Example result of submission

제출 번호	아이디	문제	결과	메모리	시간	언어	코드 길이	제출한 시간
48606321	aikiho	11004	맞았습니다!!	646904 KB	1936 ms	PyPy3 / 수정	1207 B	1분 전

THANK YOU

