Solving Recurrences

Divide-and-Conquer design paradigm

1. Divide : divide a problem into subproblems
2. Conquer : solve the subproblems recursively
3. Combine : combine the subproblem solutions appropriately

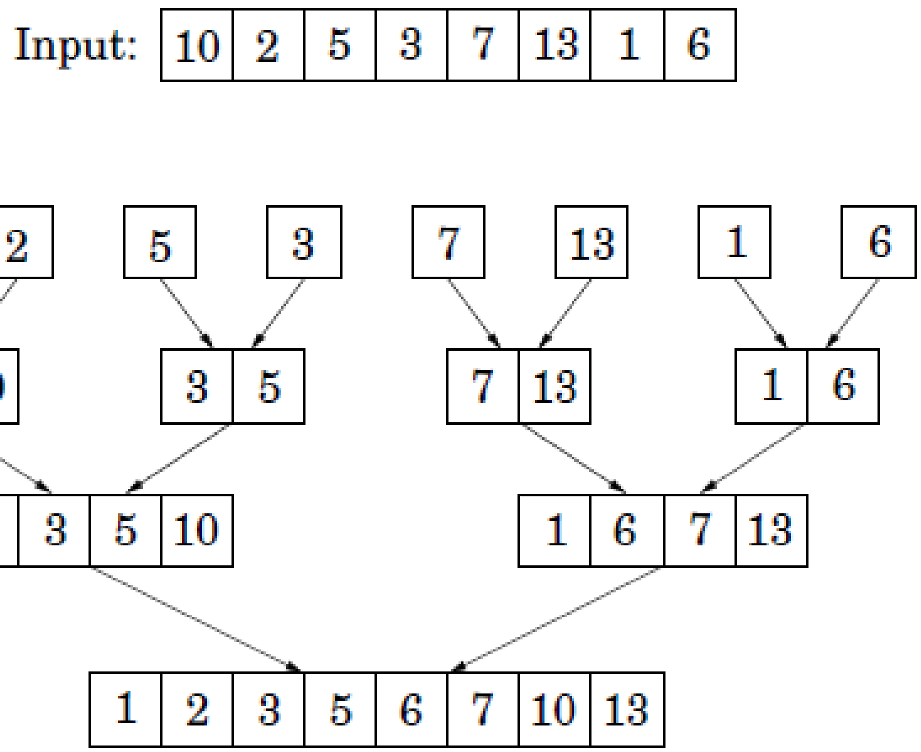
Merge Sort

1. Divide: Divide the input array in 2 subarrays.
2. Conquer: Recursively sort 2 subarrays.
3. Combine: Merge 2 sorted subarrays.

```
function mergesort(a[1...n])
Input:  An array of numbers a[1...n]
Output:  A sorted version of this array

if n > 1:
  return merge(mergesort(a[1...⌊n/2⌋]), mergesort(a[⌊n/2⌋ + 1...n]))
else:
  return a
```

Merge Sort


Input:
10 2 5 3 7 13 1 6
10
2
5
3
7
13
1
6
2 10
3 5
7 13
1 6
2 3 5 10
1 6 7 13
1 2 3 5 6 7 10 13

Merge

```
function merge(x[1...k], y[1...l])
if k = 0:  return y[1...l]
if l = 0:  return x[1...k]
if x[1] ≤ y[1]:
    return x[1] ∘ merge(x[2...k], y[1...l])
else:
    return y[1] ∘ merge(x[1...k], y[2...l])
```

Analysis of Merge Sort

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

 # subproblems

subproblem size

work dividing and combining

Recurrences

- When we analyze algorithms expressed in a recursive way, we get a recurrence.
  - Merge sort
  - Selection sort : find the smallest element and put it in the leftmost position. Then, recursively sort the remainder of the array.

- Base cases : when the problem size gets down to a small constant, just use a brute force approach that takes constant time.

  $T(n) \leq c$ for all $n \leq n_0$

Solving recurrences

- Solve by unrolling
- Substitution method
- Recursion tree
- Master method

Solve by unrolling

- Selection sort : find the smallest element and put it in the leftmost position. Then, recursively sort the remainder of the array.

$T(n) = cn + T(n\text{-}1) = cn + c(n\text{-}1) + c(n\text{-}2) + \ldots + c$

top n/2 terms are each at least $cn/2$

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

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

Substitution method

- Guess the form of the solution
- Use mathematical induction to find the constants and show that the solution works.

Substitution method

- Guess the form of the solution
- Use mathematical induction to find the constants and show that the solution works.

Ex) $T(n) = 4\,T(n/2) + n$

- [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.

$$
\begin{aligned}
T(n) &= 4\,T(n/2) + n \\
&\le 4\,c\,(n/2)^3 + n \\
&= (c/2)\,n^3 + n \\
&= cn^3 - ((c/2)\,n^3 - n) \\
&\le cn^3 \quad \text{if } (c/2)\,n^3 - n \ge 0\text{, for example, if } c \ge 2 \text{ and } n \ge 1.
\end{aligned}
$$

Substitution method

Ex) $T(n) = 4\,T(n/2) + n$

- We must handle the initial conditions, i.e., ground the induction with base cases.
- Base : $T(n) = \Theta(1)$ for all $n < n_0$, for a suitable constant n_0 .
- For $1 \le n < n_0$, we have "$\Theta(1)$" $\le cn^3$, if we pick c big enough.
- This bound is not tight.

Tighter bound?

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

- We shall prove that $T(n) = O(n^2)$.
- Assume that $T(k) \leq ck^2$ for $k < n$.

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

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

$= c\ n^2 + n$

$= O(n^2)$

Tighter bound?

Ex) $T(n) = 4\,T(n/2) + n$

- We shall prove that $T(n) = O(n^2)$.
- Assume that $T(k) \le ck^2$ for $k < n$.

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

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

$= c\,n^2 + n$

$\cancel{= O(n^2)}$

Wrong! We must prove the exact form of the I.H.

$\le c\,n^2$ for no choice of $c > 0$!

IDEA : Strengthen the inductive hypothesis by subtracting a low-order term.

I.H. : $T(k) \le c_1 k^2 - c_2 k$ for $k < n$.

Tighter bound?

Ex) $T(n) = 4\,T(n/2) + n$

IDEA : Strengthen the inductive hypothesis by subtracting a low-order term.

I.H. : $T(k) \le c_1 k^2 - c_2 k$ for $k < n$.

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

Pick c_1 big enough to handle the initial conditions.

Substitution method

- Show the upper and lower bounds separately. (Might need to use different constants for each.)
- Make sure you show the same *exact* form of the inductive hypothesis.
  - Subtract a lower-order term if necessary

Recursion-tree method

- Can be used to provide a good guess for the substitution method.
- Each node represents the cost of a single subproblem somewhere in the set of recursive function invocations.
- Sum the costs within each level of the tree to obtain a set of per-level costs.
- Sum all the per-level costs to determine the total cost of all levels of the recursion.

Example of recursion-tree

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

Example of recursion-tree

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

$$T(n)$$

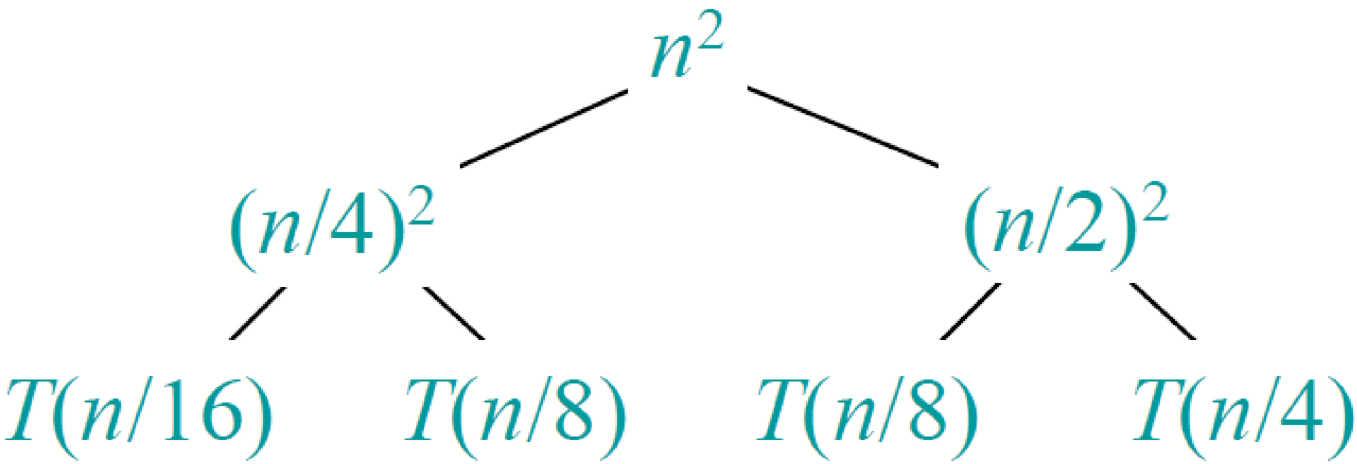
Example of recursion-tree

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

$$\begin{array}{ccc} & n^2 & \\ \swarrow & & \searrow \\ T(n/4) & & T(n/2) \end{array}$$

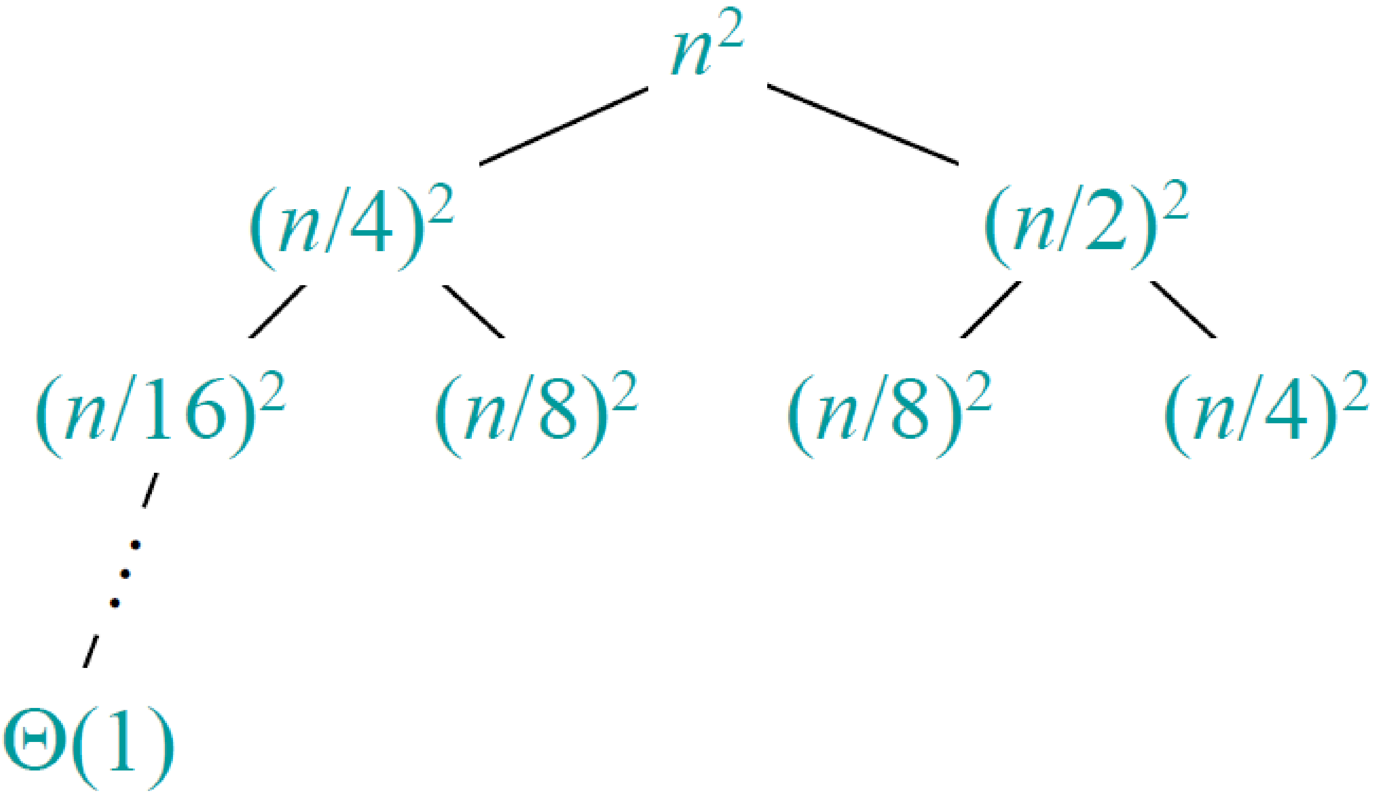
Example of recursion-tree

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



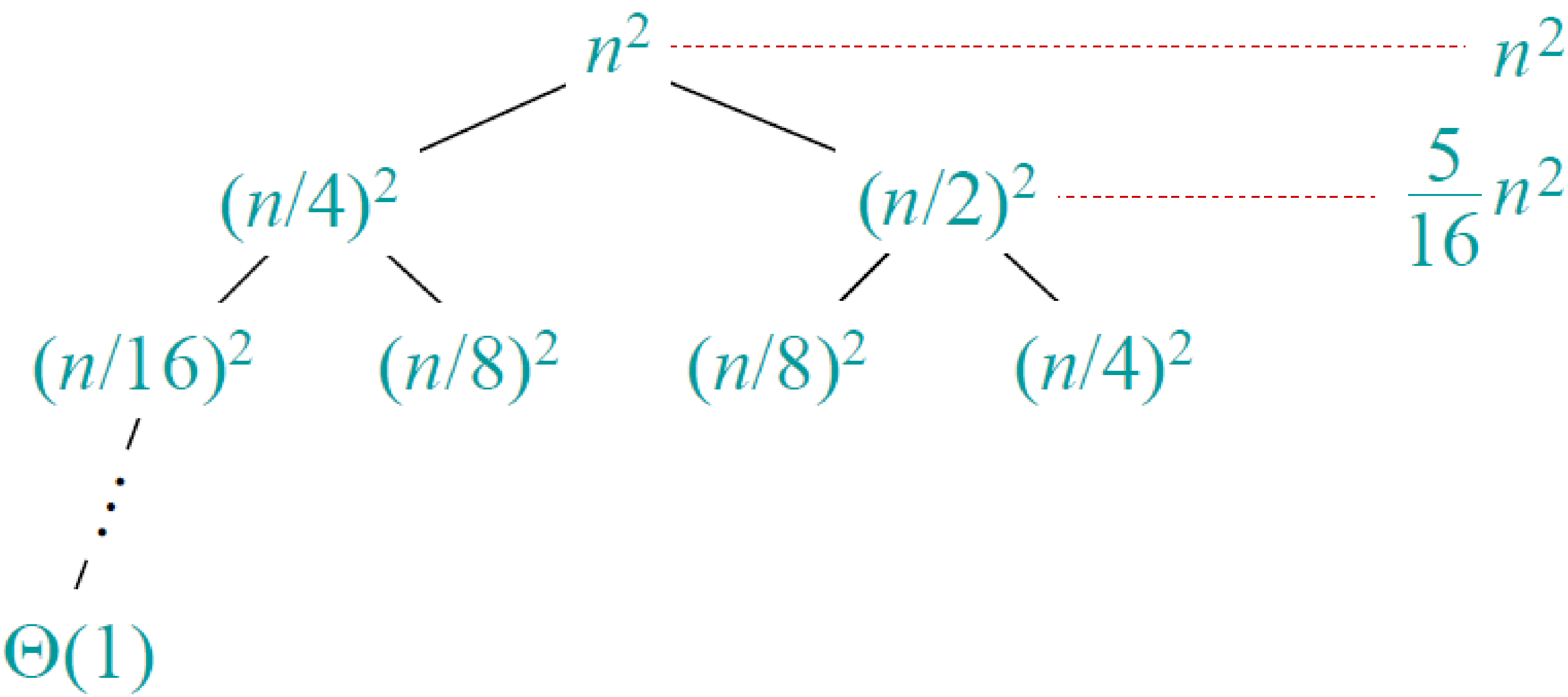
Example of recursion-tree

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



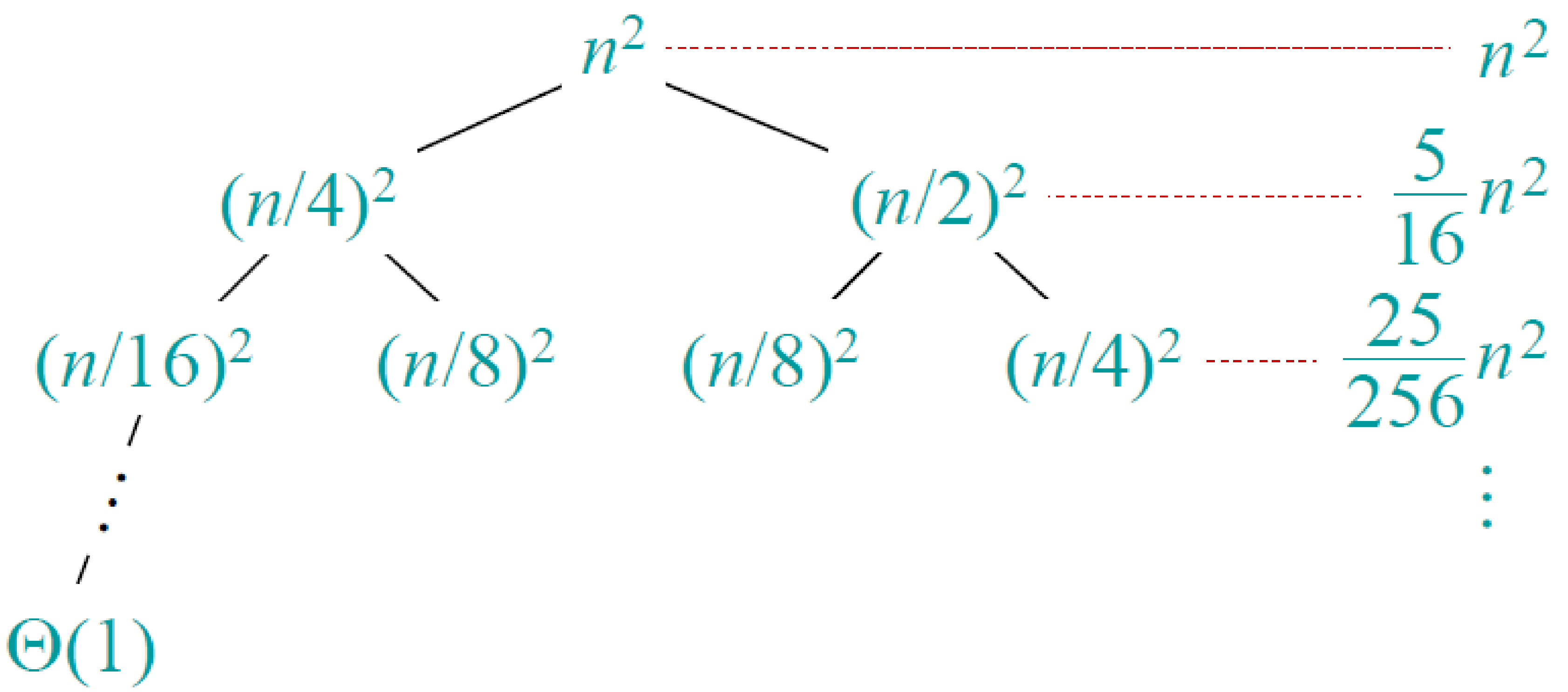
Example of recursion-tree

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



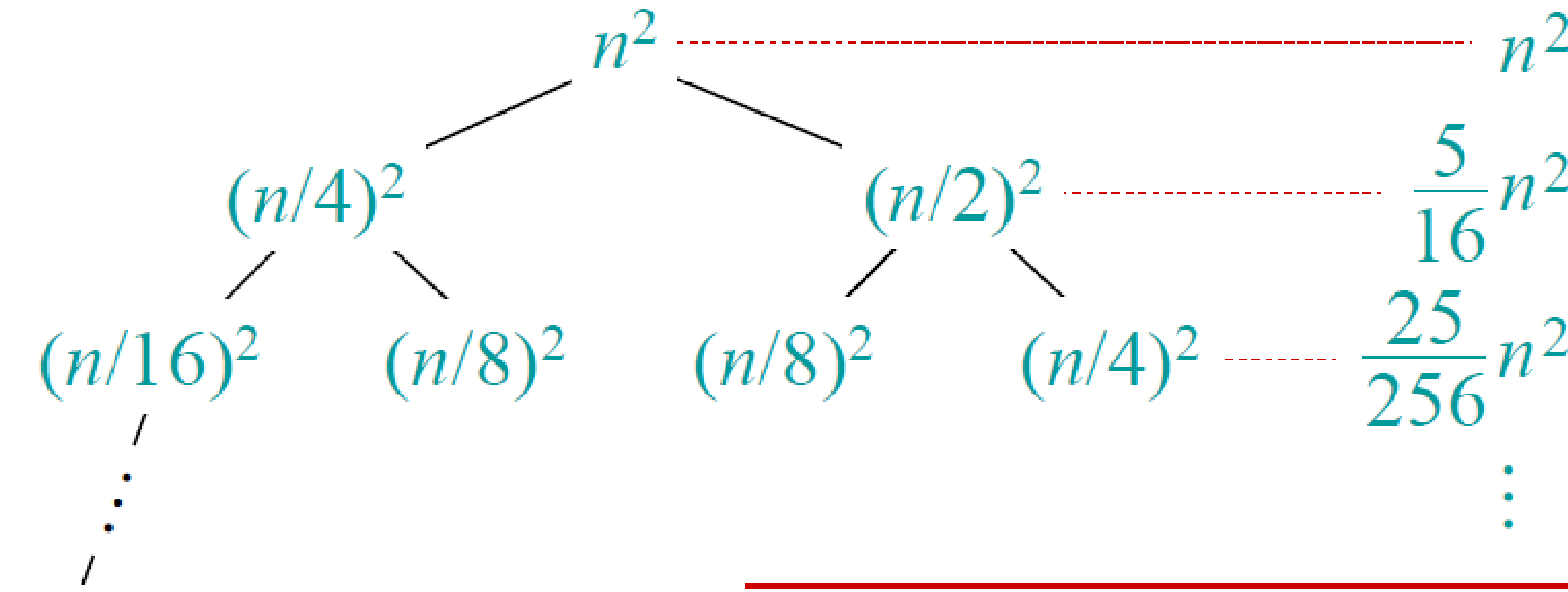
Example of recursion-tree

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



Example of recursion-tree

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



$\Theta(1)$

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

geometric series

Geometric series

$$\sum_{i=0}^{k} \alpha^{i} = 1 + \alpha + \alpha^{2} + \ldots + \alpha^{k}$$

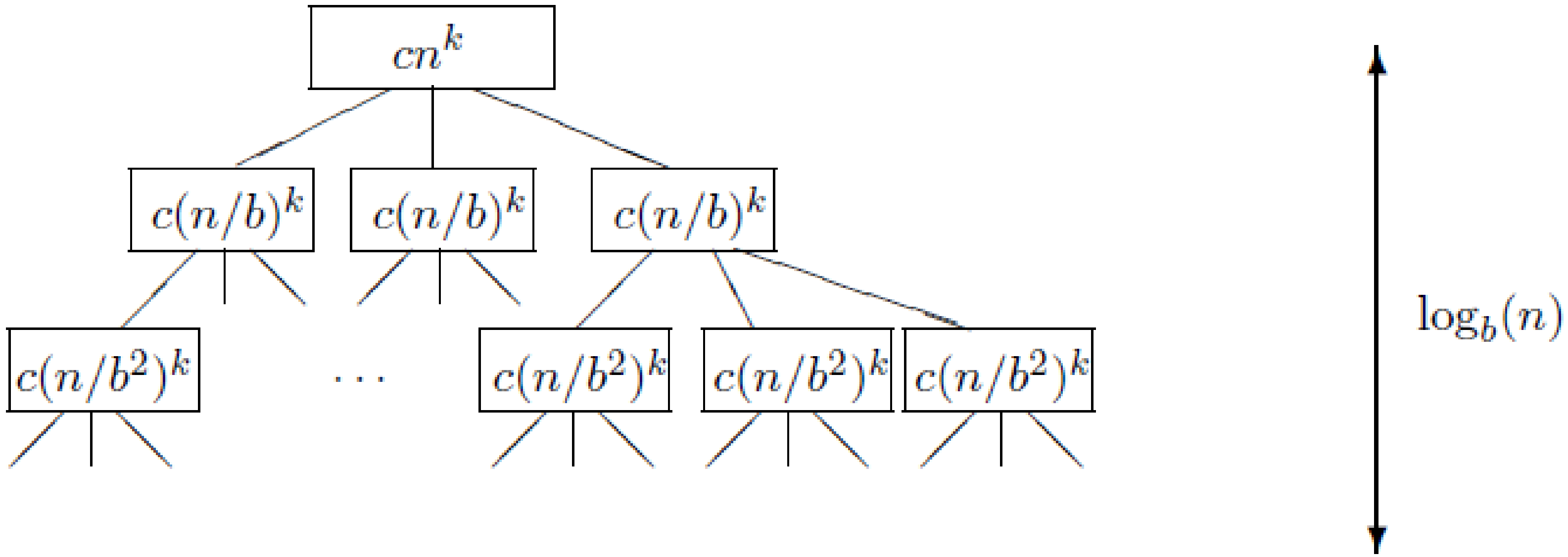
- If $\alpha > 1$, the last term dominates.
- If $\alpha < 1$, the first term dominates.

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

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

Divide-and-conquer style recurrence

$T(n) = a\, T(n / b) + c\, n^k$



$$cn^k \left[1 + a/b^k + (a/b^k)^2 + (a/b^k)^3 + \ldots + (a/b^k)^{\log_b n}\right]$$

Define $r = a / b^k$

$$cn^k \left[1 + r + r^2 + r^3 + \ldots + r^{\log_b n}\right]$$

$$cn^k \left[1 + r + r^2 + r^3 + \ldots + r^{\log_b n}\right]$$

- Case 1 : $r < 1$

  Upper bound : $cn^{k}/(1-r)$

  Lower bound : cn^{k}

  $\Theta(n^{k})$

- Case 2 : $r = 1$

  $\Theta(n^{k} \log n)$

- Case 3 : $r > 1$

  The last term dominates.

  $$cn^k r^{\log_b n} \left[(1/r)^{\log_b n} + \ldots + 1/r + 1\right]$$

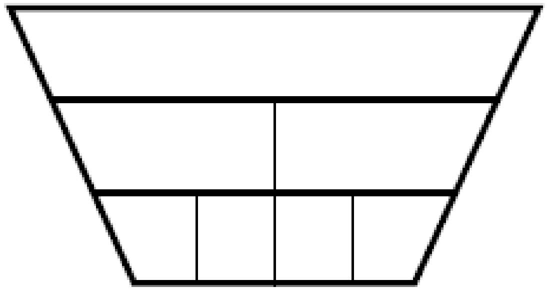
$$T(n) \in \Theta\left(n^k (a/b^k)^{\log_b n}\right)$$

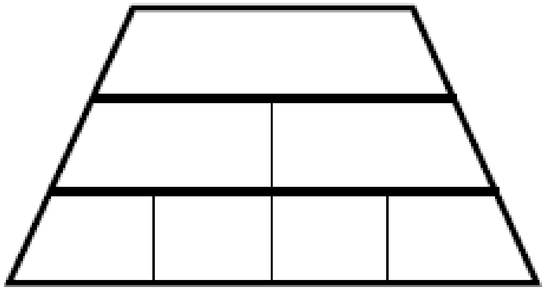
$$b^{k \log_b n} = n^k \qquad T(n) \in \Theta\left(a^{\log_b n}\right).$$

$$T(n) \in \Theta\left(n^{\log_b a}\right)$$

$$T(n) = a\,T(n / b) + c\,n^k$$

$$\begin{aligned} T(n) &\in \Theta(n^k) \ \textit{if}\ a < b^k \\ T(n) &\in \Theta(n^k \log n) \ \textit{if}\ a = b^k \\ T(n) &\in \Theta(n^{\log_b a}) \ \textit{if}\ a > b^k \end{aligned}$$





Master method

- Master theorem

Let $a \geq 1$, $b > 1$ be constants, function $f(n) > 0$ and let $T(n)$ be defined on the nonnegative integers by the recurrence $T(\mathrm{n}) = a\,T(n/b) + f(n)$.

Then $T(n)$ can be bounded asymptotically as follows.

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

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

2. If $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $\mathrm{k} \geq 0$, then

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

3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and

   if $f(n)$ satisfies the regularity condition $a\,f(n/b) \leq c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then

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

Case 1

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

$T(\mathrm{n}) = a\, T(n/b) + f(n)$.

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

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

Case 1 : $f(n)$ is polynomially smaller. ($f(n)$ is asymptotically smaller than $n^{\log_b a}$ by a factor of n^{ε} for some constant $\varepsilon > 0$.)

– Cost is dominated by leaves.

Case 1

$T(\mathrm{n}) = a\,T(n/b) + f(n)$.

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

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

Case 1 : $f(n)$ is polynomially smaller. ($f(n)$ is asymptotically smaller than $n^{\log_b a}$ by a factor of n^{ε} for some constant $\varepsilon > 0$.)

– Cost is dominated by leaves.

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 $\varepsilon = 1$.

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

Case 2

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

If $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$,
then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

Case 2 : $f(n)$ and $n^{\log_b a}$ grow at similar rates - multiply by a logarithmic factor.

Case 2

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

If $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$,
then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

Case 2 : $f(n)$ and $n^{\log_b a}$ grow at similar rates - multiply by a logarithmic factor.

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, $k = 0$.

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

Case 3

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

If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $f(n)$ satisfies the regularity condition $a\,f(n/b) \le c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then

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

Case 3 : $f(n)$ is polynomially larger. Satisfy “regularity” condition

$$a\,f(n/b) \le c\,f(n)$$

– Cost is dominated by root.

Case 3

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

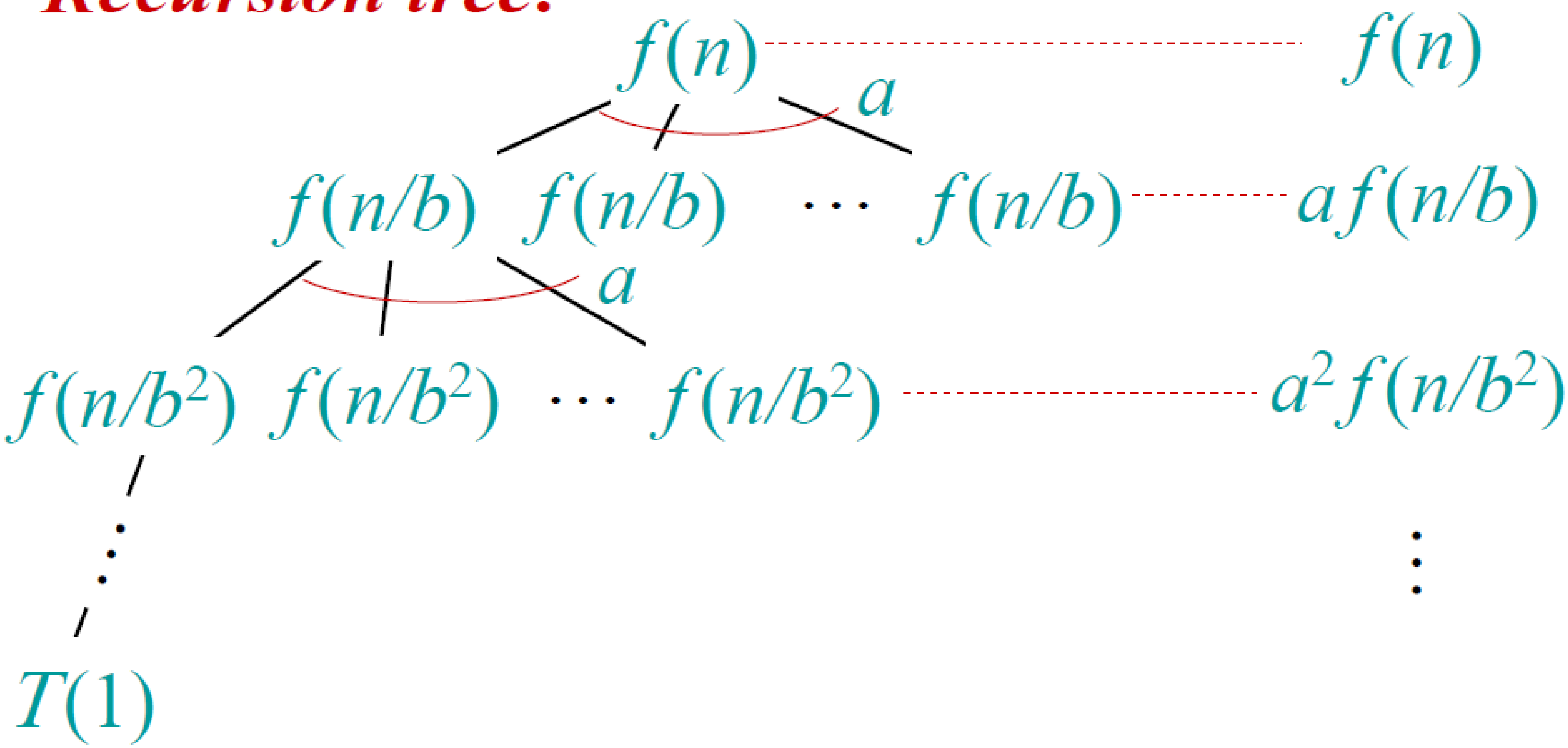
If $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some constant $\varepsilon > 0$, and
if $f(n)$ satisfies the regularity condition $a\, f(n/b) \leq c\, f(n)$ for some constant $c < 1$ and all sufficiently large n, then
$T(n) = \Theta(f(n))$.

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

Master theorem

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

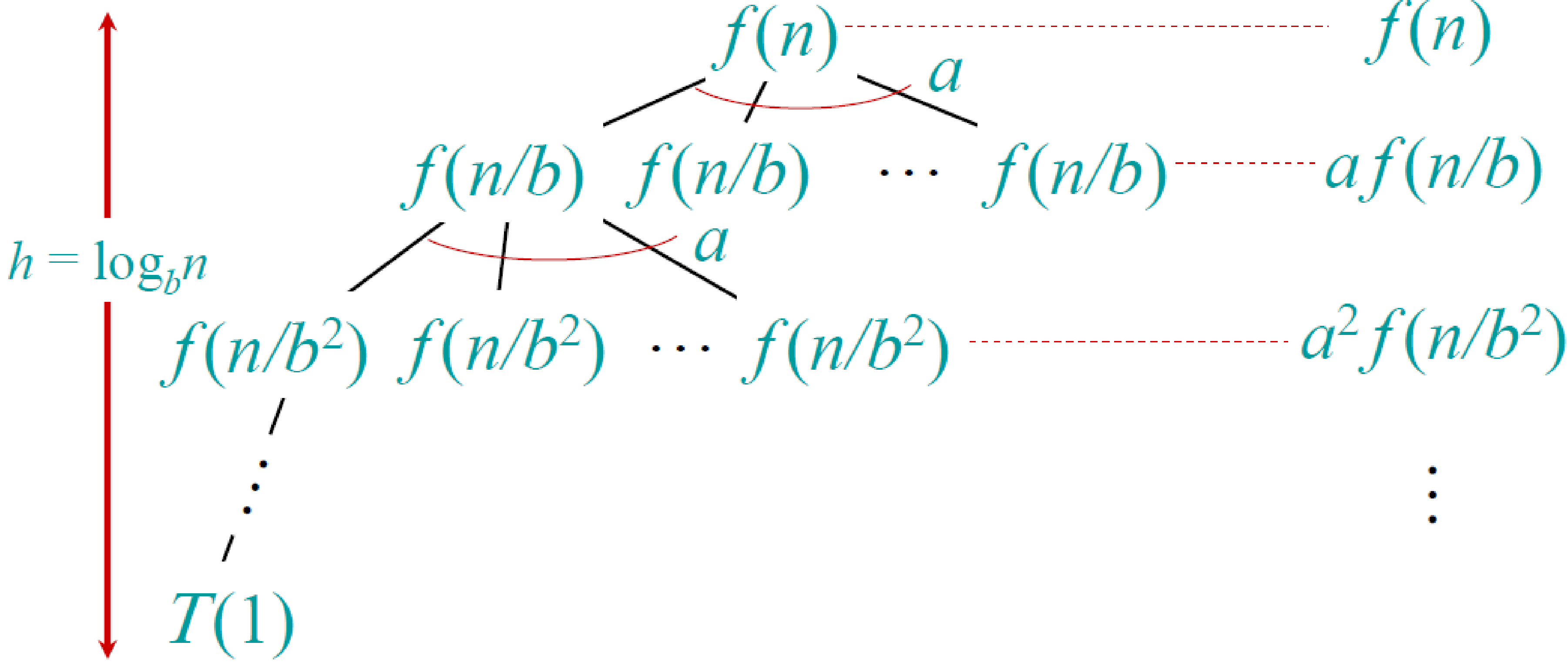
Recursion tree:



Master theorem

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

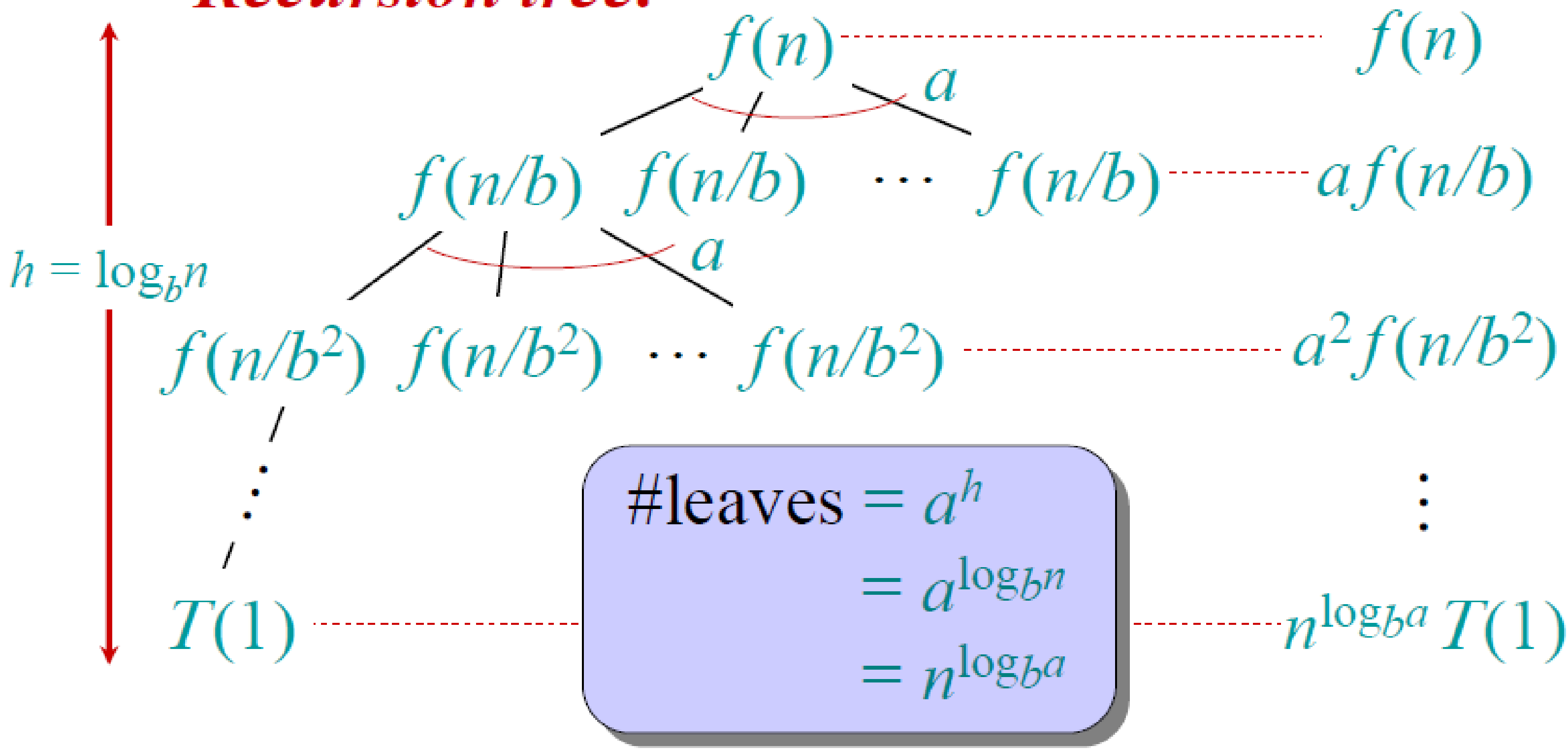
Recursion tree:



Master theorem

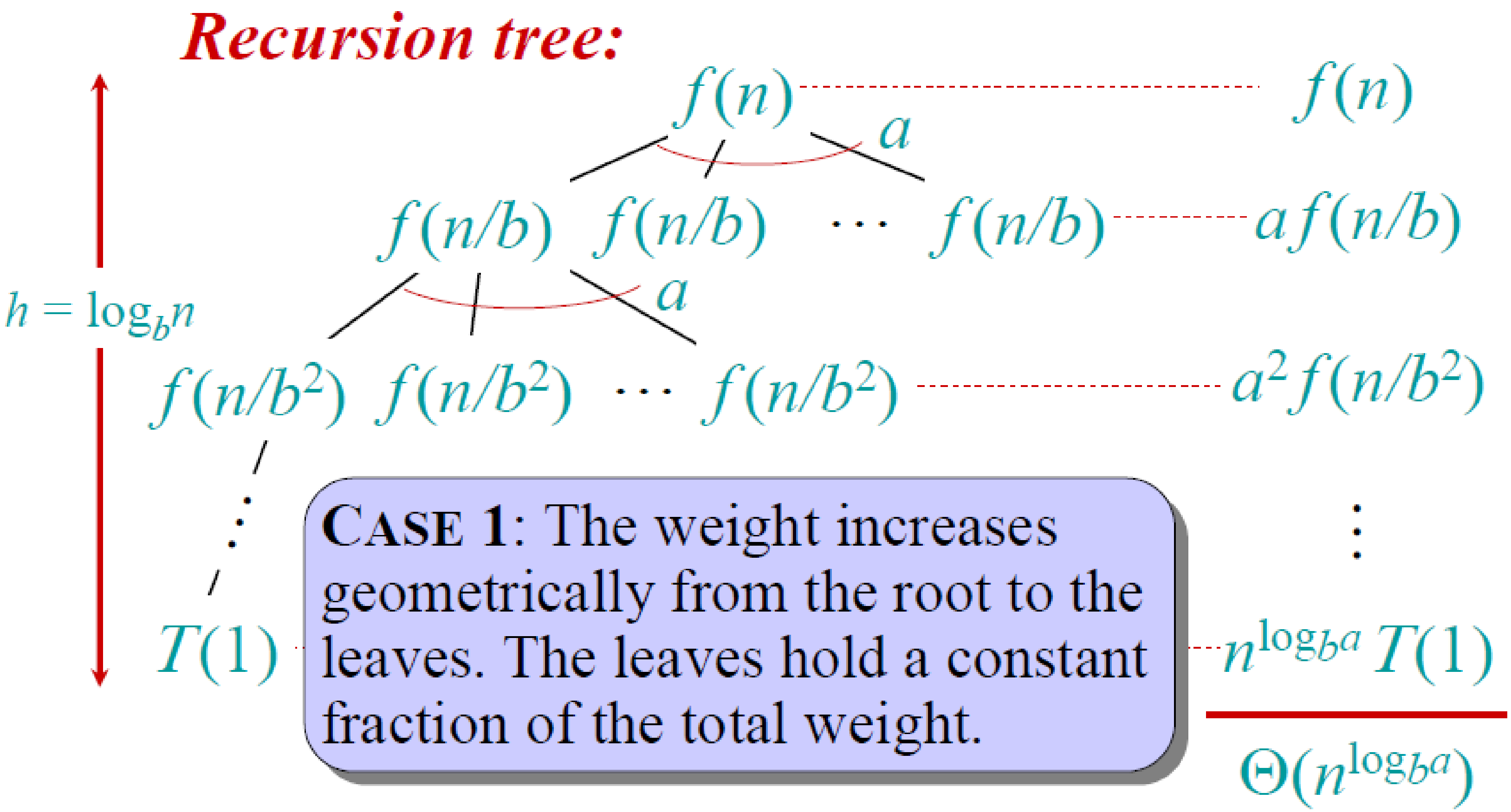
$$T(n) = a\,T(n/b) + f(n)$$

Recursion tree:



Master theorem

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

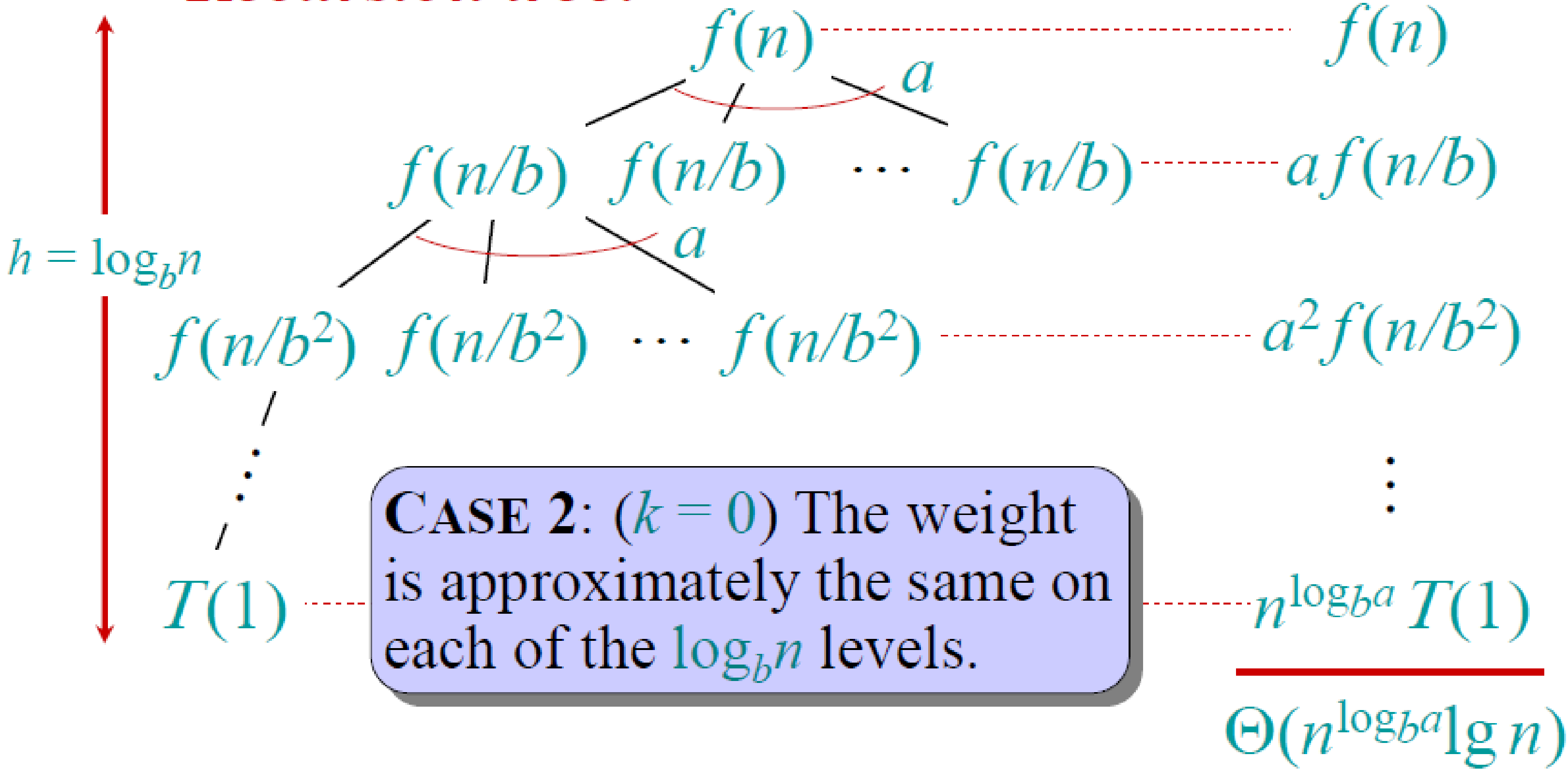


CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

Master theorem

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

Recursion tree:



Master theorem

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

Recursion tree: