JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Chapter 4
Divide-and-Conquer

Outline

- Solving Recurrences
- Methods for Solving Recurrences
- Divide-and-Conquer
- Substitution method
- Recursion tree
- Master Theorem

Solving Recurrences

- The analysis of merge sort from **Chapter 2** required a recursive solution.
- Recurrences are like solving integrals, differential equations, etc.
- A few tricks must be learned to gain an intuitive understanding of recurrences.
- **Chapter 4** covers the applications of recurrences as it relates to divide-and-conquer algorithms.

Methods for solving recurrences

- Obtaining asymptotic O or Θ bounds
  - In the **substitution method** the bound is guessed and then mathematical induction is used to determine if the guess is correct.
  - The **recursion-tree method** converts the recurrence into a tree whose nodes represent the cost incurred at various levels of the recursion.
  - The **master method** provides bounds for recurrences of the form $\mathrm{T}(n) = a\mathrm{T}(n/b) + f(n)$

The Divide-and-Conquer Design Paradigm

1. ***Divide*** the problem (instance) into subproblems.

2. ***Conquer*** the subproblems by solving them recursively.

3. ***Combine*** subproblem solutions.

Analyzing Divide-and- Conquer Algorithms

- Use a recurrence to characterize the running time of a divide-and-conquer algorithm. Solving the recurrence give an asymptotic running time.

- A recurrence is a function that is defined in terns of the following:
  - One or more base cases
  - Itself, with smaller arguments

Divide-and-Conquer Examples

- $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 1 & \text{if } n > 1 \end{cases}$

  Solution: $T(n) = n$

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

  Solution: $T(n) = n\lg n + n$

- $T(n) = \begin{cases} 0 & \text{if } n = 2 \\ T(\sqrt{n}) + 1 & \text{if } n > 2 \end{cases}$

  Solution: $T(n) = \lg\lg n$

- $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/3) + T(2n/3) + n & \text{if } n > 1 \end{cases}$

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

Maximum-Subarray Problems

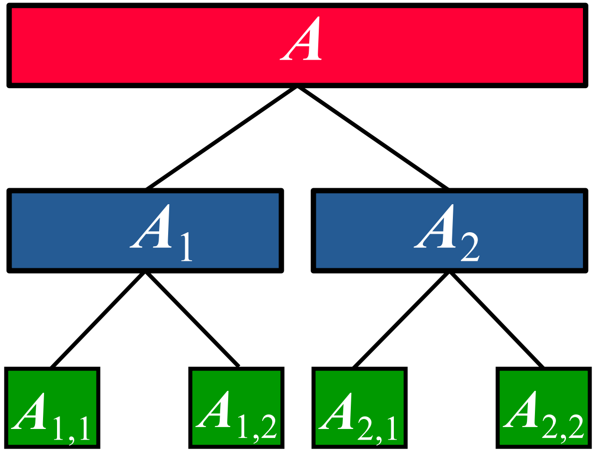
- Input: An array $A[1 \ldots n]$. of numbers. [Assume that some of the numbers are negative, because this problem is trivial when all numbers are nonnegative.]

- Output: Indices i and j such that $A[i \ldots \mathrm{j}]$. has the greatest sum of any nonempty, contiguous subarray of A, along with the sum of the values in $A[i \ldots \mathrm{j}]$.

Solving Maximum-Subarray Problems by Divide-and-Conquer

Use divide-and-conquer to solve in $O(n\lg n)$

- Subproblem: Find a maximum subarray of A[low .. high]
  - Divide the subarray A in two subarrays A_1, A_2 of equal size as possible. Find the midpoint mid of the subarrays, and consider the subarrays A[low .. mid] and A[mid+1 .. high]
  - Conquer by finding a maximum subarrays of A[low .. mid] and A[mid+1 .. high]
  - Combine by finding a maximum subarray that crosses the midpoint, and using the best solution out of the three (the subarray crossing the midpoint and the two solutions found in the conquer step).
- This strategy works because any subarray must either lie entirely on one side of the midpoint or cross the midpoint.



Divide and Conquer Analysis

- Simplified assumption: Original problem size is a power of 2, so that all subproblem sizes are integer.
- Let $T(n)$ denote the running time of FIND-MAXIMUM-SUBARRAY on a subarray of n elements.
- Base case: Occurs when high equals low, so that $n/2$. The procedure just returns)$\Longrightarrow T(n) = \Theta(1)$.

Divide and Conquer Analysis cont.

- Recursive case: Occurs when $n > 1$.

- Dividing takes $\Theta(1)$ time.
  - Conquering solves two subproblems, each on a subarray of $n = 2$ elements. Takes $T(n/2)$ time for each subproblem $\Longrightarrow 2T(n/2)$ time for conquering.
  - Combining consists of calling FIND-MAX-CROSSING-SUBARRAY, which takes, $\Theta(n)$ time, and a constant number of constant-time tests $\Longrightarrow \Theta(n) + \Theta(1)$ time for combining.

Divide and Conquer Analysis cont.

- Recurrence for recursive case becomes

$$\begin{aligned} T(n) &= \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1) \\ &= 2T(n/2) + \Theta(n) \end{aligned}$$

  The $\Theta(1)$ terms are absorbed into the $\Theta(n)$

- The recurrence for all cases:

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

- Same recurrence as for merge sort. Can use the master method to show that it has solution $T(n) = \Theta(n \lg n)$.
- Thus, with divide-and-conquer, we have developed a $\Theta(n \lg n)$-time solution. Better than the , $\Theta(n^2)$-time brute-force solution.

Substitution Method

The most general method:

1. ***Guess*** the form of the solution.
2. ***Verify*** by induction.

   Use induction to find the constants and show that the solution works.
3. ***Solve*** for constants.

Substitution Method Example 1

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

Guess: $T(n) = n\lg n + n$. [Here, we have a recurrence with an exact function, rather than asymptotic notation, and the solution is also exact rather than asymptotic. We'll have to check boundary conditions and the base case.]

Substitution Method Example 1 cont.

Induction:

Basis: $n = 1 \Longrightarrow n\lg n + n = 1 = T(n)$

Inductive step: Inductive hypothesis is that $T(k) = k\lg k + k$ for all $k < n$. We'll use this inductive hypothesis for $T(n/2)$.

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2\big((n/2)\lg(n/2) + n/2\big) + n \\ &= n\lg(n/2) + n + n \\ &= n(\lg n - \lg 2) + n + n \\ &= n\lg n - n + n + n \\ &= n\lg n + n. \qquad \blacksquare \end{aligned}$$

Asymptotic Notation

Generally, asymptotic notation is used as:

- Write $T(n) = 2T(n/2) + \Theta(n)$
- Assume $T(n) = O(1)$ for sufficiently small n
- Express the solution by asymptotic notation $T(n) = \Theta(n\lg n)$
- Boundary cases are not considered, nor are base cases in the substitution proof
  - $T(n)$ is always constant for any constant n
  - Since an asymptotic solutions is sought for a recurrence, it will always be possible to choose a base case that works
  - When an asymptotic solution to a recurrence is sought, the base case is not considered in the proofs
  - When an exact solutions is sought, the base case must be considered

Substitution Method Example 2

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

- Assume that $T(1) = \Theta(1)$.
- Guess $O(n^3)$. (Prove O and Ω separately.)
- Assume that $T(k) \leq ck^3$ for $k < n$.
- Prove $T(n) \leq cn^3$ by induction.

Substitution Method Example 2 cont.

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

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

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

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

$\leq cn^3$ ← *desired*

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

$c \geq 2$ and $n \geq 1$.

residual

Substitution Method Example 2 cont.

- 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.

Substitution Method Example 2 cont.

A Tighter Upper Bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \leq ck^2$ for $k < n$:

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

Substitution Method Example 2 cont.

A Tighter Upper Bound cont.

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

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

Wrong! We must prove the I.H.

Substitution Method Example 2 cont.

A Tighter Upper Bound cont.

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

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

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

$= cn^2 + n$ ***Wrong!*** We must prove the I.H.

$= cn^2 - (-n)$ [*desired – residual*]

$= O(n^2)$

$\le cn^2$ for *no* choice of $c > 0$.

Substitution Method Example 2 cont.

A Tighter Upper Bound cont.

Idea: Strengthen the inductive hypothesis.

- *Subtract* a low-order term.
- *Inductive hypothesis*: $T(k) \le c_1k^2 - c_2k$ for $k < n$.
- $T(n) = 4T(n/2) + n$

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

  $= c_1n^2 - 2c_2n + n$

  $= c_1n^2 - c_2n - (c_2n - n)$

  $\le c_1n^2 - c_2n$ if $c_2 \ge 1$.

  Pick c_1 big enough to handle the initial conditions.

Substitution Method

For the substitution method:

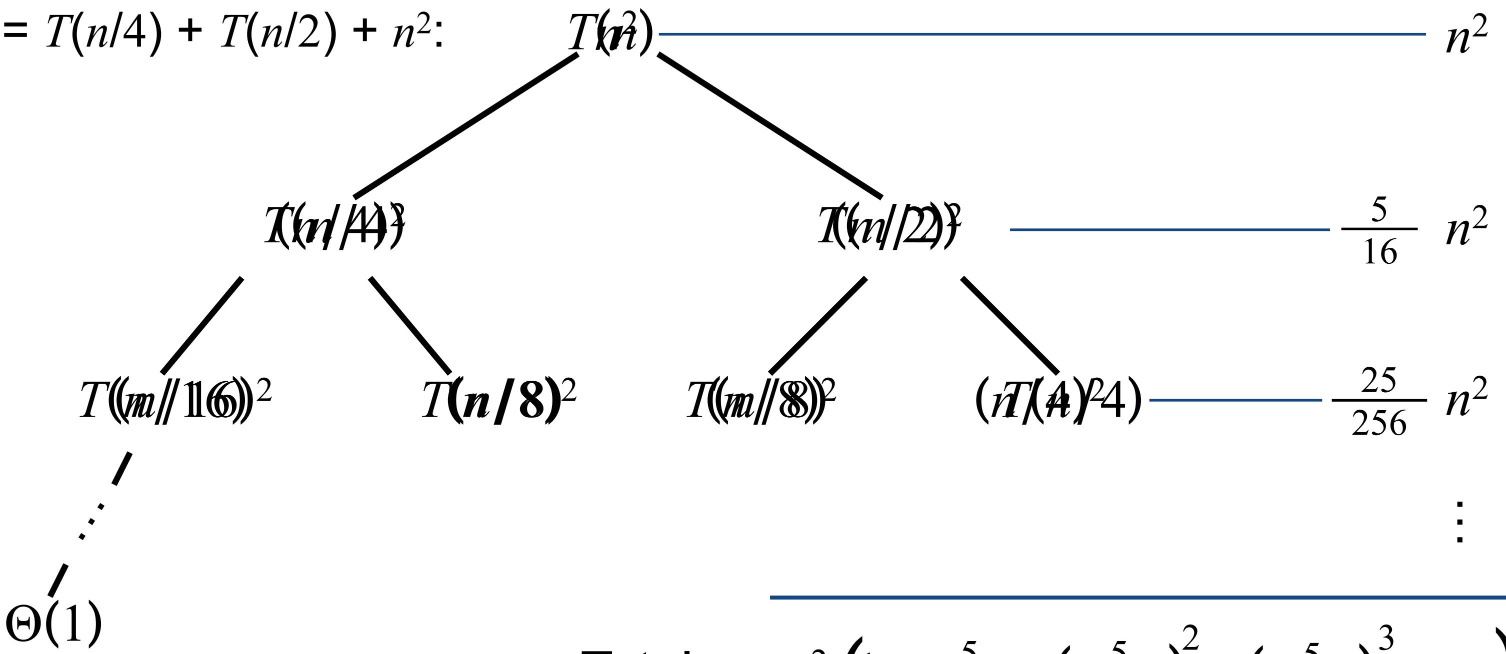
- The constant must be identified in the additive term
- The upper bounds O and the lower bounds Ω must be shown separately. A different constant may be needed for each proof.

Recursion-Tree Method

- A recursion tree models the associated costs (time) of a recursive execution of an algorithm.
- The recursion-tree method promotes intuition.
- The recursion tree method is good for generating guesses for the substitution method.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (a Gaussian distribution).

Example of Recursion Tree

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



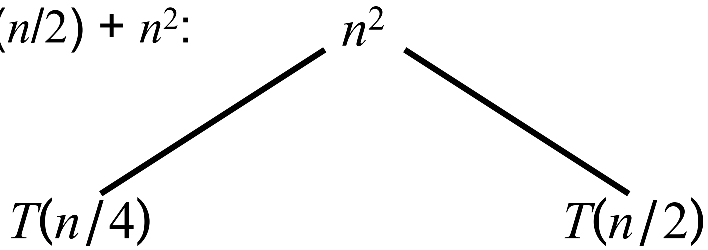
$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$$
$$= \Theta(n^2)$$

Example of Recursion Tree

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

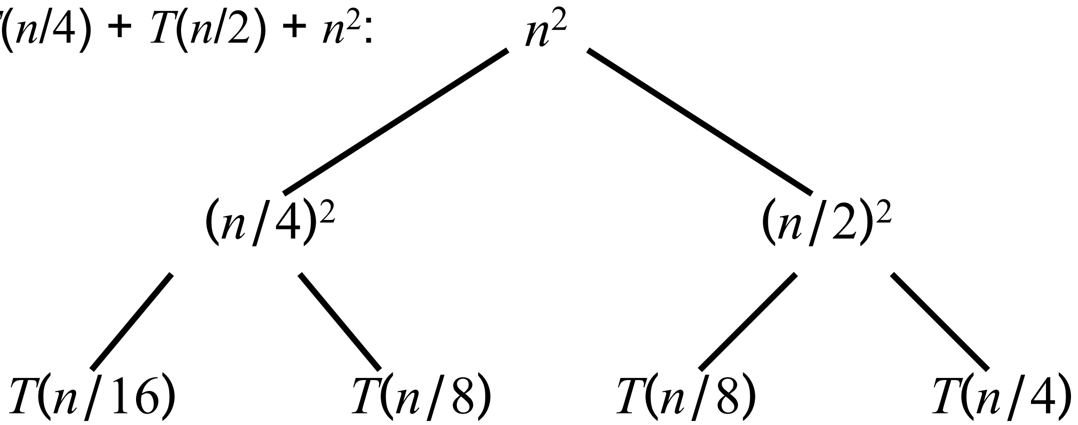
Example of Recursion Tree

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



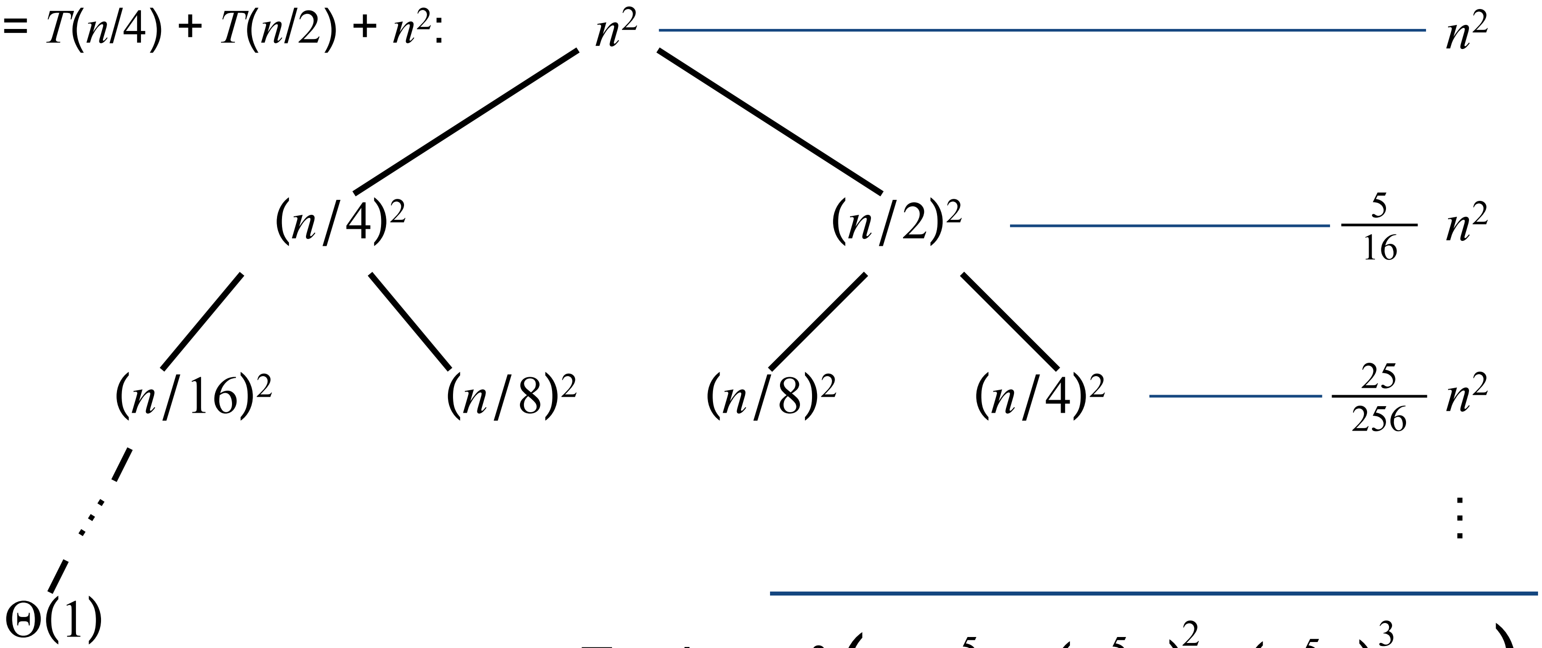
Example of Recursion Tree

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



Example of Recursion Tree

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



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

Recursion Trees

- Recursion trees are best used to generate a guess for the substitution method
- The generated guess can then be verifies by substitution method

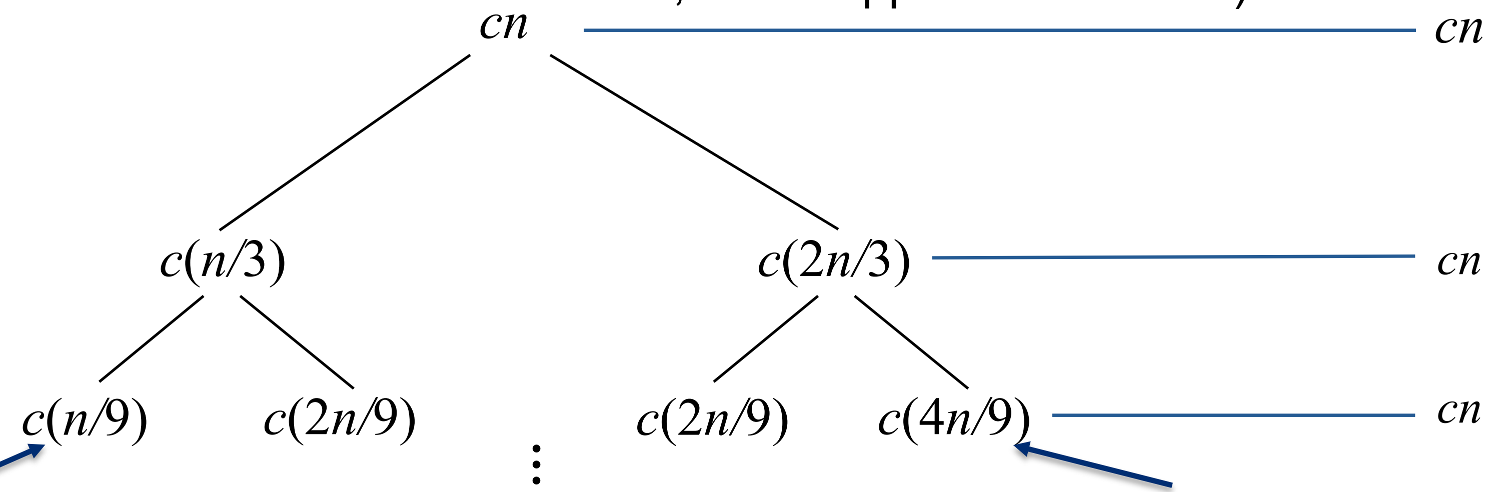
Recursion Tree Example

$T(n) = T(n/3) + T(2n/3) + \Theta(n)$

- For upper bound, rewrite as $T(n) = T(n/3) + T(2n/3) + \Theta(n)$
- For lower bound, rewrite as $T(n) = T(n/3) + T(2n/3) + \Theta(n)$

Recursion Tree Example cont.

By summing across each level, the recursion tree shows the cost at each level of recursion (minus the costs of recursive calls, which appear in subtrees):



Leftmost branch diminishes after $\log_3 n$ levels

Rightmost branch diminishes after $\log_{3/2} n$ levels

Recursion Tree Example cont.

- There are $\log_3 n$ full levels, and after $\log_{3/2} n$ levels, the problem size is down to 1.
- Each level contributes $\leq cn$.
- Lower bound guess: $\geq dn \log_3 n = \Omega(n \lg n)$ for some positive constant d.
- Upper bound guess: $\leq dn \log_{3/2} n = O(n \lg n)$ for some positive constant d.
- Then prove by substitution.

Master Method

- Many divide-and-conquer recurrence equations have the form:
  $T(n) = a\,T(n/b) + f(n)$,
  - $a \geq 1$, $b > 1$ are constants.
  - $f(n)$ is asymptotically positive.
  - n/b may not be an integer, but we ignore floors and ceilings.

- The master method applies to recurrences of the form

$$T(n) = \begin{cases} c & \text{if } \; n < d \\ aT(n/b) + f(n) & \text{if } \; n \geq d \end{cases}$$

- Requires memorization of three cases.

The Master Theorem

Theorem 4.1 (*Masters Theorem*)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on nonnegative integers by the recurrence

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

where we can replace n/b by floor(n/b) or ceil(n/b).

$T(n)$ can be bounded asymptotically in three cases:

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

Master theorem (reprise)

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

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

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

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

- $n^{\log_b a} = a^{\log_b n}$: Number of leaves in the recursion tree.
- $f(n) = O(n^{\log_b a - \varepsilon})$ implies that the sum of the cost of the nodes at each internal level are asymptotically smaller than the cost of leaves by a *polynomial* factor.
- Cost of the problem dominated by leaves, hence cost is $\Theta(n^{\log_b a})$.

Master theorem (reprise) cont.

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

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

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

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

- $n^{\log_b a} = a^{\log_b n}$: Number of leaves in the recursion tree.
- $f(n) = \Theta(n^{\log_b a})$ implies that the sum of the cost of the nodes at each level is asymptotically the same as the cost of leaves.
- There are $\Theta(\lg n)$ levels.
- Hence, total cost is $\Theta(n^{\log_b a} \lg n)$.

For MERGE-SORT

$a = 2, b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n \Rightarrow$ **Case 2** $(k = 0) \Rightarrow T(n) = \Theta(n \lg n)$.

Master theorem (reprise) cont.

$T(n) = aT(n/b) + f(n)$
Compare $f(n)$ with $n^{\log_b a}$

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

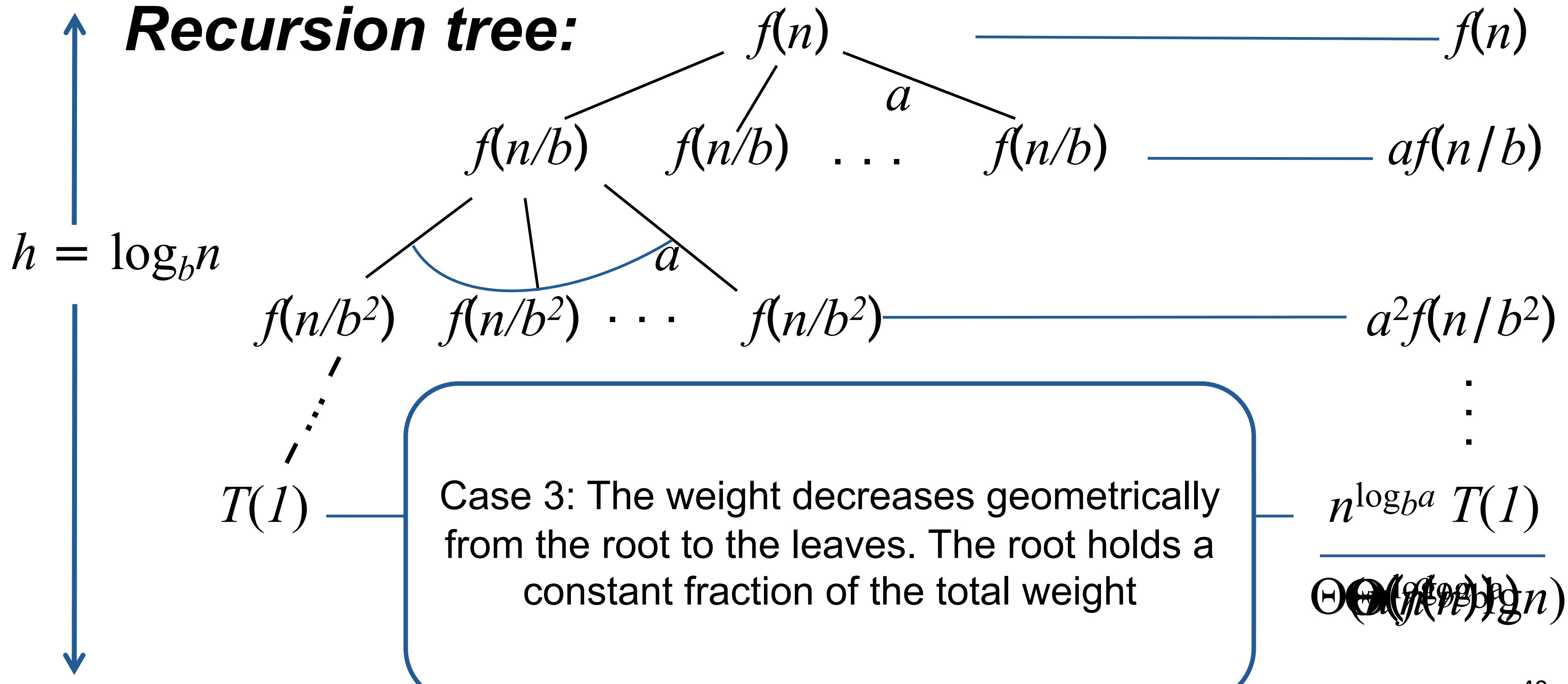
$f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor)
and $f(n)$ satisfies regularity condition that $T(n) = \Theta(f(n))$,
provided $a\, f(n/b) \leq c\, f(n)$, for some constant $c < 1$

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

If $f(n) = \Omega(n^{\log_b a+\varepsilon})$

- $n^{\log_b a} = a^{\log_b n}$: Number of leaves in the recursion tree.
- $f(n) = \Omega(n^{\log_b a+\varepsilon})$ implies that the cost is dominated by the root. Cost of the root is asymptotically larger than the sum of the cost of the leaves by a polynomial factor.
- Hence, cost is $\Theta(f(n))$.

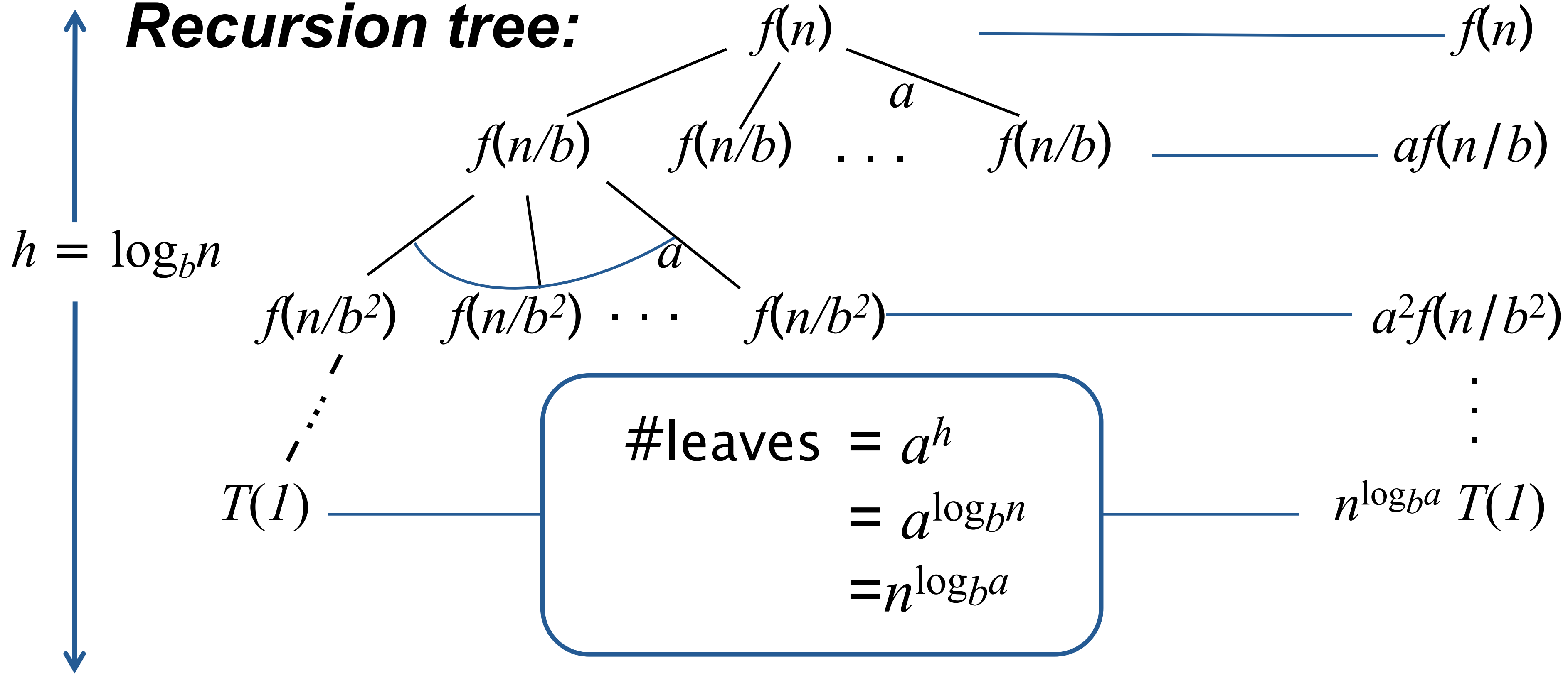
Idea of Master Theorem



Case 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight

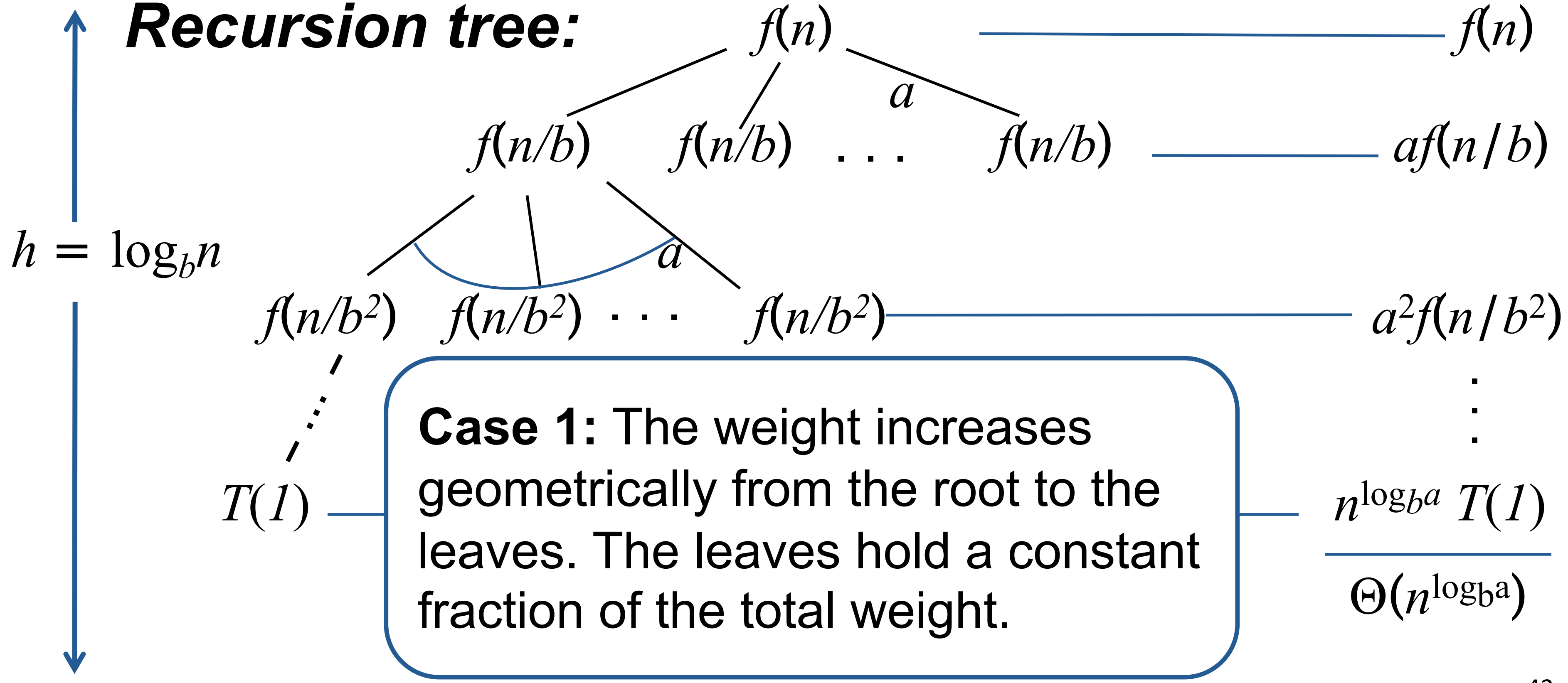
Idea of Master Theorem

Recursion tree:



Idea of Master Theorem

Recursion tree:



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

Recursion tree:

$f(n)$ — $f(n)$

a

$f(n/b)$ $f(n/b)$ . . . $f(n/b)$ — $af(n/b)$

a

$f(n/b^2)$ $f(n/b^2)$ . . . $f(n/b^2)$ — $a^2 f(n/b^2)$

$h = \log_b n$

$\vdots$

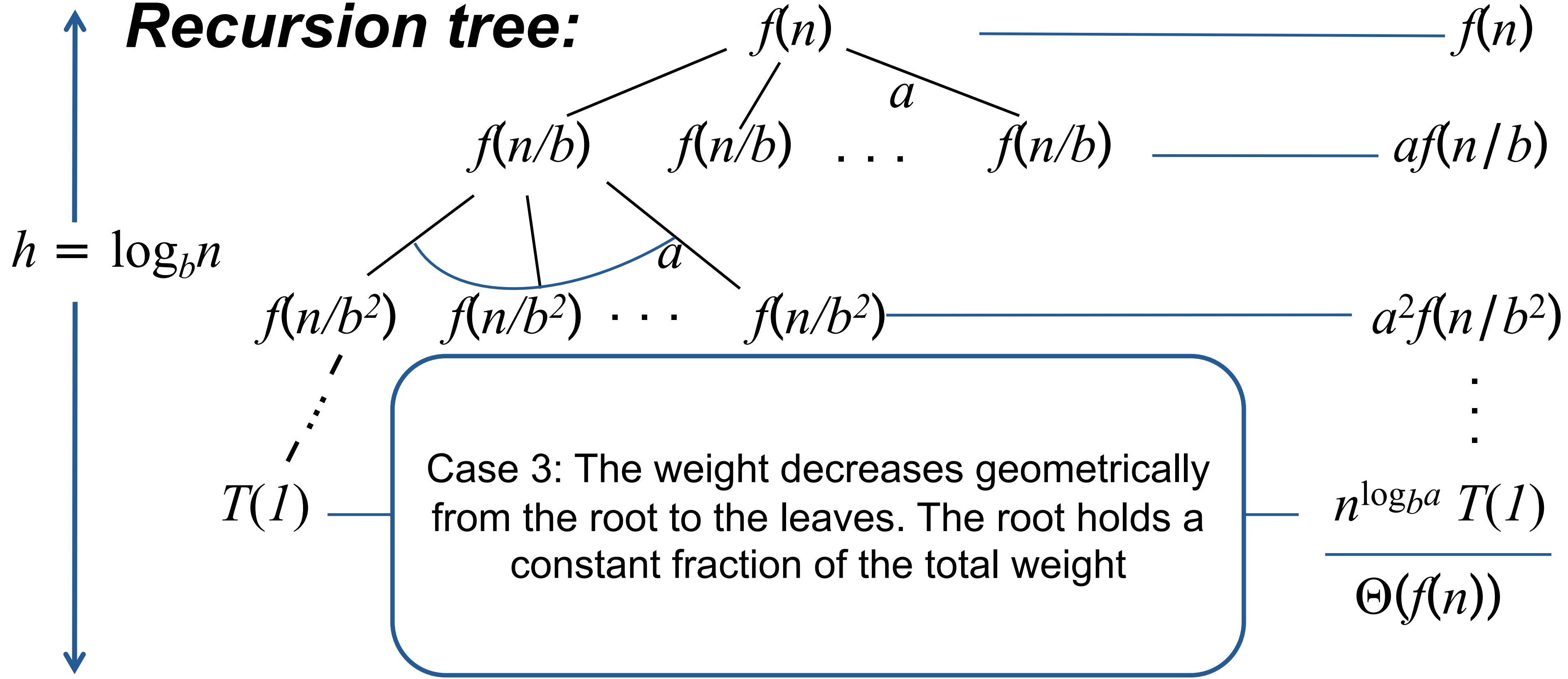
$T(1)$ — $n^{\log_b a}\, T(1)$

Case 2: ($k = 0$) The weight is approximately the same on each of the $\log_b n$ levels.

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

Idea of Master Theorem

Recursion tree:



Case 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight

Case 1 Examples

- $T(n) = 4T(n/2) + n$
  - $a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$
  - $f(n) = n = O(n^{2-\varepsilon})$ for $\varepsilon = 1$ **Case 1** applies
  - Therefore, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

- $T(n) = 16T(n/4)+n$
  - $a = 16$, $b = 4 \Rightarrow n^{\log_b a} = n^{\log_4 16} = n^2$.
  - $f(n) = n = O(n^{\log_b a-\varepsilon}) = O(n^{2-\varepsilon})$, where $\varepsilon = 1$ **Case 1** applies
  - Therefore, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

Case 2 Examples

- $T(n) = 4T(n/2) + n^2$
  - $a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2$
  - $f(n) = n^2 = \Theta(n^2 \lg^0 n)$, that is, $k = 0$ **Case 2** applies
  - Therefore, $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n^2 \lg n)$

- $T(n) = T(3n/7) + 1$
  - $a = 1, b = 7/3 \Rightarrow n^{\log_b a} = n^{\log_{7/3} 1} = n^0 = 1$
  - $f(n) = 1 = \Theta(n^{\log_b a})$ **Case 2 applies**
  - Therefore, $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n)$

Case 3 Examples

- $T(n) = 4T(n/2) + n^3$
  - $a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$
  - $f(n) = n^3 = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$ **Case 3** applies
  - **Case 3:** $f(n)$
  - and $4(n/2)^3 \leq cn^3$ (reg. cond.) for $c = 1/2$.
  - Therefore, $T(n) = \Theta(f(n)) = \Theta(n^3)$.

- $T(n) = 3T(n/4) + n \lg n$
  - $a = 3$, $b=4$, thus $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
  - $f(n) = n \lg n = \Omega(n^{\log_4 3 + \varepsilon})$ where $\varepsilon \approx 0.2$ **Case 3** applies
  - Therefore, $T(n) = \Theta(f(n)) = \Theta(n \lg n)$.

Master Method *does not* apply Examples

- $T(n) = 4T(n/2) + n^2/\lg n$
  - $a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$
  - $f(n) = n^2/\lg n$.
  - Master method *does not* apply.
  - In particular, for every constant $\varepsilon > 0$, we have $n^\varepsilon = \omega(\lg n)$.

- $T(n) = 2T(n/2) + n \lg n$
  - $a = 2$, $b=2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$
  - $f(n) = n \lg n$
  - $f(n)$ is asymptotically larger than $n^{\log_b a}$, but not polynomially larger.
  - The ratio $\lg n$ is asymptotically less than n^ε for any positive ε.
  - Thus, the Master method *does not* apply here.

Master Theorem – Proof for exact powers

Proof when n is an exact power of b.

Three steps.

1. Reduce the problem of solving the recurrence to the problem of evaluating an expression that contains a summation.
2. Determine bounds on the summation.
3. Combine 1 and 2.

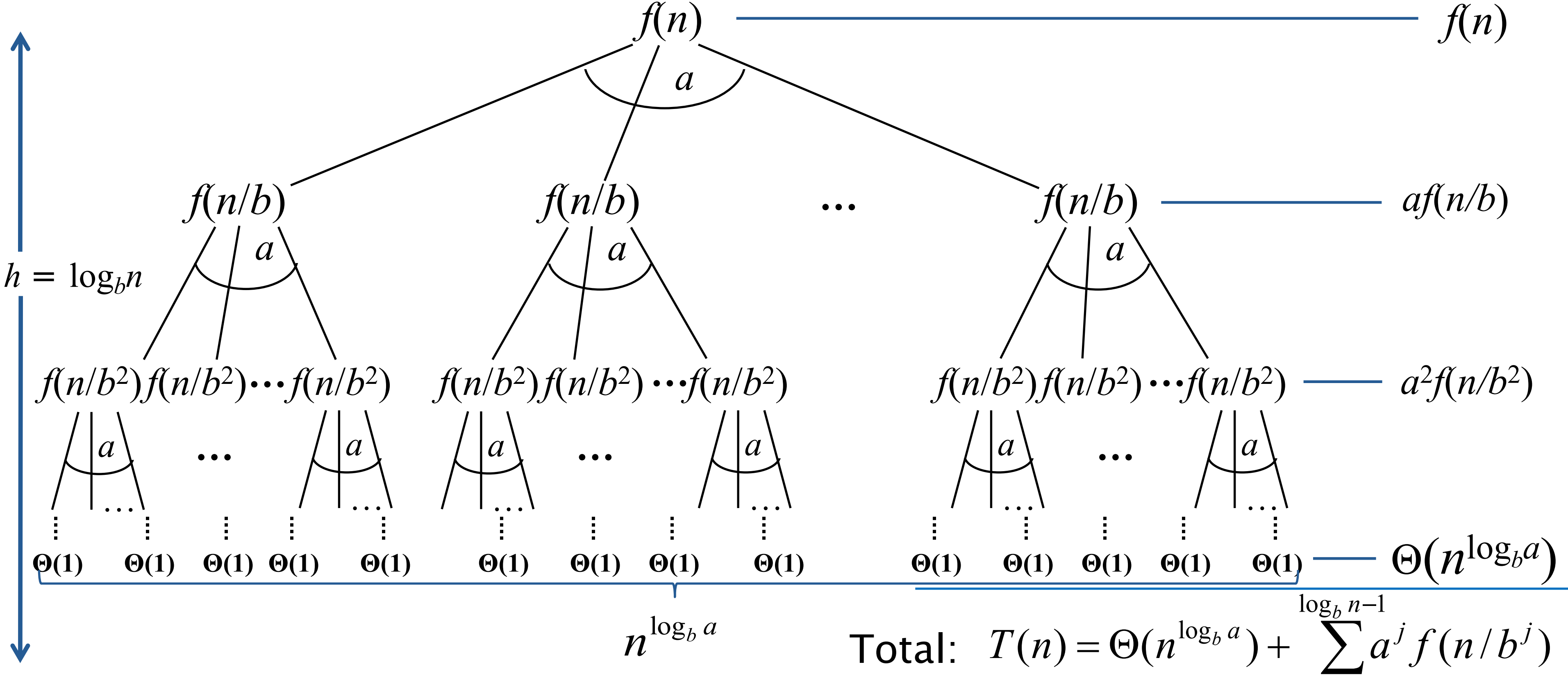
Iterative "Proof" of the Master Theorem

- Using iterative substitution, determine if a pattern can be found:

$$
\begin{aligned}
T(n) &= aT(n/b) + f(n) \\
&= a(aT(n/b^2)) + f(n/b)) + bn \\
&= a^2T(n/b^2) + af(n/b) + f(n) \\
&= a^3T(n/b^3) + a^2 f(n/b^2) + af(n/b) + f(n) \\
&= \ldots \\
&= a^{\log_b n} T(1) + \sum_{j=0}^{(\log_b n)-1} a^j f(n/b^j) \\
&= n^{\log_b a} T(1) + \sum_{j=0}^{(\log_b n)-1} a^j f(n/b^j)
\end{aligned}
$$

- Now distinguish between the three cases as
  - The first term is dominant
  - Each part of the summation is equally dominant
  - The summation is a geometric series

Recursion-tree Example



$$\text{Total: } T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

References

- T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "*Introduction to Algorithms, Third Edition*", 2009

- MIT Prof. Erik D. Demaine and MIT Prof. Charles E. Leiserson, Lecture Notes, Slides and Videos