CH08-320201

Algorithms and Data Structures

ADS

Lecture 4

Dr. Kinga Lipskoch

Spring 2019

Solving Recurrences

- Merge Sort analysis required us to solve the recurrence:

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

- A recurrence (or recurrence relation) is an equation that recursively defines a sequence (given an initial term).
- How can we generally solve recurrences?

Three Recurrence Solving Methods

- Substitution method
- Recursion tree
- Master method

Substitution Method

- The substitution method is based on some intuition.
- It executes the following steps:
  - Guess the form of the solution.
  - Verify by induction.
  - Solve for constants.

Example (1)

- Consider the recurrence $T(n) = 4T(n/2) + n$ with the base case $T(1) = \Theta(1)$.
- Prove O and Ω separately.
- Guess that $T(n) = O(n^3)$.
- Verify by induction:
  - Check the base case $n = 1$.
  - Assuming $T(k) \leq ck^3$ for $k < n$ show $T(n) \leq cn^3$.

Example (2)

Induction step:

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4c(n/2)^3 + n \\
&= (c/2)n^3 + n \\
&= cn^3 - ((c/2)n^3 - n) \leftarrow \textit{desired} - \textit{residual} \\
&\le cn^3 \leftarrow \textit{desired}
\end{aligned}
$$

whenever $(c/2)n^3 - n \ge 0$, for example, if $c \ge 2$ and $n \ge 1$.

residual

Example (3)

- Was our guess a good one?
- Was it tight enough?
- Make a new guess: $T(n) = O(n^2)$.
- Try to prove by induction.
  - Base step: as before
  - Induction step:
    Assuming $T(k) \le ck^2$ for $k < n$, show $T(n) \le cn^2$.

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4c(n/2)^2 + n \\ &= cn^2 + n \\ &= cn^2 - (-n) \quad [\text{ desired} - \text{residual }] \\ &\le cn^2 \quad \text{for } \textbf{\textit{no}} \text{ choice of } c > 0. \text{ Lose!} \end{aligned}$$

Example (4)

- Idea: Adjust hypothesis by subtracting a lower-order term.
- Induction step:
  Assuming $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$
  show $T(n) \leq c_1 n^2 - c_2 n$.

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

Example (5)

Finally, solve for constants:

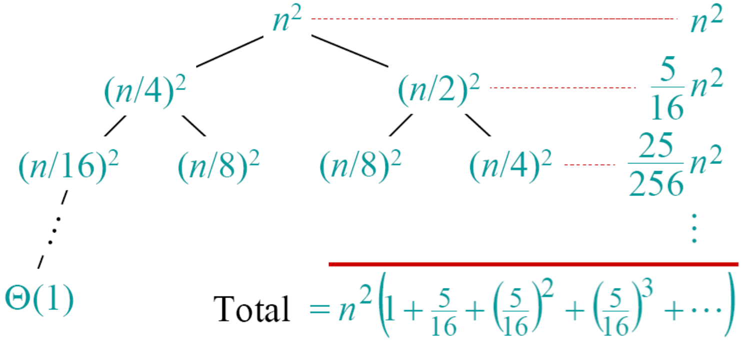
- Pick c_2 according to the induction proof from before ($c_2 > 1$).
- Pick c_1 large enough to handle the base case:
  - $T(1) = \Theta(1)$ implies $T(1) = O(1)$
  - $T(1) \leq c_1 1^2 - c_2 1 = c_1 - c_2$, where $c_2 > 1$
  - Therefore, $c_1 > c_2$

Recursion Tree

- For the Merge Sort analysis, we used a recursion tree
- A recursion tree models the costs (time) of a recursive execution of an algorithm
- This does not necessarily lead to a reliable solution
- However, the recursion-tree method promotes intuition
- It is good for generating guesses for the substitution method

Example (1)

Consider the recurrence $T(n) = T(n/4) + T(n/2) + n^2$ with the base case $T(1) = \Theta(1)$.



Example (2)

Considering the geometric series from below we get $T(n) = \Theta(n^2)$.

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

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

Master Method (1)

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.

It distinguishes 3 common cases by comparing
$f(n)$ with $n^{log_b a}$

Master Method (2)

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

1. If $f(n) = O(n^{log_b a - \epsilon})$ for some constant $\epsilon > 0$
   – $f(n)$ is polynomially smaller than $n^{log_b a}$ –,
   then $T(n) = \Theta(n^{log_b a})$.
2. If $f(n) = \Theta(n^{log_b a})$,
   then $T(n) = \Theta(n^{log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{log_b a + \epsilon})$ for some constant $\epsilon > 0$
   – $f(n)$ is polynomially larger than $n^{log_b a}$ –
   and $af(n/b) \leq cf(n)$ – regularity condition – for some constant $c < 1$,
   then $T(n) = \Theta(f(n))$.

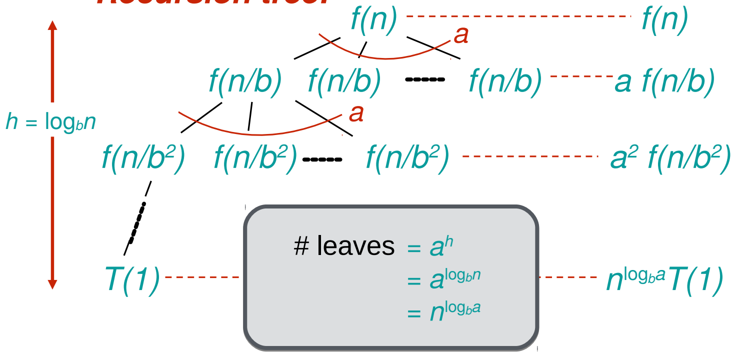
Master Method Not Always Applicable

- There is a gap between cases 1 and 2 when $f(n)$ is smaller than $n^{log_b a}$ but not polynomially smaller
- There is a gap between cases 2 and 3 when $f(n)$ is larger than $n^{log_b a}$ but not polynomially larger
- If the regularity condition in case 3 fails to hold or you are in one of the gaps then you cannot use the master method to solve the recurrence

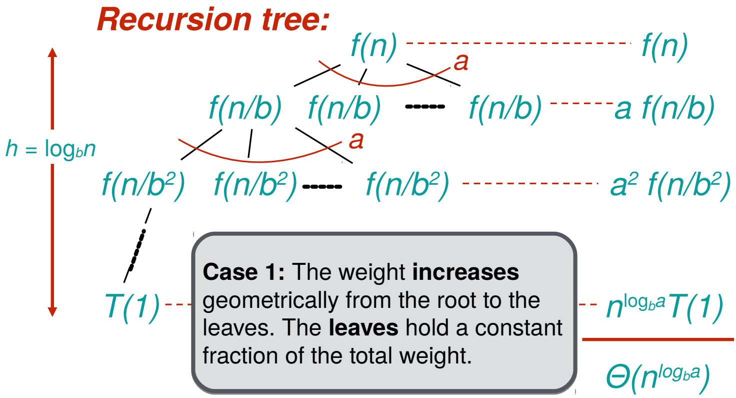
Idea of the Master Theorem (1)

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

Recursion tree:



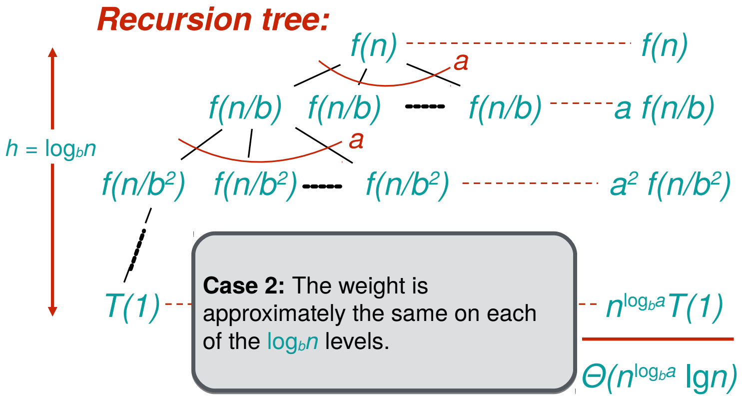
Idea of the Master Theorem (2)



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

1

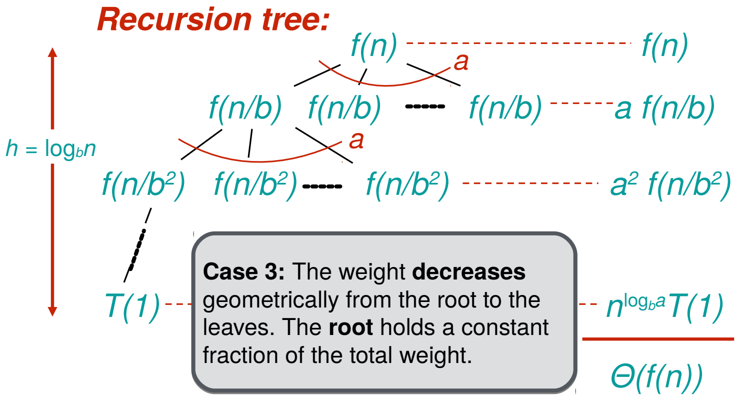
Idea of the Master Theorem (3)



Case 2: The weight is approximately the same on each of the $\log_b n$ levels.

1

Idea of the Master Theorem (4)



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

1

Example (1)

$T(n) = 4T(n/2) + n$
$a = 4, b = 2$

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

Case 1: $f(n) = O(n^{2-\epsilon})$ for $\epsilon = 1$
Thus, $T(n) = \Theta(n^2)$.

Example (2)

$T(n) = 4T(n/2) + n^2$
$a = 4, b = 2$

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

Case 2: $f(n) = \Theta(n^2)$,
Thus, $T(n) = \Theta(n^2 \lg n)$.

Example (3)

$T(n) = 4T(n/2) + n^3$
$a = 4, b = 2$

$n^{log_b a} = n^2$
$f(n) = n^3$

Case 3: $f(n) = \Omega(n^{2+\epsilon})$ for $\epsilon = 1$
and $4(n/2)^3 \leq cn^3$ for $c = 1/2$ (regularity condition)
Thus, $T(n) = \Theta(n^3)$.

Example (4)

$T(n) = 4T(n/2) + n^2/\lg n$
$a = 4, b = 2$

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

Master method does not apply
(for every constant $\epsilon > 0$, we have $n^\epsilon = \omega(\lg n)$)