

# Fundamental Algorithms

## Chapter 2b: Recurrences

Michael Bader

Winter 2011/12



# Recurrences

## Definition

A **recurrence** is an (in-)equality that defines (or characterizes) a function in terms of its values on smaller arguments.

## Examples:

- Time complexity of MergeSort:

$$T_{\text{MS}}(n) = \begin{cases} c_1 & \text{for } n \leq 1 \\ 2T_{\text{MS}}\left(\frac{n}{2}\right) + c_2n & \text{for } n \geq 2 \end{cases}$$

- or, given in  $\Theta$ -notation:

$$T_{\text{MS}}(n) = \begin{cases} \Theta(1) & \text{for } n \leq 1 \\ 2T_{\text{MS}}\left(\frac{n}{2}\right) + \Theta(n) & \text{for } n \geq 2 \end{cases}$$

- inequality for the BFPRT algorithm (to compute the median ...):

$$T_{\text{BF}}(n) \leq \begin{cases} \Theta(C) & \text{for } n \leq C \\ T_{\text{BF}}\left(\lceil \frac{n}{5} \rceil\right) + T_{\text{BF}}\left(\frac{7}{10}n + 6\right) + O(n) & \text{for } n > C \end{cases}$$

# The Substitution Method

- step 1:** guess the type of the solution
- step 2:** find the respective parameters, and prove that the resulting function satisfies the recurrence (e.g. by induction)

**Example:** (MergeSort recurrence)

$$T_{\text{MS}}(n) = \begin{cases} c_1 & \text{for } n \leq 1 \\ 2T_{\text{MS}}\left(\frac{n}{2}\right) + c_2n & \text{for } n \geq 2 \end{cases}$$

1. guess solution:  $T(n) = a n \log_2 n + b n$
2. determine the correct values for the parameters  $a$  and  $b$

Note:

- Is recurrence formula correct? Shouldn't be  $2T_{\text{MS}}\left(\lceil \frac{n}{2} \rceil\right) + c_2n$ ?

# Solving the MergeSort Recurrence via Substitution

**For  $n \leq 1$ :**  $T_{\text{MS}}(n) = c_1$

- $T_{\text{MS}}(1) = a \cdot 1 \cdot \log_2(1) + b \cdot 1 \stackrel{!}{=} c_1$
- can only be true, if  $b := c_1$

**For  $n > 1$ :**  $T_{\text{MS}}(n) = 2T_{\text{MS}}\left(\frac{n}{2}\right) + c_2n$

- insert  $T_{\text{MS}}(n) = an \log_2 n + c_1n$  into equation:

$$\begin{aligned} an \log_2 n + c_1n &= 2 \left( a \frac{n}{2} \log_2 \left( \frac{n}{2} \right) + c_1 \frac{n}{2} \right) + c_2n \\ \Leftrightarrow an \log_2 n + c_1n &= an (\log_2 n - 1) + c_1n + c_2n \\ \Leftrightarrow &0 = -an + c_2n \\ \Leftrightarrow &a = c_2 \end{aligned}$$

- therefore:  $T_{\text{MS}}(n) = c_2n \log_2 n + c_1n$

# The Recursion-Tree Method (or Iteration Method)

## General Steps:

1. draw a tree of all recursive function calls
2. state the local costs for each node (function call) of the tree
3. sum up the costs of all nodes on each level of the tree

## Possible Results:

- a sum of costs-per-level that can be added up easily
- an easier recurrence for the costs-per-level
- a good guess for the substitution method

Example: → MergeSort recurrence

# The Master Theorem

## Prerequisites:

- constants  $a \geq 1, b > 1$  ( $a, b \in \mathbb{R}$ ); a function  $f(n)$
- recurrence given by

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), \quad T(1) \in \Theta(1)$$

## Then, $T(n)$ can be bounded asymptotically as follows:

1. if  $f(n) \in O(n^{\log_b a - \epsilon})$  for some  $\epsilon > 0$ , then  $T(n) \in \Theta(n^{\log_b a})$
2. if  $f(n) \in \Theta(n^{\log_b a})$ , then  $T(n) \in \Theta(n^{\log_b a} \log n)$
3. if  $f(n) \in \Omega(n^{\log_b a + \epsilon})$  for some  $\epsilon > 0$ , and  
if  $af\left(\frac{n}{b}\right) \leq cf(n)$  for some constant  $c < 1$  and all  $n > n_0$ ,  
then  $T(n) \in \Theta(f(n))$

**Proof:** see textbook (Cormen et al.)

# The Master Theorem – Remarks

## Interpreting the Master Theorem:

- for  $f(n) = 0$ , i.e.,  $T(n) = aT\left(\frac{n}{b}\right)$ , a solution is  $T_0(n) = n^{\log_b a}$

$$aT_0\left(\frac{n}{b}\right) = a\left(\frac{n}{b}\right)^{\log_b a} = a\frac{n^{\log_b a}}{a} = n^{\log_b a} = T_0(n)$$

- The master theorem compares the non-recursive part of the costs,  $f(n)$ , with this solution  $T_0(n)$
- case 1:  $f(n) \in O(T_0(n) \cdot n^{-\epsilon})$ ,  
 $\Rightarrow$  costs of recursion dominate, and  $T(n) \in \Theta(n^{\log_b a})$
- case 2:  $f(n) \in \Theta(T_0(n))$ ,  
 $\Rightarrow$  costs are balanced, and  $T(n) \in \Theta(n^{\log_b a} \log n)$
- case 3:  $f(n) \in \Omega(T_0(n) \cdot n^\epsilon)$ ,  
 $\Rightarrow$  costs  $f(n)$  dominate, and  $T(n) \in \Theta(f(n))$

## The Master Theorem – Remarks (2)

### Floor and Ceil:

- if in  $aT\left(\frac{n}{b}\right)$  the fraction  $\frac{n}{b}$  occurs as  $\lceil \frac{n}{b} \rceil$  or  $\lfloor \frac{n}{b} \rfloor$ , the theorem still holds
  - situations as in  $T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor\right)$  (compare MergeSort recurrence) are also covered  $\Rightarrow 2T\left(\frac{n}{2}\right)$
- $\Rightarrow$  the master theorem will cover many (but not all) divide-and-conquer recurrences

### Technicalities of the Theorem:

- case 1:  $f(n) \in O(T_0(n))$  is not sufficient:  
 $f(n)$  needs to be polynomially smaller than  $T_0(n)$
- case 3:  $f(n) \in \Omega(T_0(n))$  is not sufficient:  
 $f(n)$  needs to be polynomially larger than  $T_0(n)$



# The Master Theorem – Examples

**MergeSort:**  $T(n) = 2T(\frac{n}{2}) + f(n)$ , where  $f(n) \in \Theta(n)$

- $a = 2$  and  $b = 2$ , therefore  $T_0(n) = n^{\log_2 2} = n$
- case 2 applies:  $f(n) \in \Theta(n)$ , therefore  $T(n) \in \Theta(n \log(n))$

**Expensive Merge:**  $T(n) = 2T(\frac{n}{2}) + f(n)$ , but  $f(n) \in \Theta(n^2)$

- again  $a = 2$  and  $b = 2$ , thus  $T_0(n) = n^{\log_2 2} = n$
- now  $f(n) \in \Omega(n^{1+\epsilon})$  for any  $0 < \epsilon < 1$
- therefore case 3 applies:  $T(n) \in \Theta(f(n)) = \Theta(n^2)$

**Odd-Even MergeSort:**  $T(n) = 2T(\frac{n}{2}) + f(n)$ , with  $f(n) \in \Theta(n \log n)$

- still  $a = 2$  and  $b = 2$ , thus  $T_0(n) = n^{\log_2 2} = n$
- now,  $f(n) \in \Omega(n)$ , but  $f(n) \notin \Omega(n^{1+\epsilon})$  for any  $\epsilon > 0$
- thus, the Master theorem **does not apply**