Fundamental Algorithms

Chapter 2b: Recurrences

Harald Räcke

Winter 2015/16

Recurrences

Definition

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

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:

$$T_{\text{MS}}(n) = \begin{cases} c_1 & \text{for } n \leq 1 \\ 2T_{\text{MS}}\left(\frac{n}{2}\right) + c_2 n & \text{for } n \geq 2 \end{cases}$$

- or, given in Θ-notation:

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

- inequality for the BFPRT algorithm (to find the median):

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

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)

$$T_{\mathrm{MS}}(n) = \begin{cases} c_1 & \text{for } n \leq 1 \\ 2\,T_{\mathrm{MS}}\left(\frac{n}{2}\right) + c_2 n & \text{for } n \geq 2 \end{cases}$$

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

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)

$$T_{\text{MS}}(n) = \begin{cases} c_1 & \text{for } n \leq 1 \\ 2T_{\text{MS}}\left(\frac{n}{2}\right) + c_2 n & \text{for } n \geq 2 \end{cases}$$

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

Note:

- Is recurrence formula correct? Shouldn't it be $2T_{\text{MS}}\left(\left\lceil \frac{n}{2} \right\rceil\right) + c_2 n$?

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 $b := c_1$

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 $n > 1$: $T_{\text{MS}}(n) = 2T_{\text{MS}}\left(\frac{n}{2}\right) + c_2 n$

- insert $T_{\text{MS}}(n) = an\log_2 n + c_1 n$ into equation:

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

- therefore: $T_{\text{MS}}(n) = c_2 n\log_2 n + c_1 n$

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

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: $\rightarrow$ MergeSort recurrence

The Master Theorem

Prerequisites:

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

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

The Master Theorem

Prerequisites:

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

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

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

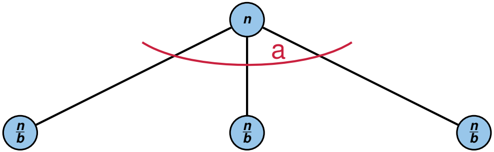
1. if $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$ for some $\epsilon > 0$, then $T(n) \in \Theta\left(n^{log_b a}\right)$
2. if $f(n) \in \Theta\left(n^{\log_b a}\right)$, then $T(n) \in \Theta\left(n^{log_b a} \log n\right)$
3. if $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ for some $\epsilon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

Proof: see textbook (Cormen et al.)

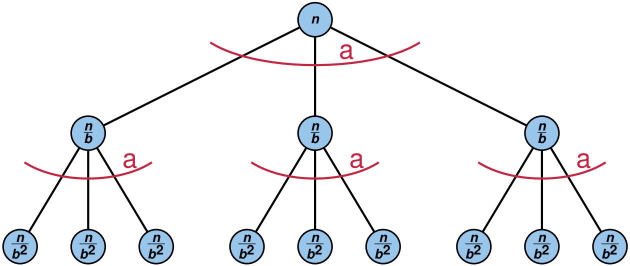
The Recursion Tree

The Recursion Tree

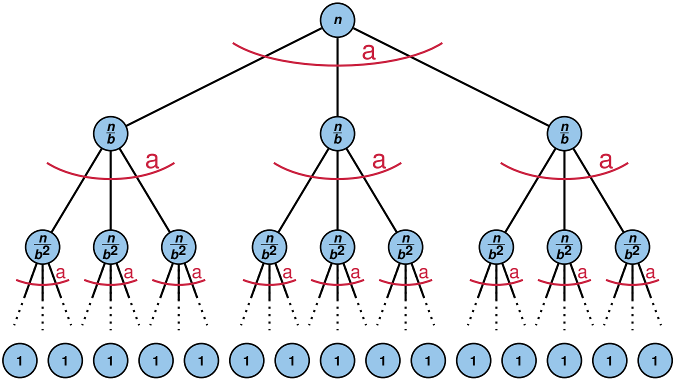
The Recursion Tree



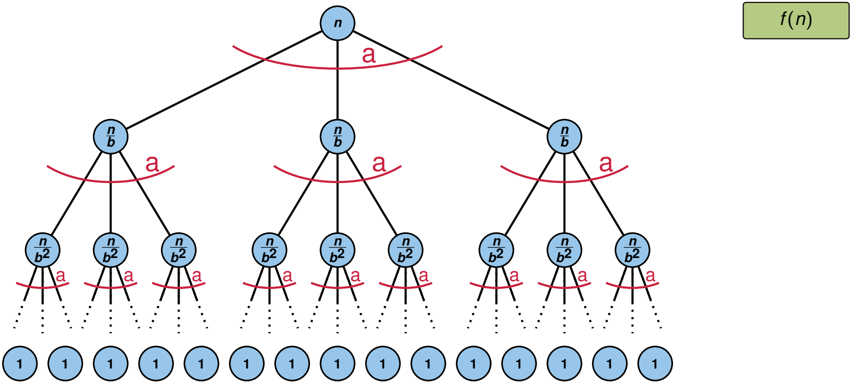
The Recursion Tree



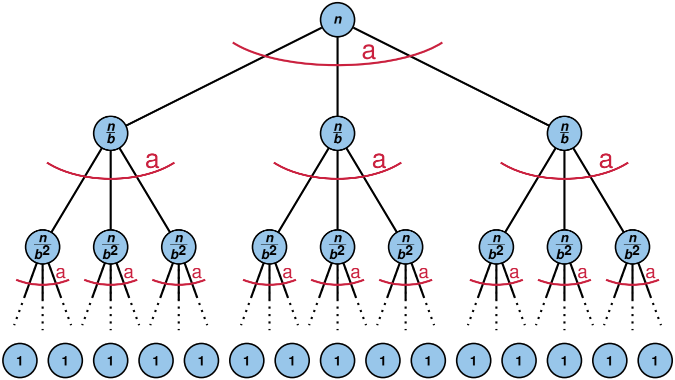
The Recursion Tree



The Recursion Tree



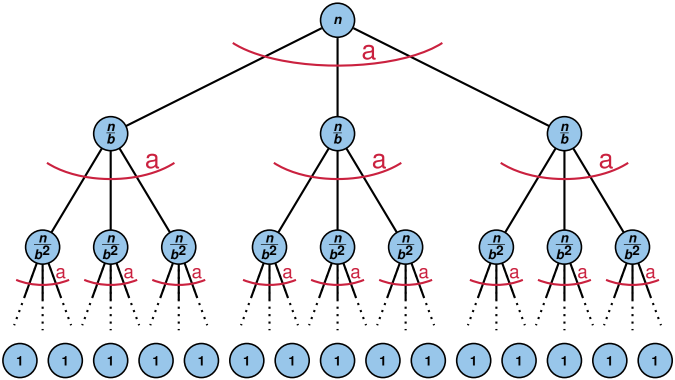
The Recursion Tree



$f(n)$

$af(\frac{n}{b})$

The Recursion Tree

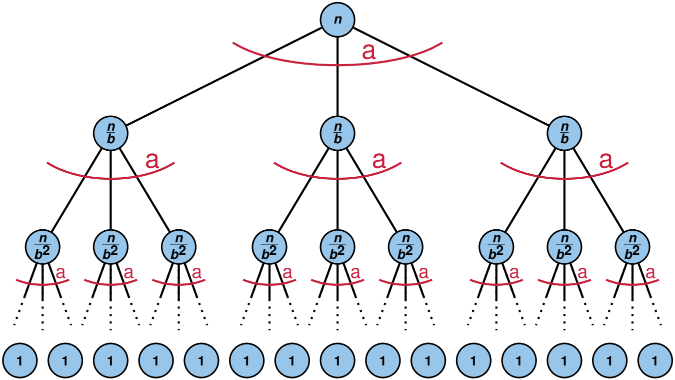


$f(n)$

$af(\frac{n}{b})$

$a^2 f(\frac{n}{b^2})$

The Recursion Tree



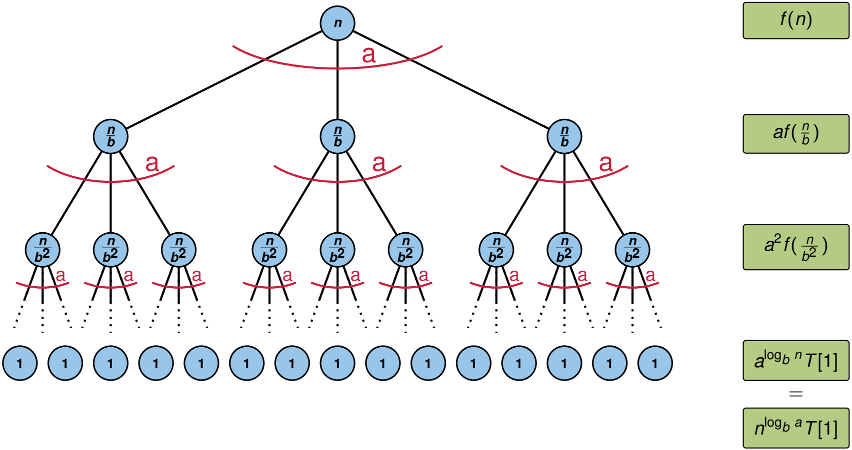
$f(n)$

$af(\frac{n}{b})$

$a^2 f(\frac{n}{b^2})$

$a^{\log_b n} T[1]$

The Recursion Tree



The Master Theorem – Remarks

Interpreting the Master Theorem:

- for $f(n) = 0$ and $T[1] = 1$ the solution is equal to the number of leafs in the recursion tree: $T_0(n) = n^{\log_b a}$
- 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 $T(n) \in \Theta\left(n^{log_b a}\right)$
- case 2: $f(n) \in \Theta(T_0(n))$,
  $\Rightarrow$ costs are balanced, and $T(n) \in \Theta\left(n^{log_b a} \log n\right)$
- 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 $\left\lceil\frac{n}{b}\right\rceil$ or $\left\lfloor\frac{n}{b}\right\rfloor$, the theorem still holds
- situations as in $T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right)$ (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

The Master Theorem – Remarks (2)

Floor and Ceil:

- if in $aT\left(\frac{n}{b}\right)$ the fraction $\frac{n}{b}$ occurs as $\left\lceil\frac{n}{b}\right\rceil$ or $\left\lfloor\frac{n}{b}\right\rfloor$, the theorem still holds
- situations as in $T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right)$ (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: $f(n) \in O\left(T_0(n)\right)$ is not sufficient:
  $f(n)$ needs to be polynomially smaller than $T_0(n)$
- case 3: $f(n) \in \Omega\left(T_0(n)\right)$ 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 $f(n) \in \Theta(n)$

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 $T(n) \in \Theta(n \log(n))$

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 $f(n) \in \Theta(n^2)$

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 $0 < \epsilon < 1$
- and $a\,f(n/b) = \frac{1}{2}n^2 \leq cn^2$ for any $\frac{1}{2} < c < 1$
- therefore case 3 applies: $T(n) \in \Theta(f(n)) = \Theta(n^2)$

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 $0 < \epsilon < 1$
- and $a\, f(n/b) = \frac{1}{2} n^2 \leq cn^2$ for any $\frac{1}{2} < c < 1$
- 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 $f(n) \in \Theta(n \log n)$

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 $0 < \epsilon < 1$
- and $a\,f(n/b) = \frac{1}{2}n^2 \leq cn^2$ for any $\frac{1}{2} < c < 1$
- 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**