

Datenstrukturen und Algorithmen

Vorlesung 3: Rekursionsgleichungen (K4)

Prof. Dr. Erika Ábrahám

Theorie Hybrider Systeme
Informatik 2

[http://ths.rwth-aachen.de/teaching/ss-14/
datenstrukturen-und-algorithmen/](http://ths.rwth-aachen.de/teaching/ss-14/datenstrukturen-und-algorithmen/)

Diese Präsentation verwendet in Teilen Folien von Joost-Pieter Katoen.

24. April 2014



Übersicht

- 1 Beispiele für rekursive Algorithmen
- 2 Rekursionsgleichungen
- 3 Lösen von Rekursionsgleichungen
 - Die Substitutionsmethode
 - Rekursionsbäume
 - Mastertheorem

Übersicht

- 1 Beispiele für rekursive Algorithmen
- 2 Rekursionsgleichungen
- 3 Lösen von Rekursionsgleichungen
 - Die Substitutionsmethode
 - Rekursionsbäume
 - Mastertheorem

Die Teile-und-Beherrsche-Methode

Teile-und-Beherrsche (divide-and-conquer)

- ▶ **Teile** das Problem in eine Anzahl von Teilproblemen auf
- ▶ **Beherrsche** die Teilprobleme
 - ▶ kleine Teilprobleme: direktes Lösen
 - ▶ große Teilprobleme: Teile-und-Beherrsche
- ▶ **Verbinde** die Lösungen der Teilprobleme zur Lösung des Ausgangsproblems

Summenberechnung

Iterative Berechnung ($n \in \mathbb{N}$)

$$sum(n) = \sum_{i=0}^n i$$

Rekursive Berechnung ($n \in \mathbb{N}$)

$$sum(n) = \begin{cases} 0 & \text{falls } n = 0 \\ sum(n-1) + n & \text{sonst.} \end{cases}$$

Multiplikation von Binärzahlen der Länge n

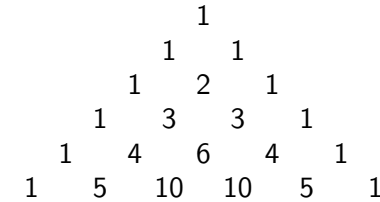
- ▶ Bisher: Addition und Multiplikation haben Zeitaufwand $\mathcal{O}(1)$
- ▶ Addition: $\mathcal{O}(n)$ Bitoperationen
- ▶ Multiplikation: Auch $\mathcal{O}(n)$?

1. Multiplikationsalgorithmus

$$(a_1 \dots a_n) \cdot (b_1 \dots b_n) = \begin{array}{r} (a_1 \dots a_n) \cdot b_1 \\ (a_1 \dots a_n) \cdot b_2 \\ \dots \\ (a_1 \dots a_n) \cdot b_n \\ \hline \text{Summenbildung} \end{array}$$

$n - 1$ Additionen von Zahlen der Länge $\leq 2n \Rightarrow \mathcal{O}(n^2)$!

Pascalsches Dreieck



Rekursive Berechnung ($n, m \in \mathbb{N}, m \leq n$)

$$pas(n, m) = \begin{cases} 1 & \text{falls } m = 0 \vee m = n \\ pas(n-1, m-1) + pas(n-1, m) & \text{sonst.} \end{cases}$$

Multiplikation von Binärzahlen der Länge n

Beobachtung

Seien a, b Binärzahlen mit $n = 2^m$ bits.

$$a = \underbrace{a_1 \dots a_{n/2}}_{a'} \underbrace{a_{n/2+1} \dots a_n}_{a''} \quad b = \underbrace{b_1 \dots b_{n/2}}_{b'} \underbrace{b_{n/2+1} \dots b_n}_{b''}$$

Dann

$$a \cdot b = (a' \cdot 2^{\frac{n}{2}} + a'') \cdot (b' \cdot 2^{\frac{n}{2}} + b'') \\ = a' \cdot b' \cdot 2^n + a' \cdot b'' \cdot 2^{\frac{n}{2}} + a'' \cdot b' \cdot 2^{\frac{n}{2}} + a'' \cdot b''$$

2. Multiplikationsalgorithmus

$$mul(a, b, n) = \begin{cases} a \cdot b & \text{falls } n = 1 \\ mul(a', b', \frac{n}{2}) \cdot 2^n + mul(a', b'', \frac{n}{2}) \cdot 2^{\frac{n}{2}} + \\ mul(a'', b', \frac{n}{2}) \cdot 2^{\frac{n}{2}} + mul(a'', b'', \frac{n}{2}) & \text{sonst.} \end{cases}$$

Übersicht

- 1 Beispiele für rekursive Algorithmen
- 2 **Rekursionsgleichungen**
- 3 Lösen von Rekursionsgleichungen
 - Die Substitutionsmethode
 - Rekursionsbäume
 - Mastertheorem

Rekursionsgleichungen für \mathcal{O} (Worst-Case-Laufzeit)

Rekursionsgleichung für eine obere Grenze der Worst-Case Laufzeit $T_A(n)$ eines Algorithmus A mit Parameterlänge n aufstellen

- ▶ **Elementare** Operationen und ihre **Kosten** werden bestimmt.
- ▶ Bei **sequentieller Komposition** werden die Kosten **addiert**.
- ▶ Bei **bedingter Verzweigung** wird das **Maximum** der beiden Zweige genommen.
- ▶ Der Aufruf eines **Unterprogrammes B** mit **Parameterlänge $f(n)$** wird mit den **Kosten $T_B(f(n))$** der Ausführung des Unterprogrammes versehen.
- ▶ Auch **rekursive Aufrufe** werden wie **Unterprogramm-Aufrufe** behandelt.

Rekursionsgleichungen

Rekursionsgleichung

- ▶ Eine **Rekursionsgleichung** ist eine Gleichung oder eine Ungleichung, die eine Funktion durch ihre eigenen Funktionswerte für kleinere Eingaben beschreibt.
- ▶ Wir benutzen Rekursionsgleichungen um die **Laufzeit** von rekursiven Algorithmen zu beschreiben.
- ▶ Zur Ermittlung der Laufzeit **lösen** wir diese Gleichungen.

Rekursionsgleichungen für Worst-Case-Laufzeit

Beispiel: Teile-und-Beherrsche (divide-and-conquer)

- ▶ **Teile** das Problem in eine Anzahl von Teilproblemen auf
- ▶ **Beherrsche** die Teilprobleme
 - ▶ kleine Teilprobleme: direktes Lösen
 - ▶ große Teilprobleme: Teile-und-Beherrsche
- ▶ **Verbinde** die Lösungen der Teilprobleme zur Lösung des Ausgangsproblems

$$\text{Laufzeit} = \text{Laufzeit}_{\text{Teilen}} + \text{Laufzeit}_{\text{Beherrschen}} + \text{Laufzeit}_{\text{Verbinden}}$$

Summenberechnung

Rekursive Berechnung ($n \in \mathbb{N}$)

$$sum(n) = \begin{cases} 0 & \text{falls } n = 0 \\ sum(n-1) + n & \text{sonst.} \end{cases}$$

Laufzeit

$$T_{sum}(n) = \begin{cases} 0 & \text{falls } n = 0 \\ 1 + T_{sum}(n-1) + 1 & \text{sonst.} \end{cases}$$

Multiplikation von Binärzahlen der Länge n

2. Multiplikationsalgorithmus

$$mul(a, b, n) = \begin{cases} a \cdot b & \text{falls } n = 1 \\ mul(a', b', \frac{n}{2}) \cdot 2^n + mul(a', b'', \frac{n}{2}) \cdot 2^{\frac{n}{2}} + \\ mul(a'', b', \frac{n}{2}) \cdot 2^{\frac{n}{2}} + mul(a'', b'', \frac{n}{2}) & \text{sonst.} \end{cases}$$

Laufzeit

$$T_{mul}(n) = \begin{cases} 1 & \text{falls } n = 1 \\ d_1 \cdot n + 4 \cdot T_{mul}(\frac{n}{2}) + d_2 \cdot n & \text{sonst.} \end{cases}$$

Pascalsches Dreieck

Rekursive Berechnung ($n, m \in \mathbb{N}, m \leq n$)

$$pas(n, m) = \begin{cases} 1 & \text{falls } m = 0 \vee m = n \\ pas(n-1, m-1) + pas(n-1, m) & \text{sonst.} \end{cases}$$

Laufzeit

$$T_{pas}(n, m) = \begin{cases} 0 & \text{falls } m = 0 \vee m = n \\ 3 + T_{pas}(n-1, m-1) + T_{pas}(n-1, m) + 1 & \text{sonst.} \end{cases}$$

Übersicht

- 1 Beispiele für rekursive Algorithmen
- 2 Rekursionsgleichungen
- 3 Lösen von Rekursionsgleichungen
 - Die Substitutionsmethode
 - Rekursionsbäume
 - Mastertheorem

Ansätze zur Lösung von Rekursionsgleichungen

Lösungsansätze

- ▶ **Substitutionsmethode:** Lösung raten, per Induktion Korrektheit beweisen
- ▶ **Rekursionsbaum-Methode:** Rekursionsbaum aufstellen, Summenformeln beschränken
- ▶ **Mastertheorem:** Liefert Lösungen für Rekursionsgleichungen einer bestimmten Form, einfach aber nicht immer anwendbar

Beispiel Substitutionsmethode: Summenberechnung

Laufzeit

$$T_{sum}(n) = \begin{cases} 0 & \text{falls } n = 0 \\ T_{sum}(n-1) + 2 & \text{sonst.} \end{cases}$$

Beispiel

- ▶ Vermutung: $T_{sum}(n) \in \mathcal{O}(n)$
- ▶ Zeige: $\exists c > 0. \exists n_0 \geq 0. \forall n \geq n_0. T_{sum}(n) \leq c \cdot n$
- ▶ Induktionsanfang $n_0 = 0$: $T_{sum}(0) = 0 \leq c \cdot 0$ for all $c > 0$.
- ▶ Induktionsannahme: $T_{sum}(k) \leq c \cdot k$ für alle $0 \leq k < n$
- ▶ Induktionsschritt:

$$T_{sum}(n) = T_{sum}(n-1) + 2 \leq c \cdot (n-1) + 2 = c \cdot n + (2-c) \leq c \cdot n$$
 für $c \geq 2$.

Die Substitutionsmethode

Substitutionsmethode

Die Substitutionsmethode besteht aus zwei Schritten:

1. **Rate** die Form der Lösung, durch z.B. durch
 - ▶ scharfes Hinsehen oder
 - ▶ kurze Eingaben ausprobieren und einsetzen.
2. **Vollständige Induktion** um die Konstanten zu finden und zu zeigen, dass das Geratene eine Lösung ist.

Bemerkungen

- ▶ Diese Methode ist sehr leistungsfähig, aber
- ▶ kann nur angewendet werden wenn es relativ einfach ist, die Form der Lösung zu erraten.

Die Substitutionsmethode: Annahmenverstärkung

Induktiv vs. invariant

Eine asymptotische Schranke kann nicht immer direkt mit der vollständigen Induktion bewiesen werden.

$$\text{induktiv} \xrightarrow{\quad} \text{invariant}$$

Beispiel

- ▶ $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1$
- ▶ $T(n) \in \mathcal{O}(n)$
- ▶ Zu zeigen: $\exists c > 0. \exists n_0 \geq 0. \forall n \geq n_0. T(n) \leq c \cdot n$
- ▶ $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \leq c \lceil n/2 \rceil + c \lfloor n/2 \rfloor + 1 = c \cdot n + 1$

Die Substitutionsmethode: Annahmenverstärkung

Beispiel

- ▶ $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1$
 - ▶ $T(n) \in \mathcal{O}(n)$
 - ▶ Zu zeigen: $\exists c > 0. \exists n_0 \geq 0. \forall n \geq n_0. T(n) \leq c \cdot n$
 - ▶ $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \leq c \lceil n/2 \rceil + c \lfloor n/2 \rfloor + 1 = c \cdot n \boxed{+ 1}$

 - ▶ Verstärke die Invariante und zeige $T(n) \in \mathcal{O}(n-1)$ und $n-1 \in \mathcal{O}(n)$
 - ▶ Zu zeigen: $\exists c > 0. \exists n_0 \geq 0. \forall n \geq n_0. T(n) \leq c \cdot (n-1)$
- $$\begin{aligned}
 T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \\
 &\leq c(\lceil n/2 \rceil - 1) + c(\lfloor n/2 \rfloor - 1) + 1 \\
 &= c \cdot (n-1) + (1-c) \\
 &\leq c \cdot (n-1)
 \end{aligned}$$
- für $c \geq 1$.

Übersicht

- 1 Beispiele für rekursive Algorithmen
- 2 Rekursionsgleichungen
- 3 Lösen von Rekursionsgleichungen
 - Die Substitutionsmethode
 - Rekursionsbäume
 - Mastertheorem

Die Substitutionsmethode: Variablentransformation

Manchmal hilft eine Variablentransformation beim Lösen.

Beispiel

$T(n) = 2 \cdot T(\lfloor \sqrt{n} \rfloor) + \log n$ für $n > 0$
 (Im Folgenden vernachlässigen wir die Rundung von Werten.)

$$\begin{aligned}
 T(n) &= 2 \cdot T(\sqrt{n}) + \log n && | \text{ Variablentransformation } m = \log n \\
 \Leftrightarrow T(2^m) &= 2 \cdot T(2^{m/2}) + m && | \text{ Umbenennung } T(2^m) = S(m) \\
 \Leftrightarrow S(m) &= 2 \cdot S(m/2) + m && | \text{ Substitutionsmethode} \\
 \Leftrightarrow S(m) &\leq c \cdot m \cdot \log m \\
 \Leftrightarrow S(m) &\in \mathcal{O}(m \cdot \log m) && | m = \log n \\
 \Leftrightarrow T(n) &\in \mathcal{O}(\log n \cdot \log \log n)
 \end{aligned}$$

Raten der Lösung durch Rekursionsbäume

Grundidee

Stelle das Ineinander-Einsetzen als Baum dar, indem man Buch über das aktuelle Rekursionsargument und die nichtrekursiven Kosten führt.

Rekursionsbaum

1. Jeder **Knoten** stellt die Kosten eines Teilproblems dar.
 - ▶ Die Wurzel stellt die zu analysierenden Kosten $T(n)$ dar.
 - ▶ Die Blätter stellen die Kosten der Basisfälle dar, z.B. $T(0)$ oder $T(1)$.
2. Wir summieren die Kosten innerhalb jeder **Ebene** des Baumes.
3. Die **Gesamtkosten** := summieren über die **Kosten aller Ebenen**.

Hinweis

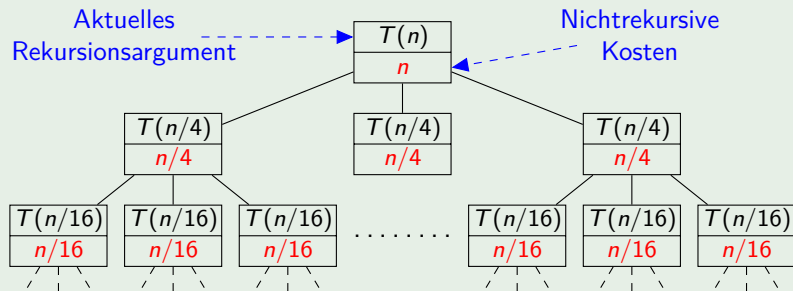
Ein Rekursionsbaum ist sehr nützlich, um eine Lösung zu raten, die dann mit Hilfe der Substitutionsmethode überprüft werden kann.

Der Baum selber reicht jedoch meistens nicht als Beweis.

Rekursionsbaum: Beispiel

Beispiel

Der Rekursionsbaum von $T(n) = 3 \cdot T(n/4) + n$ sieht etwa so aus:



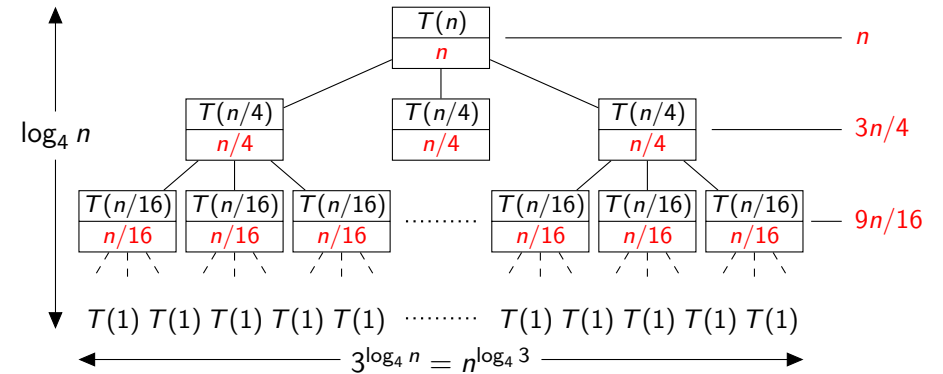
Rekursionsbaum: Beispiel

Eine obere Schranke für die Komplexität erhält man nun folgendermaßen:

$$\begin{aligned}
 T(n) &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{4}\right)^i \cdot n + c \cdot n^{\log_4 3} && | \text{ Vernachlässigen kleinerer Terme} \\
 &< \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i \cdot n + c \cdot n^{\log_4 3} && | \text{ Geometrische Reihe} \\
 &= \frac{1}{1 - (3/4)} \cdot n + c \cdot n^{\log_4 3} && | \text{ Umformen} \\
 &= 4 \cdot n + c \cdot n^{\log_4 3} && | \text{ Asymptotische Ordnung bestimmen} \\
 &&& \text{ setze ein, dass } \log_4 3 < 1
 \end{aligned}$$

$T(n) \in \mathcal{O}(n)$.

Rekursionsbaum: Beispiel



$$T(n) = \underbrace{\sum_{i=0}^{\log_4 n - 1}}_{\text{Summe über alle Ebenen}} \underbrace{\left(\frac{3}{4}\right)^i \cdot n}_{\text{Kosten pro Ebene}} + \underbrace{c \cdot n^{\log_4 3}}_{\text{Gesamtkosten für die Blätter mit } T(1) = c}$$

Korrektheit

Wir können die Substitutionsmethode benutzen, um die Vermutung zu bestätigen dass:

$T(n) \in \mathcal{O}(n)$ eine obere Schranke von $T(n) = 3 \cdot T(n/4) + n$ ist.

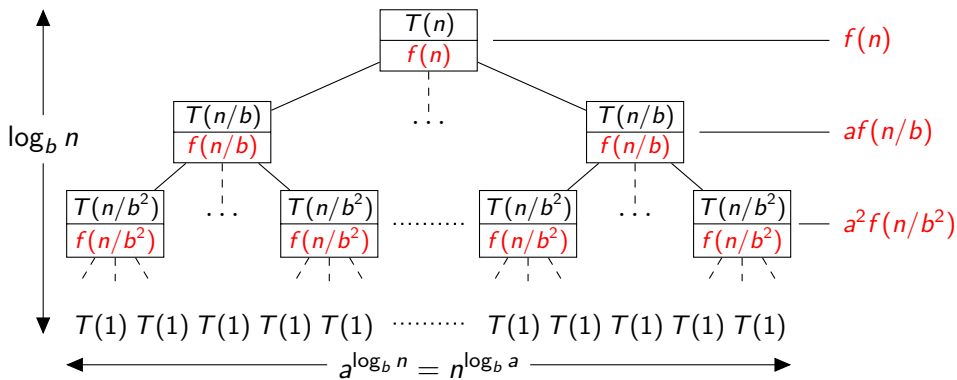
$$\begin{aligned}
 T(n) &= 3 \cdot T(n/4) + n && | \text{ Induktionshypothese} \\
 &\leq 3c \cdot n/4 + n \\
 &= \frac{3}{4}c \cdot n + n \\
 &= \left(\frac{3}{4}c + 1\right) \cdot n && | \text{ für } c \geq 4 \text{ folgt:} \\
 &\leq c \cdot n
 \end{aligned}$$

Und wir stellen fest, dass es ein n_0 gibt, sodass $T(n_0) \leq c \cdot n_0$ ist.

Übersicht

- 1 Beispiele für rekursive Algorithmen
- 2 Rekursionsgleichungen
- 3 Lösen von Rekursionsgleichungen
 - Die Substitutionsmethode
 - Rekursionsbäume
 - Mastertheorem

Erinnerung: Rekursionsbaum $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$



$$T(n) = \underbrace{\sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)}_{\text{Summe über alle Ebenen}} + \underbrace{c \cdot n^{\log_b a}}_{\text{Gesamtkosten für die Blätter mit } T(1) = c}$$

Mastertheorem

Allgemeine Form der Rekursionsgleichung

Eine Rekursionsgleichung für die Komplexitätsanalyse sieht meistens folgendermaßen aus:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

wobei $a > 0$, $b > 1$ gilt und $f(n)$ eine gegebene Funktion ist.

Intuition:

- ▶ Das zu analysierende Problem teilt sich jeweils in a Teilprobleme auf
- ▶ Jedes dieser Teilprobleme hat die Größe $\frac{n}{b}$
- ▶ Die Kosten für das Aufteilen eines Problems und Kombinieren der Teillösungen sind $f(n)$.

Das Mastertheorem

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \text{ mit } a \geq 1 \text{ und } b > 1.$$

- ▶ Anzahl der Blätter im Rekursionsbaum: n^E mit $E = \log_b a$.

Mastertheorem

Wenn	Dann
1. $f(n) \in \mathcal{O}(n^{E-\epsilon})$ für ein $\epsilon > 0$	$T(n) \in \Theta(n^E)$
2. $f(n) \in \Theta(n^E)$	$T(n) \in \Theta(n^E \cdot \log n)$
3. $f(n) \in \Omega(n^{E+\epsilon})$ für ein $\epsilon > 0$ und $a \cdot f(n/b) \leq d \cdot f(n)$ für ein $d < 1$ und n hinreichend groß	$T(n) \in \Theta(f(n))$

- ▶ Bemerke, dass das Mastertheorem nicht alle Fälle abdeckt.

Das Mastertheorem verstehen

In jedem der 3 Fälle wird die Funktion $f(n)$ mit $n^E = n^{\log_b a}$ verglichen.

Mastertheorem: Intuition

Wenn	Dann
1. $f(n)$ polynomiell kleiner ist als n^E	$T(n) \in \Theta(n^E)$
2. $f(n)$ und n^E die gleiche Größe haben	$T(n) \in \Theta(n^E \cdot \log n)$
3. $f(n)$ ist polynomiell größer als n^E und erfüllt $a \cdot f(n/b) \leq d \cdot f(n)$	$T(n) \in \Theta(f(n))$

Nicht abgedeckte Fälle:

- $f(n)$ ist kleiner als n^E , jedoch nicht polynomiell kleiner.
- $f(n)$ ist größer als n^E , jedoch nicht polynomiell größer.
- $f(n)$ ist polynomiell größer als n^E , erfüllt nicht $a \cdot f(n/b) \leq d \cdot f(n)$.

Anwendung des Mastertheorems

Beispiel

$$T(n) = 4 \cdot T(n/2) + n^3$$

- ▶ Somit: $a = 4$, $b = 2$ und $f(n) = n^3$; $E = \log 4 / \log 2 = 2$.
- ▶ Wegen $E = 2$ gelten Fälle 1 und 2 offenbar **nicht**.
- ▶ Da $f(n) = n^3 \in \Omega(n^{2+\varepsilon})$ für $\varepsilon = 1$, könnte Fall 3 gelten.
- ▶ Überprüfe: gilt $f(n/2) \leq \frac{d}{4} \cdot f(n)$ für ein $d < 1$ und hinreichend große n ?
- ▶ Dies liefert $\frac{1}{8}n^3 \leq \frac{d}{4} \cdot n^3$, und dies gilt für alle $\frac{1}{2} \leq d < 1$ (und n)
- ▶ Somit gilt Fall 3 tatsächlich und wir folgern: $T(n) \in \Theta(n^3)$

Anwendung des Mastertheorems

Beispiel

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

- ▶ Somit: $a = 4$, $b = 2$ und $f(n) = n$; $E = \log 4 / \log 2 = 2$.
- ▶ Da $f(n) = n \in \mathcal{O}(n^{2-\varepsilon})$, gilt Fall 1: $T(n) \in \Theta(n^2)$

Beispiel

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

- ▶ Somit: $a = 4$, $b = 2$ und $f(n) = n^2$; $E = \log 4 / \log 2 = 2$.
- ▶ Da $f(n) = n^2 \notin \mathcal{O}(n^{2-\varepsilon})$, gilt Fall 1 nicht.
- ▶ Aber weil $f(n) = n^2 \in \Theta(n^2)$, gilt Fall 2: $T(n) \in \Theta(n^2 \cdot \log n)$

Das Mastertheorem ist nicht immer anwendbar

Beispiel

$$T(n) = 4 \cdot T(n/2) + \frac{n^2}{\log n}$$

- ▶ Also gilt: $a = 4$, $b = 2$ und $f(n) = n^2 / \log n$; $E = 2$.

Fall 1 ist **nicht** anwendbar:

$$n^2 / \log n \notin \mathcal{O}(n^{2-\varepsilon}), \text{ da } f(n)/n^2 = (\log n)^{-1} \notin \mathcal{O}(n^{-\varepsilon}).$$

Fall 2 ist **nicht** anwendbar: $n^2 / \log n \notin \Theta(n^2)$.

Fall 3 ist **nicht** anwendbar:

$$f(n) \notin \Omega(n^{2+\varepsilon}), \text{ da } f(n)/n^2 = (\log n)^{-1} \notin \mathcal{O}(n^{+\varepsilon}).$$

⇒ Das Mastertheorem hilft hier überhaupt nicht weiter!

- ▶ Durch Substitution erhält man: $T(n) \in \Theta(n^2 \cdot \log \log n)$