

# Informatik II - Tutorium

– Sommersemester 2007 –

Joachim Wilke

<http://joachim-wilke.de/info2tut07.htm>

22/23. Mai 2007



Universität Karlsruhe (TH)

Forschungsuniversität · gegründet 1825

Quellennachweis & Dank an:  
Susanne Dinkler, Bernhard Müller, Michael Rother, wikipedia.de

# Übersicht



- 1 Organisatorisches
- 2 Und nochmal...
- 3 Übungsblatt 04
- 4 Aufwandsanalyse
- 5 Mastertheorem
- 6 Programmierkonzepte

- 1 Organisatorisches
- 2 Und nochmal...
- 3 Übungsblatt 04
- 4 Aufwandsanalyse
- 5 Mastertheorem
- 6 Programmierkonzepte

# Rechnerübung



- Bitte beachtet folgende Frist für das Vorführen von Programmen:  
Alle Programme der Übungsblätter 1-4 müssen bis spätestens 11./12. Juni 2007 vorgeführt werden um noch Punkte zu erzielen.
- Am 28./29. Mai 2007 gibt es keine Rechnerübungen.

- 1 Organisatorisches
- 2 Und nochmal...**
- 3 Übungsblatt 04
- 4 Aufwandsanalyse
- 5 Mastertheorem
- 6 Programmierkonzepte

# Und nochmal...



## Aufwandsanalyse ...

- 1 ... eignet sich um a priori ineffiziente Algorithmen zu verwerfen.
- 2 ... nutzt die drei Aufwandsklassen  $O(n)$ ,  $\Theta(n)$ ,  $\Phi(n)$ .
- 3 ... zeigt, dass Bubble-Sort im Best-Case besser als Quick-Sort ist.

## Durch das kontrollflussorientierte Testen von Programmen ...

- 1 ... kann man die Korrektheit von Rückgabewerten beweisen.
- 2 ... wird ein sogenannter White-Box-Test durchgeführt.
- 3 ... kann der Grad der Bedingungsüberdeckung ermittelt werden.

## Eine Rekurrenz ist eine Aufwandsfunktion, die ...

- 1 ... unabhängig von der Komplexität der Eingabemenge ist.
- 2 ... geschlossen definiert sein muss.
- 3 ... abhängig vom betrachteten Algorithmus ist.

# Und nochmal...



## Aufwandsanalyse ...

- ① ... eignet sich um a priori ineffiziente Algorithmen zu verwerfen.
- ② ... nutzt die drei Aufwandsklassen  $O(n)$ ,  $\Theta(n)$ ,  $\Phi(n)$ .
- ③ ... zeigt, dass Bubble-Sort im Best-Case besser als Quick-Sort ist.

## Durch das kontrollflussorientierte Testen von Programmen ...

- ① ... kann man die Korrektheit von Rückgabewerten beweisen.
- ② ... wird ein sogenannter White-Box-Test durchgeführt.
- ③ ... kann der Grad der Bedingungsüberdeckung ermittelt werden.

## Eine Rekurrenz ist eine Aufwandsfunktion, die ...

- ① ... unabhängig von der Komplexität der Eingabemenge ist.
- ② ... geschlossen definiert sein muss.
- ③ ... abhängig vom betrachteten Algorithmus ist.

# Und nochmal...



## Aufwandsanalyse ...

- ① ... eignet sich um a priori ineffiziente Algorithmen zu verwerfen.
- ② ... nutzt die drei Aufwandsklassen  $O(n)$ ,  $\Theta(n)$ ,  $\Phi(n)$ .
- ③ ... zeigt, dass Bubble-Sort im Best-Case besser als Quick-Sort ist.

## Durch das kontrollflussorientierte Testen von Programmen ...

- ① ... kann man die Korrektheit von Rückgabewerten beweisen.
- ② ... wird ein sogenannter White-Box-Test durchgeführt.
- ③ ... kann der Grad der Bedingungsüberdeckung ermittelt werden.

## Eine Rekurrenz ist eine Aufwandsfunktion, die ...

- ① ... unabhängig von der Komplexität der Eingabemenge ist.
- ② ... geschlossen definiert sein muss.
- ③ ... abhängig vom betrachteten Algorithmus ist.

# Und nochmal...



## Aufwandsanalyse ...

- ① ... eignet sich um a priori ineffiziente Algorithmen zu verwerfen.
- ② ... nutzt die drei Aufwandsklassen  $O(n)$ ,  $\Theta(n)$ ,  $\Phi(n)$ .
- ③ ... zeigt, dass Bubble-Sort im Best-Case besser als Quick-Sort ist.

## Durch das kontrollflussorientierte Testen von Programmen ...

- ① ... kann man die Korrektheit von Rückgabewerten beweisen.
- ② ... wird ein sogenannter White-Box-Test durchgeführt.
- ③ ... kann der Grad der Bedingungsüberdeckung ermittelt werden.

## Eine Rekurrenz ist eine Aufwandsfunktion, die ...

- ① ... unabhängig von der Komplexität der Eingabemenge ist.
- ② ... geschlossen definiert sein muss.
- ③ ... abhängig vom betrachteten Algorithmus ist.

- 1 Organisatorisches
- 2 Und nochmal...
- 3 Übungsblatt 04**
- 4 Aufwandsanalyse
- 5 Mastertheorem
- 6 Programmierkonzepte

# Übungsblatt 04



## Themen und Punkte

- Theorie: O-Notation, Rekurrenzen
- Anwendung: Suchen, Fibonacci, Würfelsimulation, Min-Cut
- Punkte: 24 Theorie, 22 Praxis

## Grundbegriffe und Themen zur Bearbeitung

- O-Kalkül, geschlossene Form, Mastertheorem
- Suchalgorithmen, Fibonacci, Aufwandsabschätzung
- zufallsgenerierte Zahlen, Min-Cut-Algorithmus

- 1 Organisatorisches
- 2 Und nochmal...
- 3 Übungsblatt 04
- 4 Aufwandsanalyse**
- 5 Mastertheorem
- 6 Programmierkonzepte

# Aufwandsanalyse



## Was bezwecken wir mit der Aufwandsanalyse?

Mittels Aufwandsanalysen lassen sich Algorithmen grob kategorisieren. So kann anhand der oberen Schranke eines Algorithmus z.B. gesagt werden, dass er im schlimmsten Fall (Worst-Case) signifikant langsamer ist, als ein schwer zu implementierender Zweiter.

Man kann somit a-priori ineffiziente Algorithmen verwerfen, ohne sie programmieren und testen zu müssen.

## Beispiel

BubbleSort ist ein einfacher, aber leider nicht sehr effizienter Algorithmus, weil man im schlimmsten Fall  $n^2$  Zahlen tauschen muss. Nicht ganz so einfach zu programmieren ist der Teile-und-Herrsche-Algorithmus "Insertion-Sort", der jedoch im schlimmsten Fall  $n \cdot \log(n)$  mal tauscht.

# Aufwandsklassen



## Fallunterscheidung: Aufwandsklassen

- $O(n)$  Obere Schranke, die der Algorithmus erreichen, aber nicht überschreiten kann
- $\Omega(n)$  Untere Schranke und ein “Mindestaufwand“, den der Algorithmus hat
- $\Theta(n)$  Vereinigung der Betrachtung aus  $\Omega(n)$  und  $O(n)$ . Es entsteht eine Art Funktionsbereich, den der Algorithmus nie verlässt

# Aufwandsklassen



Oberer asymptotische Schranke

$$O(g(n)) = \{f(n) \mid$$

$$\exists c \in \mathbb{R}, n_0 \in \mathbb{N} \forall n > n_0 : 0 \leq f(n) \leq cg(n)\}$$

Untere asymptotische Schranke

$$\Omega(g(n)) = \{f(n) \mid$$

$$\exists c \in \mathbb{R}, n_0 \in \mathbb{N} \forall n > n_0 : 0 \leq cg(n) \leq f(n)\}$$

Asymptotisch scharfe Schranke

$$\Theta(g(n)) = \{f(n) \mid$$

$$\exists c_1, c_2 \in \mathbb{R}, n_0 \in \mathbb{N} \forall n > n_0 : 0 \leq c_1g(n) \leq f(n) \leq c_2g(n)\}$$

# Aufgaben



- Welche Funktionen gehören in welche Klassen?

	$O(n^2)$	$\Omega(n^2)$	$O(\log n)$	$\Theta(n)$
$n^2 + n$				
$n \cdot \log(n)$				
$2 \cdot n + 1$				
$n^3$				
5				

# Aufgaben



- Welche Funktionen gehören in welche Klassen?

	$O(n^2)$	$\Omega(n^2)$	$O(\log n)$	$\Theta(n)$
$n^2 + n$	j	j	n	n
$n \cdot \log(n)$	j	n	n	n
$2 \cdot n + 1$	j	n	n	j
$n^3$	n	j	n	n
5	j	n	j	n

# Rekurrenzen



## Was sind Rekurrenzen?

Eine Rekurrenz  $T(n)$  ist die Aufwandsfunktion zu einem **bestimmten** Algorithmus oder Code in Abhängigkeit von der Anzahl bzw. Komplexität der Eingabemenge  $n$ .

Anders als beim O-Kalkül wird eine (mehr oder weniger) exakte Aussage über die Anzahl der ausgeführten Codezeilen getroffen und die Aufwandsfunktion  $T(n)$  definiert sich rekursiv.

Sie ist stets eine Worst-Case-Betrachtung.

## Beispiel

$T(n) = T(n - 1) + 1$  ist die Rekurrenz einer linearen Suche, bei der pro Schleifendurchlauf 1 Vergleich gemacht wird.

# Rekurrenzen



Da man aber wenig über eine Rekurrenz der Art  $T(n) = T(n/4 + 1) + 2^{n/2} + 6$  aussagen kann, wollen wir die Rekursion entfernen.

Dazu gibt es 3 Möglichkeiten:

- Raten und beweisen (nur für einfache Rekurrenzen)
- Master-Theorem (nur für einige Sonderfälle anwendbar)
- erzeugende Funktionen

Beispiel:  $T(n) = T(n - 1) + 1$

# Rekurrenzen



Da man aber wenig über eine Rekurrenz der Art  $T(n) = T(n/4 + 1) + 2^{n/2} + 6$  aussagen kann, wollen wir die Rekursion entfernen.

Dazu gibt es 3 Möglichkeiten:

- Raten und beweisen (nur für einfache Rekurrenzen)
- Master-Theorem (nur für einige Sonderfälle anwendbar)
- erzeugende Funktionen

Beispiel:  $T(n) = T(n - 1) + 1$

- Vermutung:  $T(n) = n + 1$
- Induktionsanfang: für  $n = 1$  gilt:  $T(n = 1) = 2 = T(1)$
- Induktionsannahme: für  $n$  gelte, dass  $T(n) = n + 1$
- Induktionsschritt: dann gilt für  $n + 1$ , dass 
$$T(n + 1) = 1 + T(n + 1 - 1) = 1 + T(n) = 1 + n + 1 = n + 2$$

- 1 Organisatorisches
- 2 Und nochmal...
- 3 Übungsblatt 04
- 4 Aufwandsanalyse
- 5 Mastertheorem**
- 6 Programmierkonzepte

# Mastertheorem



## Die drei Fälle

Eine Rekurrenz  $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$  lässt sich durch die Verwendung von allgemeinen Regeln lösen:

- 1 Es gilt  $f(n) = O(n^{\log_b a - \epsilon})$  für ein  $\epsilon > 0$ , dann ist  
$$T(n) = \Theta(n^{\log_b a})$$
- 2 Es gilt  $f(n) = \Theta(n^{\log_b a})$ , dann ist  
$$T(n) = \Theta(n^{\log_b a} \cdot \log_2 n)$$
- 3 Es gilt  $f(n) = \Omega(n^{\log_b a + \epsilon})$  für ein  $\epsilon > 0$ , ausserdem gilt für ein  $c < 1$  und genügend große  $n$   $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ , dann ist  
$$T(n) = \Theta(f(n))$$

# Mastertheorem-Aufgabe



- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n$
- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^2$
- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^3$

# Mastertheorem-Aufgabe



- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n$

Somit gilt  $a = 4$ ,  $b = 2$ ,  $f(n) = n$  und  $n^{\log_b a} = n^{\log_2 4} = n^2$

Da  $f(n) = O(n^{(\log_2 4) - \epsilon})$  mit z.B.  $\epsilon = 1$  gilt Fall 1:

Daher gilt:  $T(n) = \Theta(n^2)$

- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^2$

- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^3$

# Mastertheorem-Aufgabe



- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n$

Somit gilt  $a = 4$ ,  $b = 2$ ,  $f(n) = n$  und  $n^{\log_b a} = n^{\log_2 4} = n^2$

Da  $f(n) = O(n^{(\log_2 4) - \epsilon})$  mit z.B.  $\epsilon = 1$  gilt Fall 1:

Daher gilt:  $T(n) = \Theta(n^2)$

- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^2$

Somit gilt  $a = 4$ ,  $b = 2$ ,  $f(n) = n^2$  und  $n^{\log_b a} = n^{\log_2 4} = n^2$

Da  $f(n) = \Theta(n^{\log_2 4})$  gilt Fall 2:

Daher gilt:  $T(n) = \Theta(n^2 \cdot \log n)$

- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^3$

# Mastertheorem-Aufgabe



- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n$   
Somit gilt  $a = 4$ ,  $b = 2$ ,  $f(n) = n$  und  $n^{\log_b a} = n^{\log_2 4} = n^2$   
Da  $f(n) = O(n^{(\log_2 4) - \epsilon})$  mit z.B.  $\epsilon = 1$  gilt Fall 1:  
Daher gilt:  $T(n) = \Theta(n^2)$
- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^2$   
Somit gilt  $a = 4$ ,  $b = 2$ ,  $f(n) = n^2$  und  $n^{\log_b a} = n^{\log_2 4} = n^2$   
Da  $f(n) = \Theta(n^{\log_2 4})$  gilt Fall 2:  
Daher gilt:  $T(n) = \Theta(n^2 \cdot \log n)$
- $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^3$   
Somit gilt  $a = 4$ ,  $b = 2$ ,  $f(n) = n^3$  und  $n^{\log_b a} = n^{\log_2 4} = n^2$   
Da  $f(n) = \Omega(n^{(\log_2 4) + \epsilon})$  mit z.B.  $\epsilon = 1$  und  
 $4 \cdot f\left(\frac{n}{2}\right) = 4 \cdot \frac{n^3}{8} \leq c \cdot n^3$  mit z.B.  $c = \frac{1}{2}$  gilt Fall 3:  
Daher gilt:  $T(n) = \Theta(n^3)$

- 1 Organisatorisches
- 2 Und nochmal...
- 3 Übungsblatt 04
- 4 Aufwandsanalyse
- 5 Mastertheorem
- 6 Programmierkonzepte**

# Divide and Conquer



## Divide und was?

- „Divide and Conquer“ (zu Deutsch „Teile und Herrsche“) ist ein Programmierkonzept.
- Es wird verwendet, um komplexe Probleme effizient zu lösen.

## Probleme Lösen mit Divide & Conquer

- Das zu lösende Problem wird in mehrere einfachere Teilprobleme zerteilt.
- Diese Teilprobleme werden, wenn sie einfach genug sind gelöst. (sonst weiter zerteilen)
- Die Lösungen der Teilprobleme werden anschließend zur Lösung des Ausgangsproblems zusammengefügt.

# Greedy-Algorithmen



## Greedy!?!

- „Greedy“-Algorithmen (zu deutsch „Gierig“-Algorithmen) lösen Optimierungsprobleme.
- Der Algorithmus wählt dazu anhand einer gegebenen Bewertungsfunktion in jedem Schritt das „größte“ bzw. beste Element aus.

# Greedy-Algorithmen



## Probleme lösen mit Greedy

- Ein Greedy-Algorithmus wählt aus einer gegebenen Menge an Möglichkeiten zu jedem Zeitpunkt immer das „beste“ bzw. „größte“.
- Dieses Vorgehen führt schnell zu einer Lösung, diese muss jedoch nicht optimal sein.
- Ein Bekanntes mittels Greedy-Algorithmus zu lösendes Problem ist zum Beispiel das Rucksack-Problem.

Ende

