Grundlegende Algorithmen

WS 2004/2005

Jens Ernst

Lehrstuhl für Effiziente Algorithmen

Institut für Informatik

Allgemeines zur Vorlesung:

- Bereich: 3+2 SWS Vorlesung im Bachelorstudium Informatik (3. Sem)

  3+2 SWS Vorlesung im Aufbaustudium Informatik (1. Sem)

  3+2 SWS Vorlesung im Masterstudium Angewandte Informatik (1. Sem)

  3+2 SWS Vorlesung im Masterstudium Computational Sci. & Eng. (1. Sem)

  Diesmal auch 3+2 SWS im Bachelor-studium Wirtschaftsinformatik (5. Sem)

Allgemeines zur Vorlesung:

- Dozent: Dr. Jens Ernst, Zimmer 03.13.061

  email: ernstj@in.tum.de

  Tel. 289 – 19426

  Sprechstunden: n.V.

- Vorlesung: Do 12:15 - 13:00, HS MI 00.06.011

  Fr 08:30 - 10:00, HS MI 00.04.011

- Übung: Do 16:00 – 17:30, HS MI 00.06.011

- Übungsleiter: Johannes Nowak, Zimmer 03.09.043
email: nowakj@in.tum.de
Tel: 289-17736
- Anmeldung zu Vorlesung/Übungsbetrieb:
  online, auf der Webseite zur Übung
- Semestralschein: Erforderlich ist die erfolgreiche Teilnahme an Midterm- und Endklausur.
  Zur Zulassung für die Endklausur ist das Erreichen von 40% der Hausaufgabenpunkte notwendig.

- Klausurtermine (voraussichtlich !!):

Midterm:	Fr., 10.12.04
Final:	Mi., 16.02.05
Wiederholung:	??

- Gewichtung der Klausuren: Die Punkte für den Semestralschein berechnen sich wie folgt:

$$\left(\frac{2}{5}\text{Punkte in Midterm} + \frac{3}{5}\text{Punkte in Endklausur}\right)$$

- Hinweis: Bitte alle Probleme bzgl. Klausurzulassung, Hausaufgabenpunkte etc. *rechtzeitig* mit der Übungsleitung klären

- Inhalt der Vorlesung:
  1. Einleitung, Grundlagen und Notation
  2. Algorithmenentwicklung durch Induktion
  3. Sortier- und Selektionsalgorithmen
  4. Datenstrukturen und Suchoperationen
  5. Graphalgorithmen
  6. Textalgorithmen
  7. Algebraische und numerische Probleme
  8. Datenkompression

  ... Wünsche bitte per email mitteilen!

Empfohlene Literatur:

- **Volker Heun, "*Grundlegende Algorithmen*"**
  Vieweg-Verlag Braunschweig-Wiesbaden,
  2. Auflage, 2003
- Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein, "*Introduction to Algorithms*"
  MIT Press, Cambridge MA, 2. Auflage, 2001
- Robert Sedgewick, "*Algorithmen*"
  Pearson Education, München 2002
- Uwe Schöning, "*Algorithmik*"
  Spektrum-Verlag Heidelberg, 2001

1. Einleitung:

Definition (*Algorithmus*): Ein *Algorithmus* ist ein eindeutig spezifiziertes Verfahren zur Berechnung von gesuchten Ausgabegrößen aus gegebenen Eingabegrößen.
Wir betrachten *hier* Algorithmen mit folgenden Eigenschaften:

- *sequenziell*: Zu jedem Zeitpunkt wird nur genau ein Schritt ausgeführt.

  Anm: Anders bei *parallelen* und *verteilten* Algorithmen.

- *deterministisch*: Zu jedem Ausführungszeitpunkt ist der nächste Schritt eindeutig bestimmt.

Anm 1: In der Komplexitätstheorie betrachtet man auch *nichtdeterministische* Algorithmen, bei denen ein Schritt auch mehrere Folgeschritte haben kann.

Anm 2: *Randomisierte* Algorithmen können gewisse Schritte von Zufallsereignissen (vgl. Münzwurf) abhängig machen.

- *determiniert*: Wird der Algorithmus mehrfach mit derselben Eingabe ausgeführt, so ergibt sich stets dieselbe Ausgabe.

  Anm: Nicht so bei randomisierten Algorithmen.

- *statische Finitheit*: Die Beschreibung des Algorithmus benötigt nur endlich viel Platz.

- *dynamische Finitheit*: Zu jedem Zeitpunkt, an dem man den Algorithmus unterbricht, belegt er nur endlich viel Speicherplatz.
- *Terminierung*: Die Ausführung des Algorithmus endet nach endlich vielen Schritten.

  Anm: Anders u.U. bei *Online-Algorithmen*, die zu Beginn ihrer Ausführung nicht die gesamte Eingabe kennen.

Beispiele für algorithmische Probleme:

- Datenorganisation und effiziente Suche in einer Web-Suchmaschine
- Datenorganisation und –Zugriff in einer Datenbank
- Zusammensetzung der Sequenz des menschlichen Genoms aus hunderttausenden von Fragmenten
- Berechnung eines VLSI-Layouts
- Kompression einer Audio- oder Videodatei
- Schnelle Ver- und Entschlüsselung zur Übertragung geheimer Daten

etc...

Algorithmen und Effizienz:

Die Effizienz eines Algorithmus wird meist beurteilt im Hinblick auf *Laufzeit* und (*Speicher-*)*platzbedarf.* Beide werden in Abhängigkeit von der Eingabegröße *n* (in bits) angegeben.

Die Laufzeit wird zumeist als *Anzahl* der ausgeführten Operationen (z.B. Additionen, Vergleiche) angegeben.

Beispiel: Für eine gegebene Maschine koste ein (Rechen-) Schritt 1 Mikrosekunde. Wir betrachten unterschiedlich effiziente Algorithmen für ein gegebenes Problem.

(Forts.)

Wir geben für verschiedene Eingabegrößen n die Laufzeit $T(n)$ (in Sekunden, Stunden etc.) für Algorithmen an, deren Zeitbedarf $t(n)=1000n, 1000n \cdot log(n), 100n^2$ sowie 2^n beträgt.

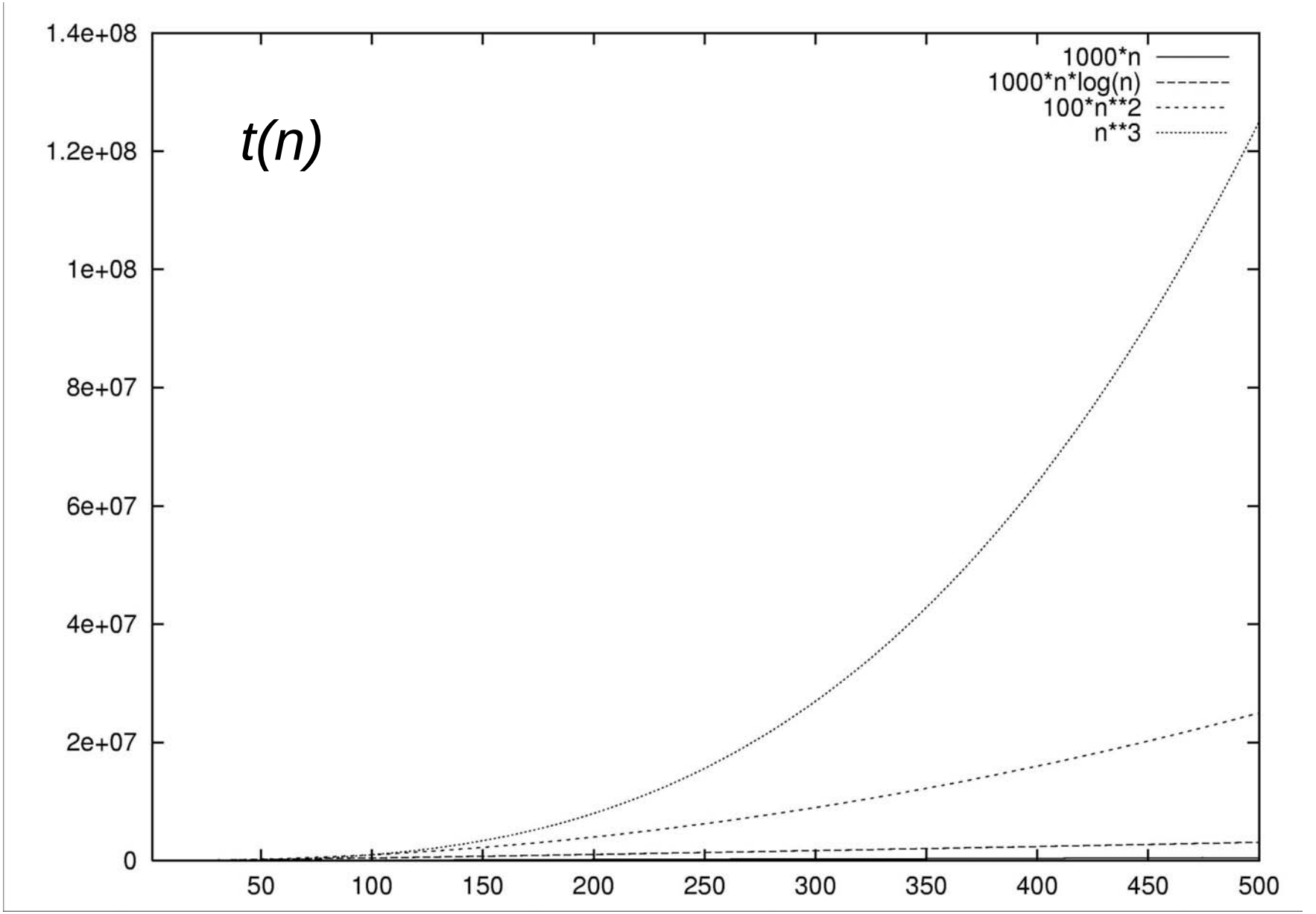
	20	**50**	**100**	**200**	**500**	**1000**	**10000**
1000 n	0.02s	0.05s	0.1s	0.2s	0.5s	1s	10s
1000 n log n	0.09s	0.3s	0.6s	1.5s	4.5s	10s	2 min
100 n^2	0.04s	0.25s	1s	4s	25s	2 min	2.8 h
10 n^3	0.02s	1s	10s	1 min	21 min	2.7 h	116 t
2^n	1s	35 J	3×10^4 Jh				

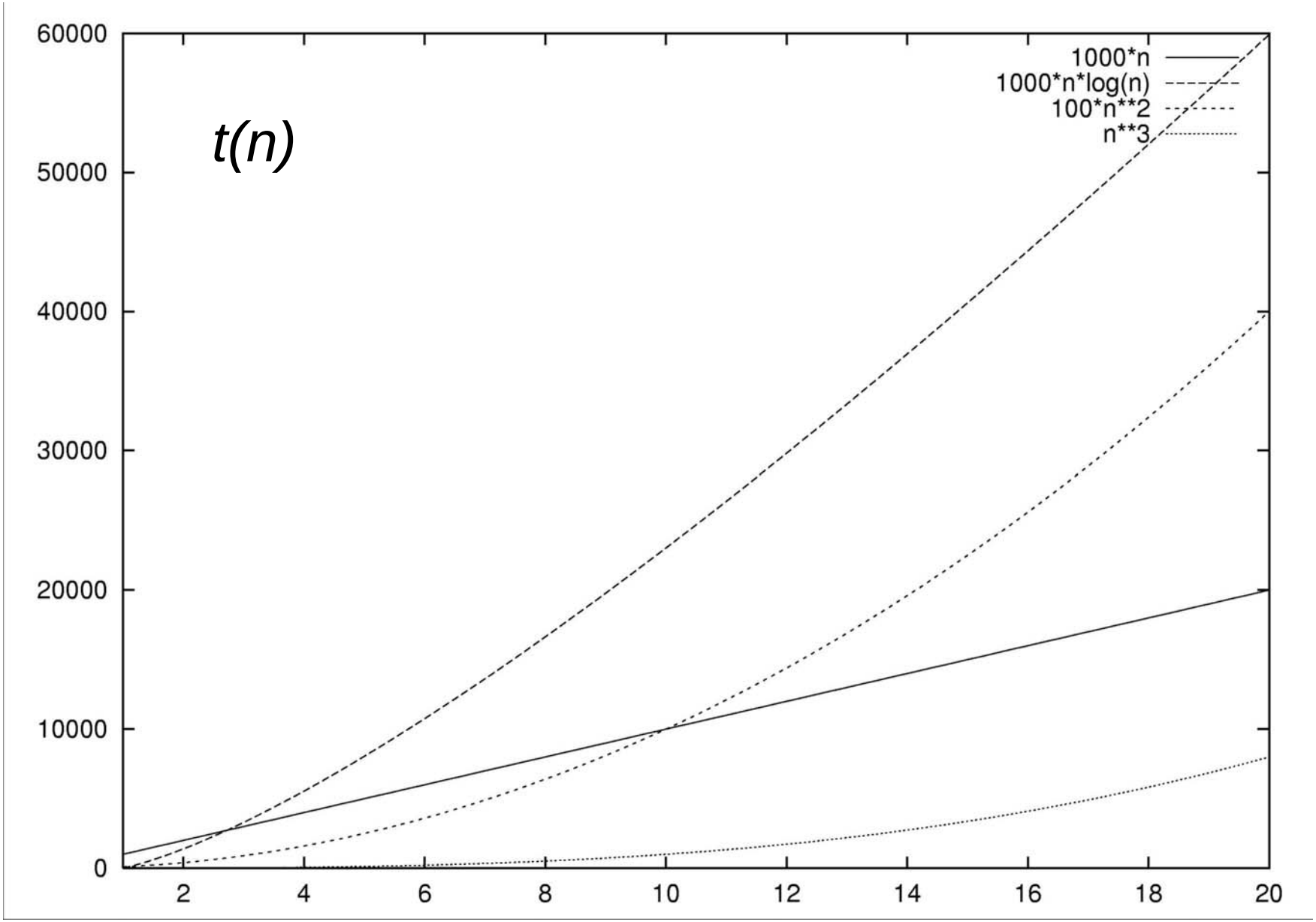
Man erkennt: Die *Komplexität* eines Algorithmus (Zeitbedarf in Abhängigkeit von der Eingabegröße) ist bei hinreichend großen Eingaben entscheidend für ihre praktische Brauchbarkeit.

Was nützen schnellere Rechner ?

Das einzige, was sich durch den Einsatz einer schnelleren Maschine an der Laufzeit ändert, ist ein *konstanter Faktor* (und auch der kann nahe 1.0 liegen). Das *asymptotische* Laufzeitverhalten aber wird dadurch nicht verbessert.

Es lohnt sich also, bei der Algorithmenentwicklung auf Effizienz zu achten.


t(n)
1000*n
1000*n*log(n)
100*n**2
n**3
1.4e+08
1.2e+08
1e+08
8e+07
6e+07
4e+07
2e+07
0
50
100
150
200
250
300
350
400
450
500


t(n)
1000*n
1000*n*log(n)
100*n**2
n**3
60000
50000
40000
30000
20000
10000
0
2
4
6
8
10
12
14
16
18
20

Bis zu welcher Eingabegröße kann man arbeiten?

Angenommen, wir verwenden eine Maschine, die f Schritte pro Sekunde ausführen kann (i. Bsp: $f = 10^6$). Der Algorithmus führee $t(n)$ Schritte für Eingaben der Größe n aus. Dann gilt für die gemessene Laufzeit $T(n)$:

$$T(n)=t(n)/f \quad [\text{sec}].$$

Soll eine Berechnung innerhalb einer Zeit von s sec. abgeschlossen sein, so kann man nur Eingaben einer Größe von

$$n \leq t^{-1}(s \cdot f)$$

verarbeiten, wobei wir annehmen, daß $t(n)$ eine streng monoton wachsende Funktion ist.

Anm: Hier sieht man auch den Effekt einer Vergrößerung der Frequenz ***f***, wie sie sich z.B. durch den Einsatz einer schnelleren Maschine ergibt.

Bsp: Bei einer Komplexität von $t(n)=n^2$ kann man bei vorgegebener Laufzeit s die Eingabe nur um einen **Faktor** von **1.414 (bzw 31.62)** vergrößern, wenn man einen "**doppelt** (bzw. **tausend**) mal so schnellen" Rechner verwendet.

Bei $t(n)=2^n$ darf n dann nur um den Wert ***log*(2)=1 (bzw. $\lfloor$log(1000)$\rfloor$=9)** wachsen!
D.h. bei einem Algorithmus von exponentieller Komplexität nützt ein schnellerer Rechner so gut wie nichts.

Ziele der Vorlesung:

- Einführung in die Terminologie der Algorithmik
- Formalisierung algorithmischer Fragestellungen
- Grundlegende Techniken der Algorithmenentwicklung
- Algorithmen für Standardprobleme (konstruktiv)
- Grundlegende Methoden der Analyse von Algorithmen
- Primitive und höhere Datenstrukturen
- Übungen zu allen oben genannten Punkten
- Implementierung der besprochenen Verfahren

Implementierung:

Neben der Effizienz der von uns entwickelten Algorithmen wollen wir uns auch um eine effiziente Implementierung bemühen.

Daher: *Programmierung* der besprochenen Algorithmen dringend empfohlen.

Programmiersprache: am besten C(++)

Gründe:

- Effizienz, Optimierbarkeit
- Maschinen- und Betriebssystemnähe
- Grundwissen eines Informatikers