FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Algorithmen und Datenstrukturen

Kapitel 4: Analyse von Algorithmen - Teil I

Prof. Ingrid Scholl

FH Aachen - FB 5

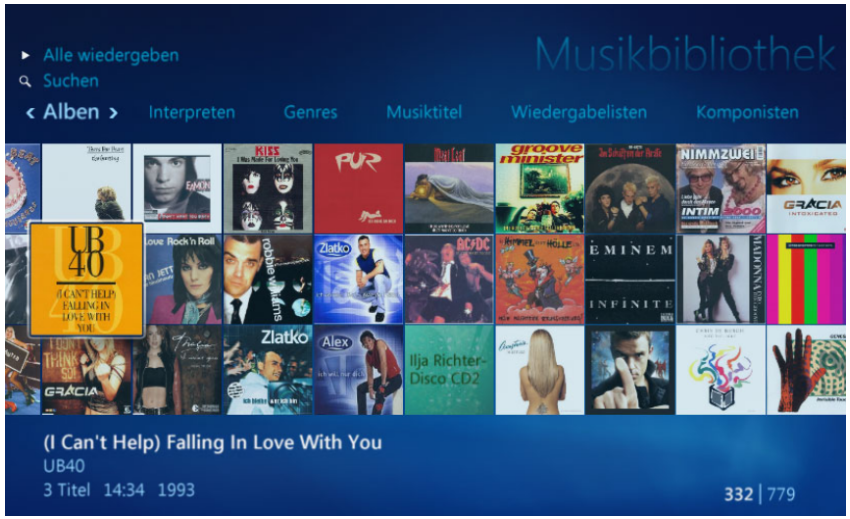
scholl@fh-aachen.de

21.04.2020

Analyse von Algorithmen

1. Wissenschaftliche Methode
2. Beobachtungen
3. Analyse der Messdaten
4. Mathematische Modelle
5. Wachstumsordnungen
6. Klassifikation der Wachstumsordnungen

Beispiele


Alle wiedergeben
Suchen
Musikbibliothek
Alben
Interpreten
Genres
Musiktitel
Wiedergabelisten
Komponisten
(I Can't Help) Falling In Love With You
UB40
3 Titel 14:34 1993
332 | 779


Moments
Select
St Marys Rd
Jan 16
Share
Collections
Moments
Select

Beispiele


facebook


iPhone 4
15.33
Title
Search
A
#3 Owned, for Keeps
Alice in Wonderland
2010 Adventure, Family, Fantasy
#5 Owned, for Keeps
Avatar
2009 Fantasy
B
#2 Owned, for Keeps
The Book of Eli
2010 Action, Adventure, Drama, Suspens...
#8 Owned, for Keeps
The Bounty Hunter
2010 Action, Comedy, Crime, Romance
Collection
Persons
Releases
Trailers
Settings
CLASH OF THE TITANS
Clash of the Titans
My Movies
for iPhone

Analyse von Algorithmen

Ziel: Umstrukturieren von komplexen Daten (Musik-, Foto-, Video-Bibliotheken, Big Data Visualization)

Gegeben: Algorithmus, der ein gegebenes Problem löst.

Fragen:

1. Wie lange benötigt mein Programm?
   → Laufzeitanalyse
2. Wann bzw. warum geht meinem Programm der Speicher aus?
   → Speicherverbrauch messen

Analyse von Algorithmen

Wovon hängen Laufzeit und Speicherverbrauch ab?

- Eigenschaften des verwendeten Rechners
- von den zu verarbeiteten Daten
- vom verwendeten Algorithmus

Ziel: Suche eine wissenschaftliche Methode zur Bewertung der Kosten.

Wissenschaftliche Methode

1. Beobachten des Laufzeitverhaltens / Speicherverbrauchs durch Messungen.
2. Erstellen eines hypothetischen Modells, das möglichst mit den Beobachtungen überein stimmt.
3. Vorhersage des Verhaltens durch Anwendung des hypothetischen Modells.
4. Verifikation der Vorhersage durch weitere Messungen.
5. Wiederholung der Schritte 1-4 bis die Hypothese mit den Beobachtungen überein stimmt.

Wissenschaftliche Methode

Albert Einstein:

"*Keine noch so große Zahl von Experimenten kann beweisen, dass ich recht habe. Aber es reicht ein einziges Experiment, um zu beweisen, dass ich unrecht habe.*"

- Wir können nie sicher sein, dass unsere Hypothese absolut korrekt ist.
- Man kann nur sagen, dass sie mit unseren Beobachtungen übereinstimmen.

Beobachtungen

- Programm-Laufzeiten quantitativ messen.
- Programm verarbeitet eine Problemgröße (z.Bsp. N Eingabedaten).
- Variiere die Problemgröße und messe die Laufzeiten.
- Beobachte das Laufzeitverhalten bei zunehmender Problemgröße.
- Stelle eine Hypothese auf und beobachte durch weitere Experimente, ob diese bestätigt werden kann.

Laufzeitmessung

Problemgröße ~ Komplexität der Programmieraufgabe

Problemgröße wird beeinflusst durch:

- Größe der Eingabedaten, z.Bsp. Anzahl N der zu sortierenden Daten
- Wert eines Eingabeparameters

Ziel

Quantifiziere die Beziehung zwischen Problemgröße und Laufzeit und approximiere diese durch eine Komplexitätsfunktion.

Beispiel: Laufzeitanalyse

Messdaten zur Funktion count von ThreeSum.cpp:

Laufzeit-Messdaten

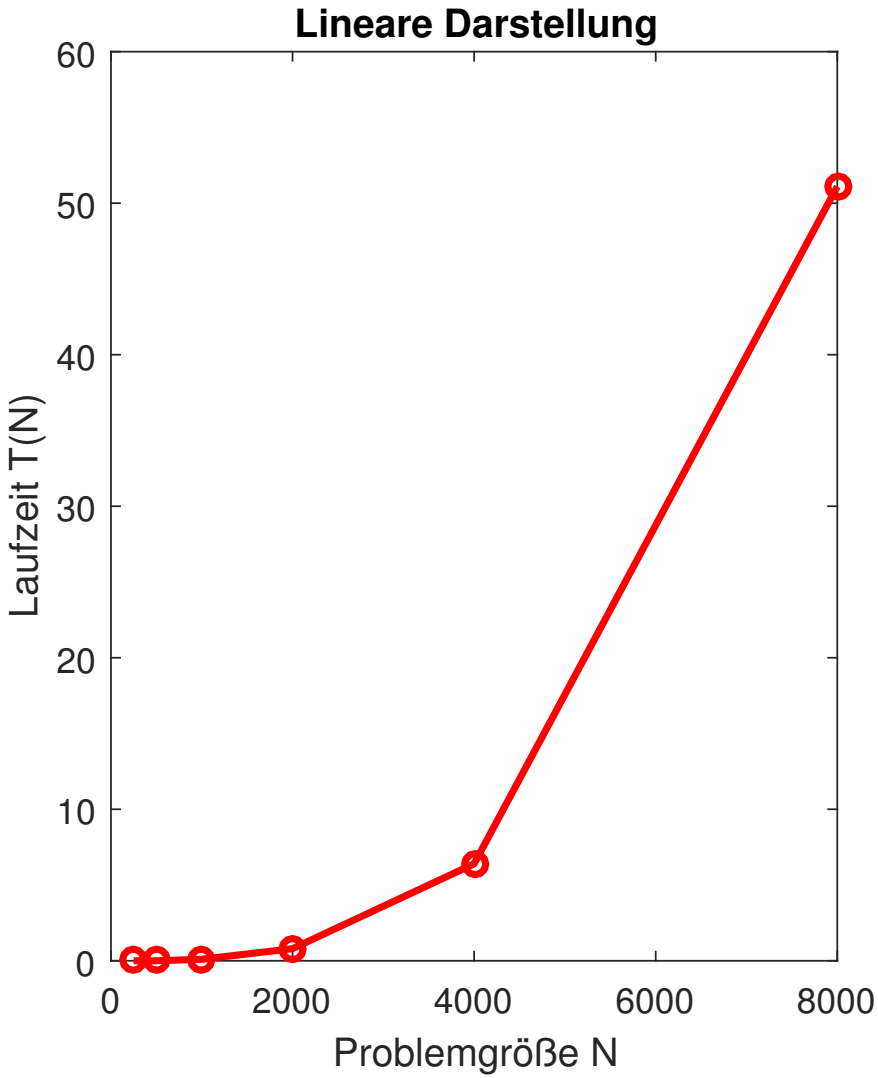
N	T(N)
250	0.0
500	0.0
1000	0.1
2000	0.8
4000	6.4
8000	51.1

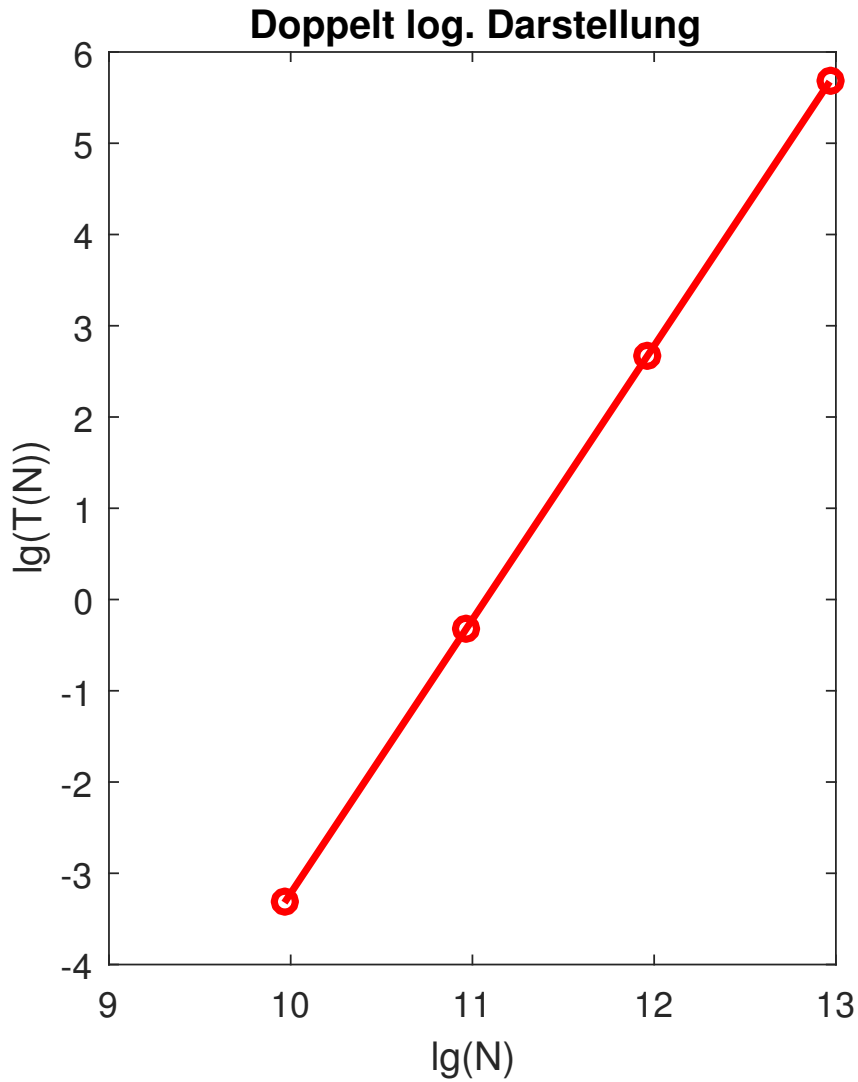
```
int count(vector<int> &a)
{
   // Zählt die Tripel, die sich zu 0
   // aufsummieren

   int N = (int) a.size();
   int cnt = 0;
   for(int i=0; i < N; i++)
      for(int j = i+1; j < N; j++)
         for (int k = j+1; k < N; k++)
            if (a[i]+a[j]+a[k] == 0) cnt++;
   return cnt;
}
```

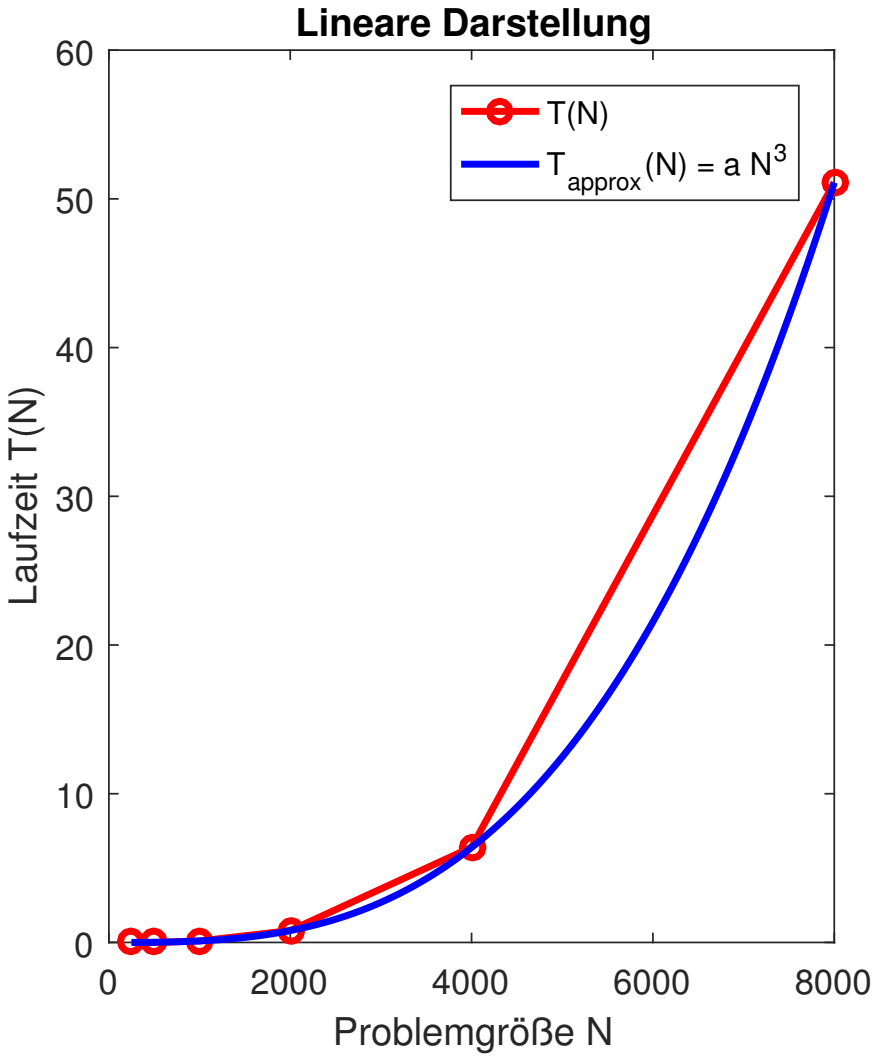
Eingabedateien:
1Kints.txt, 2Kints.txt, 4Kints.txt, 8Kints.txt, ...

Beispiel: Laufzeitanalyse



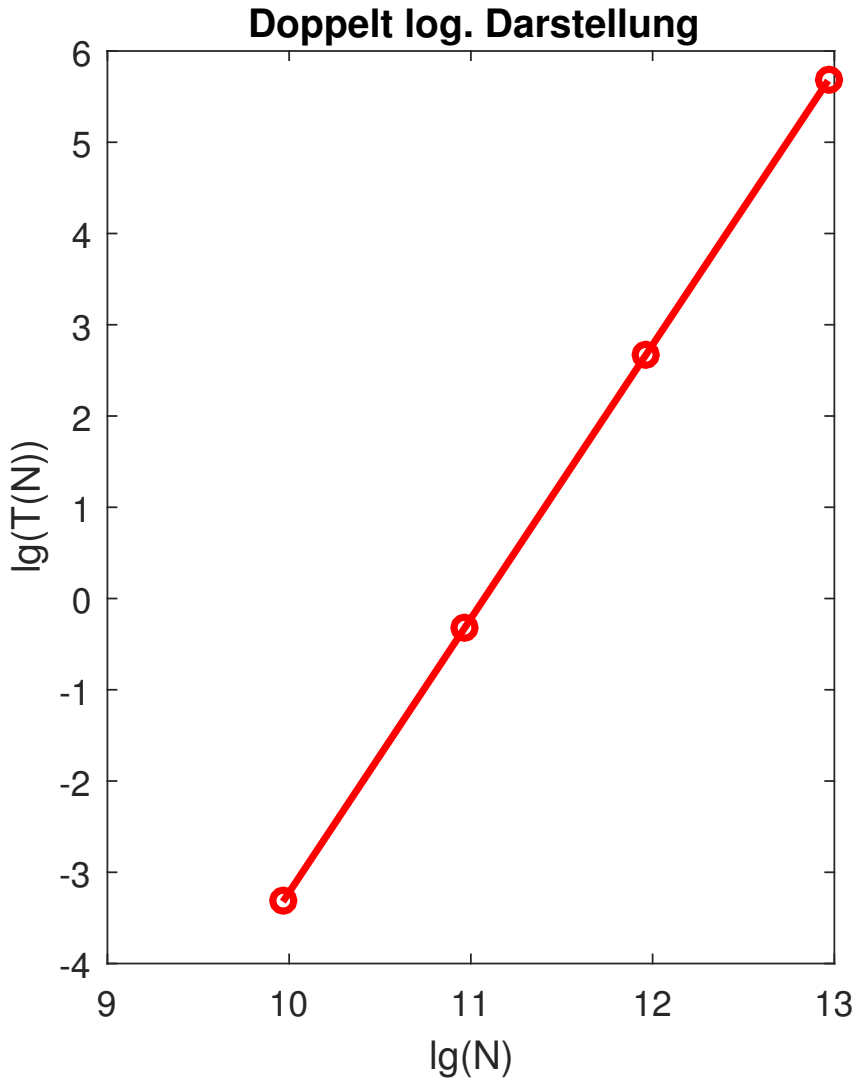


Beispiel: Laufzeitanalyse



$$T(N) \sim a \cdot N^3$$
$$a = 9.98 \cdot 10^{-11}$$

Beispiel: Laufzeitanalyse



$$
\begin{aligned}
lg(T(N)) &= 3lg(N) + lg(a) \\
&= lg(N^3) + lg(a) \\
&= lg(a \cdot N^3)
\end{aligned}
$$

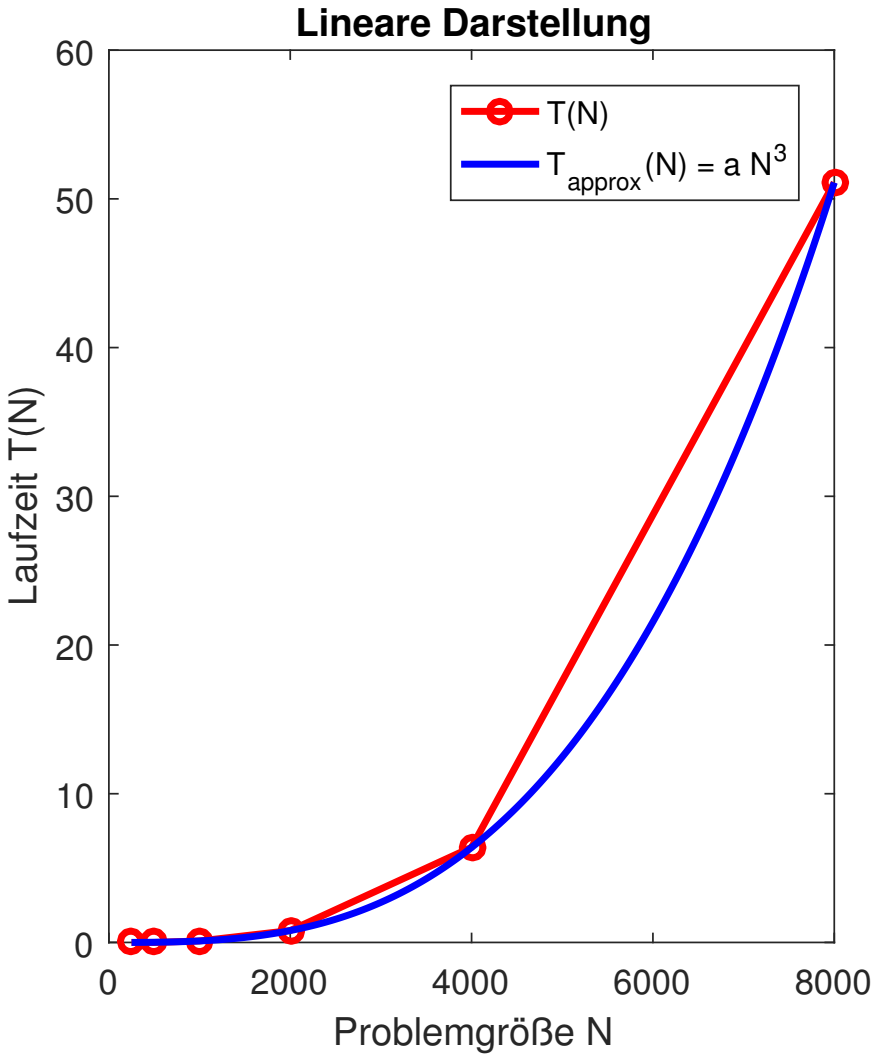
$$T(n) = a \cdot N^3$$

$$
\begin{aligned}
T(8000) &= 51.5 = a \cdot 8000^3 \\
\rightarrow a &= 9.98 \cdot 10^{-11}
\end{aligned}
$$

Hypothese

$$T(N) = 9.98 \cdot 10^{-11} \cdot N^3$$

Beispiel: Laufzeitanalyse

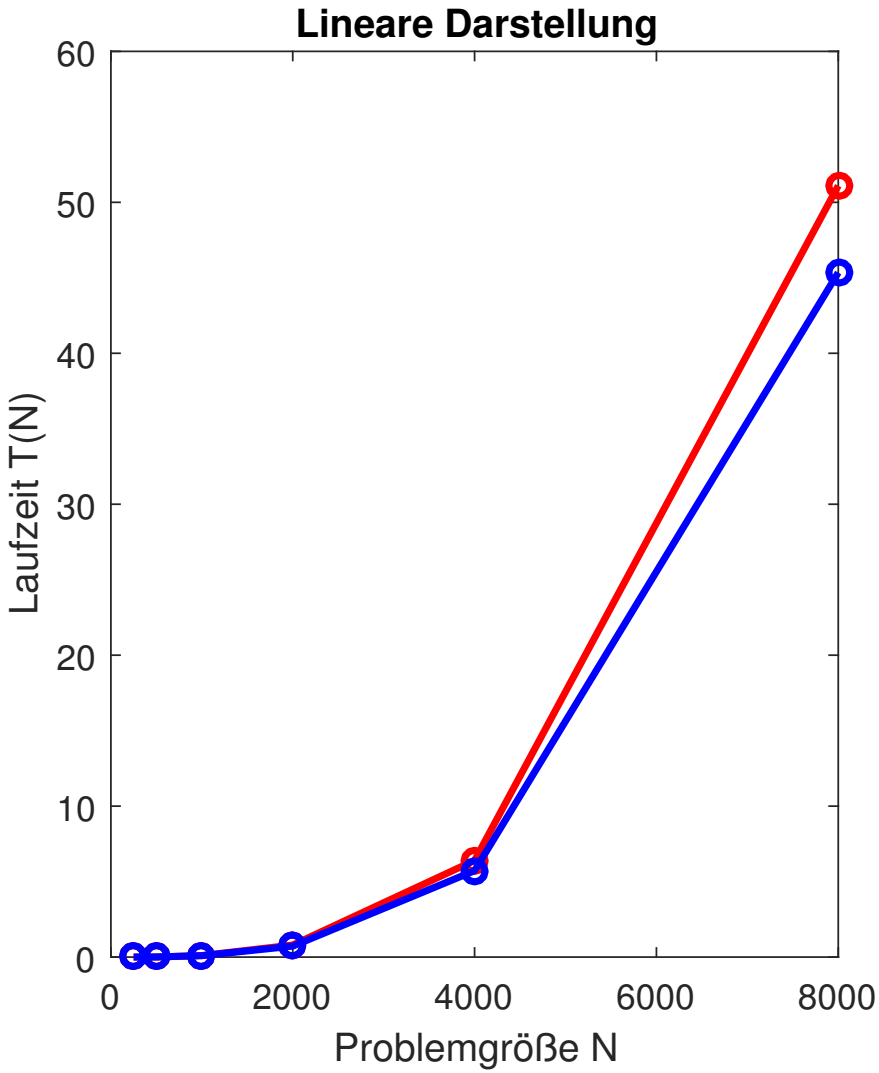


$$T(N) \sim a \cdot N^3$$
$$a = 9.98 \cdot 10^{-11}$$

$$\begin{aligned} T(16000) &= a \cdot 16000^3 \\ &= 409.8\,sec \\ &= 6.8\,min \end{aligned}$$

Reale Laufzeit für N=16000 waren 409.3 sec!

Beispiel: Vergleich Laufzeitmessung



N	sec [C1]	sec [C2]
250	0.0	0.002
500	0.0	0.012
1000	0.1	0.093
2000	0.8	0.724
4000	6.4	5.720
8000	51.1	45.346

Beobachtung

Laufzeiten auf verschiedenen Rechnern unterscheiden sich nur durch einen konstanten Faktor!

Mathematische Modelle

Ziel: Genaue Vorhersage der Laufzeit eines Programmes.
Gesamtausführungszeit wird bestimmt durch:

Hauptfaktoren für die Laufzeit

- Kosten für die Ausführung der einzelnen Anweisungen (Systemeigenschaften, Compiler, Betriebssystem)
- Häufigkeit der Ausführung der einzelnen Anweisungen (Programm selbst und Eingabe)

Mathematische Modelle

```
int count(vector<int> &a)
{
   // Zählt die Tripel, die sich zu 0
   // aufsummieren

   int N = (int) a.size();
   int cnt = 0;
   for(int i=0; i < N; i++)
      for(int j = i+1; j < N; j++)
         for (int k = j+1; k < N; k++)
            if (a[i]+a[j]+a[k] == 0) cnt++;
   return cnt;
}
```

$$T(N) = \sum_{i=0}^{N-1} \sum_{j=i-1}^{N-1} \sum_{k=j+1}^{N-1} 1$$

Mathematische Modelle

```
int count(vector<int> &a)
{
   // Zählt die Tripel, die sich zu 0
   // aufsummieren

   int N = (int) a.size();
   int cnt = 0;
   for(int i=0; i < N; i++)
      for(int j = i+1; j < N; j++)
         for (int k = j+1; k < N; k++)
            if (a[i]+a[j]+a[k] == 0) cnt++;
   return cnt;
}
```

Anzahl der Möglichkeiten, 3 verschiedene Zahlen aus dem Eingabearray zu wählen ist $\frac{N \cdot (N-1) \cdot (N-2)}{6}$

Tilde-Approximation

$$T(N) = \frac{N(N-1)(N-2)}{6} = \frac{N^3}{6} - \frac{N^2}{2} + \frac{N}{3}$$

N	T(N) (sec)	$N^3/6$	$-N^2/6 + N/3$
250	0.0	$0.0003 \cdot 10^{10}$	$-0.0031 \cdot 10^7$
500	0.0	$0.0021 \cdot 10^{10}$	$-0.0125 \cdot 10^7$
1000	0.1	$0.0167 \cdot 10^{10}$	$-0.0500 \cdot 10^7$
2000	0.8	$0.1333 \cdot 10^{10}$	$-0.1999 \cdot 10^7$
4000	6.4	$1.0667 \cdot 10^{10}$	$-0.7999 \cdot 10^7$
8000	51.1	$8.5333 \cdot 10^{10}$	$-3.1997 \cdot 10^7$

Beobachtung

Leitterm (leading term) ist derjenige Term mit dem stärksten Wachstum. Der Leitterm bestimmt das stärkste Wachstum. Die anderen Terme hier sind dagegen eher unbedeutend.

Tilde-Approximation

$$T(N) = \frac{N(N-1)(N-2)}{6} = \frac{N^3}{6} - \frac{N^2}{2} + \frac{N}{3}$$

$$T(N) \sim \frac{N^3}{6}$$

Definition (Tilde-Notation (Sedgewick))

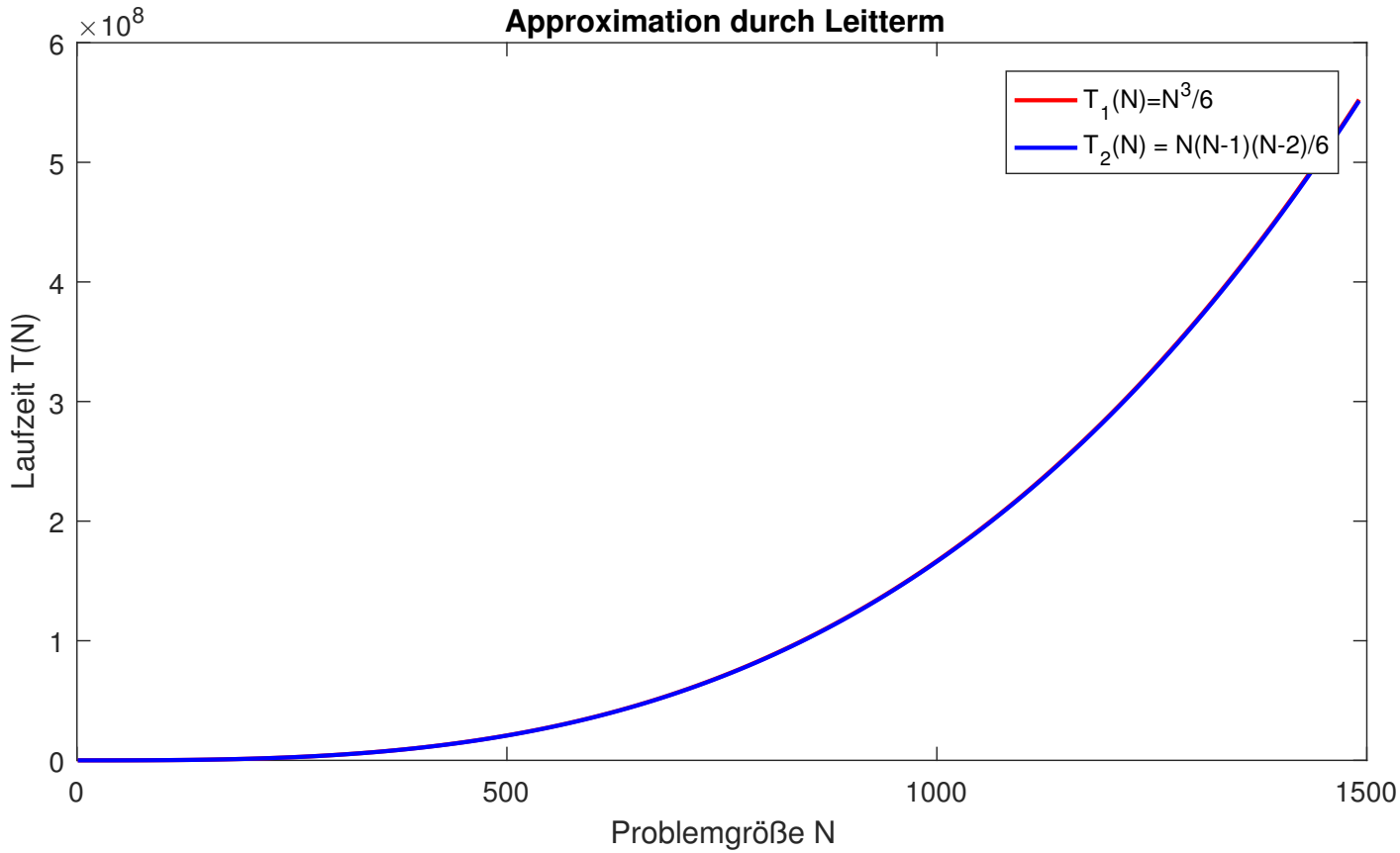
Wir schreiben $\sim f(N)$ zur Repräsentation einer beliebigen Funktion, die sich, wenn sie durch $f(N)$ geteilt wird, für zunehmende N dem Wert 1 nähert, und wir schreiben $g(N) \sim f(N)$ um anzuzeigen, dass

$$\lim_{N \to \infty} \frac{g(N)}{f(N)} = 1$$

für zunehmende N dem Wert 1 nähert.

Tilde-Approximation

$$T(N) = \frac{N^3}{6} - \frac{N^2}{2} + \frac{N}{3}; T(N) \sim \frac{N^3}{6}$$



Tilde-Approximation

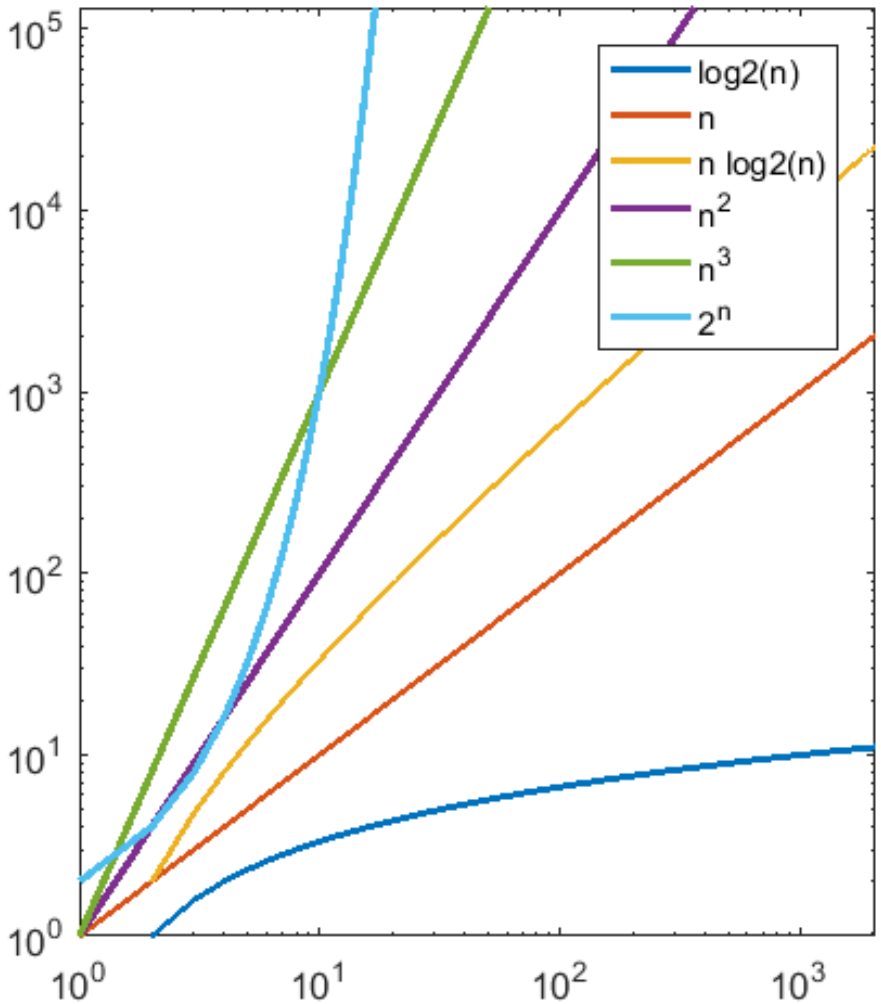
Typische Tilde-Approximationen

Funktion	Tilde-Approximation	Wachstums-Ordnung
$\frac{N^3}{6} - \frac{N^2}{2} + \frac{N}{3}$	$\sim \frac{N^3}{6}$	N^3
$\frac{N^2}{2} - \frac{N}{3}$	$\sim \frac{N^2}{2}$	N^2
$lgN + 1$	$\sim lgN$	lgN
3	~ 3	1

Wachstumsordnungen

Wachstumsordnungen

Beschreibung	Funktion
konstant	1
logarithmisch	$log\,N$
linear	N
überlinear	$N\,log\,N$
quadratisch	N^2
kubisch	N^3
polynomial	N^k
exponentiell	$2^N, k^N$



Analyse mit Tilde-Approximation

```
int count(vector<int> &a) {
A:    int N = (int) a.size();
A:    int cnt = 0;
A:    for(int i=0; i < N; i++)
B:       for(int j = i+1; j < N; j++)
C:          for (int k = j+1; k < N; k++)
D:             if (a[i]+a[j]+a[k] == 0) cnt++;
   return cnt;
}
```

Block	Häufigkeit	Tilde-Approximation
A	1	~ 1
B	N	$\sim N$
C	$N^2/2 + N/2$	$\sim N^2/2$
D	$N^3/6 - N^2/2 + N/3$	$\sim N^3/6$

Nützliche Approximationen für die Algorithmenanalyse

Approximationen

Beschreibung	Approximation
Harmonische Reihe	$H_N = 1 + 1/2 + 1/3 + \ldots + 1/N$ $\sim log_e N$
Dreieckszahlen	$1 + 2 + 3 + \ldots + N = N \cdot (N + 1)/2$ $\sim N^2/2$
Geometrische Reihe	$1 + 2 + 4 + 8 + \ldots + N = 2N - 1$ $\sim 2N$ mit $N = 2^n$
Stirlingformel	$log_2(N!) = log_2(1) + log_2(2) + \ldots + log_2(N)$ $\sim N \cdot log_2(N)$

Zusammenfassung häufigste Hypothesen zu Wachstumsfunktionen

Hypothesen zu Wachstumsfunktionen - Teil I

Beschreibung	**Wachstumsordnung**	**Typischer Coderahmen**	**Beschreibung**	**Beispiel**
Konstant	1	a = b + c	Anweisung	Addiert 2 Zahlen
Logarithmisch	*logN*	*siehe Listing*	Halbieren	Binäre Suche

Linear	*N*	double max = a[0];
		for (int i=1; i<N; i++)
		if (a[i] > max) max = a[i];
			Schleife	Maximum ermitteln

Zusammenfassung häufigste Hypothesen zu Wachstumsfunktionen

Hypothesen zu Wachstumsfunktionen - Teil II

Beschreibung	**Wachstumsordnung**	**Typischer Coderahmen**	**Beschreibung**	**Beispiel**
Leicht überlinear	$N \cdot logN$	Bsp. Merge Sort	Teile und Herrsche	Merge Sort

Quadratisch	N^2	for (int i=1; i<N; i++)
		for (int j=i+1; j<N; j++)
		if (a[i] + a[j] == 0)
		cnt++;
			Doppelte Schleife	Prüft alle Paare

Kubisch	N^3	for (int i=1; i<N; i++)
		for (int j=i+1; j<N; j++)
		for (int k=j+1; k<N; k++)
		if (a[i] + a[j] + a[k] == 0)
		cnt++;
			Dreifache Schleife	Prüft alle Tripel

Zusammenfassung häufigste Hypothesen zu d Wachstumsfunktionen

Hypothesen zu Wachstumsfunktionen - Teil III

Beschrei-bung	Wachs-tums-ordnung	Typischer Coderahmen	Beschrei-bung	Bei-spiel
Exponen-tiell	2^N	Bsp. Backtracking Globale Optimumsuche	Ausgiebige Suche	Prüft alle Teil-mengen

Analyse von Algorithmen

1. Wissenschaftliche Methode
2. Beobachtungen
3. Analyse der Messdaten
4. Mathematische Modelle
5. Wachstumsordnungen
6. Klassifikation der Wachstumsordnungen

Vielen Dank!

Prof. Ingrid Scholl
FH Aachen
Fachbereich für Elektrotechnik und Informationstechnik
Graphische Datenverarbeitung und Grundlagen der Informatik
MASKOR Institut
Eupener Straße 70
52066 Aachen
T +49 (0)241 6009-52177
F +49 (0)241 6009-52190
scholl@fh-aachen.de

www.fh-aachen.de