

Komplexität eines Algorithmus, Grössenordnung, Landau-Symbole, Beispiel einer Komplexitätsberechnung (Mergesort)

4. KOMPLEXITÄT

134

Komplexität eines Algorithmus

Algorithmen verbrauchen Ressourcen

- Rechenzeit
- Speicher
- (Energie)

Zunehmend wichtig. Aber zu kompliziert für diese Vorlesung

Wichtiges Ziel: Ressourcenverbrauch minimieren

- Beispiel einer Frage: Wie hoch ist der Ressourcenverbrauch beim Berechnen des Medians mit Mergesort?
- Präzisierte Frage: Im besten Fall, im schlechtesten Fall, im Durchschnitt?

Quelle: Vorlesung Informatik II D-ITET, Prof. Mattern 135

Begriffe

- Betrachten Probleme mit einem gewissen *Problemumfang* n
 - Anzahl Eingabewerte, Anzahl der Bits der Eingabe
- Aufwand (Zeit / Speicherplatz) typischerweise von n abhängig
 - Wird daher als Funktion $f(n)$ angegeben
- Beim Zeitaufwand wird i.A. von konstanten Faktoren abstrahiert
 - z.B. $f(n) = n \log n$ statt (genauer) $3 + 7 n \log n$
 - Beim Speicherverbrauch wird oft nicht von Konstanten abstrahiert

136

Begriffe

- Oft ist der Aufwand eines Algorithmus nicht nur von der Problemgrösse n , sondern von den konkreten Eingabewerten (bzw. deren Reihenfolge) abhängig, daher unterscheidet man:
 - günstigster Fall („best case“)
 - mittlerer Fall („average case“)
 - ungünstigster Fall („worst case“)
- Oft ist man nur am (i.Allg. leichter zu bestimmenden) asymptotischen Aufwand (für $n \rightarrow \infty$) als Funktion von n interessiert
- Achtung: für „kleine“ n kann ein Algorithmus mit schlechterem asymptotischen Aufwand besser sein ($\rightarrow$ break even points)!
- Komplexität eines Problems = geringstmöglicher Aufwand, der mit dem dafür besten Lösungsalgorithmus erreicht werden kann
- Manche Probleme sind inhärent aufwendig / schwierig / „komplex“

137

Komplexitätsgrössenordnung

- Zweck: Angabe der Grössenordnung der Komplexität eines Algorithmus als Funktion der Eingabegrösse
 - Zeitkomplexität meist Anzahl „Schritte“
 - Von unwesentlichen Konstanten wird abstrahiert
- Schreibweise

$O(1)$	konstant	} „gut“
$O(\log n)$	logarithmisch	
$O(n)$	linear	
$O(n^2)$	quadratisch	
$O(n^k), k > 0$	polynomial	
$O(c^n)$	exponentiell	} „böse“

Man sagt
"Grössenordnung x"
"O von x"
"Gross-O von x"

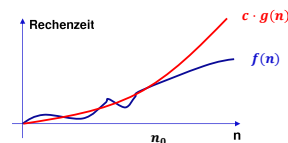
$O(x)$

138

Gross-O Notation

Wir sagen ein Algorithmus f ist $O(g)$ (oder "von der Ordnung g ") wenn für die exakte Komplexität von f gilt:

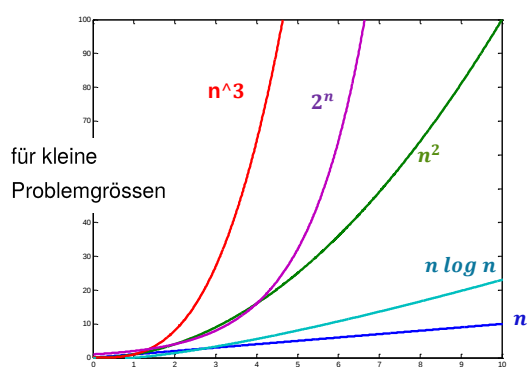
Es gibt eine Konstante c und eine positive ganze Zahl n_0 , so dass $f(n) \leq c \cdot g(n)$ für alle $n \geq n_0$ gilt.



Asymptotische Aussage. Charakterisiert meist nicht Verhalten für kleine Problemgrössen.

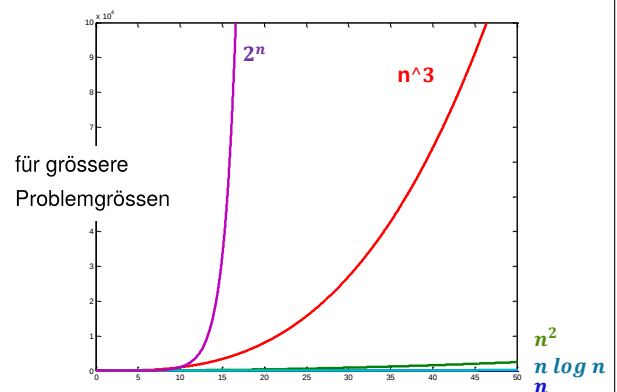
139

Laufzeiten und Problemgrössen



140

Laufzeiten und Problemgrössen



141

Zeitbedarf bei vergrössertem Problemumfang

- Beispiel: Bei $1\mu s$ für $n=1$

	$n=1$	$n=100$	$n=10000$	$n=10^6$
$\log n$	$1\mu s$	$7\mu s$	$13\mu s$	$20\mu s$
n	$1\mu s$	$100\mu s$	$0.01 s$	$1 s$
n^2	$1\mu s$	$0.01 s$	$1.7 min$	$11.5 Tage$
2^n	$1\mu s$	10^{14} Jahr.	$\sim \infty$	$\sim \infty$

142

Eine gute Strategie?

Dann kaufe ich mir eben eine bessere Maschine 

Wenn ich heute ein Problem der Grössenordnung N lösen kann, dann kann ich mit einer zehn mal (!) so schnellen Maschine ...

$f(n)$	alter und neuer Problemumfang
$\log n$	$N \rightarrow N^{10}$
n	$N \rightarrow 10 N$
$n \log n$	$N \rightarrow \text{fast } 10N \rightarrow 10N \text{ für } N \rightarrow \infty$
n^2	$N \rightarrow \sqrt{10} N \sim 3.16 N$
n^3	$N \rightarrow \sqrt[3]{10} N \sim 2.15 N$
2^n	$N \rightarrow N + \log_2 10 \sim N + 3.3$

143

Komplexitätsklassen formeller*

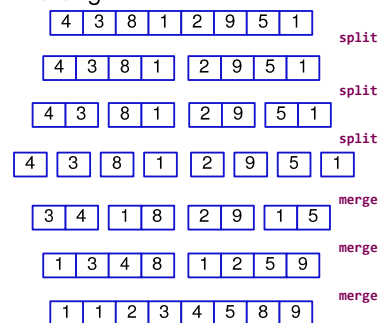
- Etwas formeller bezeichnet das *Landau-Symbol*
 $O(f) = \{h \mid \exists c > 0, \exists n_0 \in \mathbb{N}: \forall n > n_0: h(n) \leq cf(n)\}$
eigentlich eine Klasse von Funktionen.
- Man schreibt zwar manchmal salopp $g = O(f)$,
meint jedoch $g \in O(f)$
- Es gibt auch Beschränkung von unten
 $\Omega(f) = \{h \mid \exists c > 0, \text{für unendlich viele } n: h(n) \geq cf(n)\}$
- und "beides"

$$\Theta(f) = \Omega(f) \cap O(f)$$

144

Beispiel einer Komplexitätsberechnung

- Mergesort, vereinfachende Annahme: $n = 2^d$
zur Erinnerung:



145

Beispiel einer Komplexitätsberechnung

- Aufwand für Merge:
 $T(n) = n + 2 \cdot T\left(\frac{n}{2}\right)$, wenn $n > 1$
 $T(1) = 1$

Warum?
- Komplexitätsberechnung Rekursion $\rightarrow$ Iteration

$$\begin{aligned}
 T(2^d) &= 2^d + 2 \cdot T(2^{d-1}) \\
 &= 2^d + 2 \cdot (2^{d-1} + 2 \cdot T(2^{d-2})) \\
 &= 2^d + 2 \cdot (2^{d-1} + 2 \cdot (2^{d-2} + 2 \cdot T(2^{d-3}))) \\
 &= 2^0 \cdot 2^d + 2^1 \cdot 2^{d-1} + 2^2 \cdot 2^{d-2} + \dots + 2^{d-1} \cdot 2 + 2^d T(1) \\
 &= 2^d \cdot d + 2^d = n \log_2 n + n \\
 &= O(n \log n)
 \end{aligned}$$

d Terme

146

Beispiel einer Komplexitätsberechnung

Wenn man die Formel $n + n \log n$ aufgrund der Problemstruktur schon erraten hat, dann kann man auch *induktiv* beweisen:

- Behauptung
 $n \log n$ ist eine Lösung der Rekursionsgleichung

$$T(n) = \begin{cases} n + 2 T\left(\frac{n}{2}\right) & \text{falls } n > 1 \\ 1 & \text{falls } n = 1 \end{cases}$$

- Induktion

- Anfang: $T(1) = 1 + 1 \log 1 = 1$ klar

- Schritt: $T(n) = n + 2 T\left(\frac{n}{2}\right)$

$$= n + 2 \left(\frac{n}{2} + \frac{n}{2} \log \frac{n}{2} \right) = n + n + n(\log n - 1) = n + n \log n \quad \checkmark$$

147