Komplexität von Algorithmen

Prof. Dr. Christian Böhm

in Zusammenarbeit mit
Gefei Zhang

http://www.dbs.informatik.uni-muenchen.de/Lehre/NFInfoSW

Ressourcenbedarf - Größenordnungen

- Prozesse verbrauchen u.a. zwei Ressourcen:
  - Rechenzeit
  - Speicherplatz
- Die folgenden Größenordnungen geben an, wie viel Rechenzeit oder Speicherplatz verbraucht werden.

Laufzeitanalyse (1)

- *1. Ansatz*: Direktes **Messen der Laufzeit** (z.B. in ms):
  - Abhängig von vielen Parametern, wie Rechnerkonfiguration, Rechnerlast, Compiler, Betriebssystem, …
  - Deshalb: kaum übertragbar und ungenau
- *2. Ansatz*: Zählen der benötigten **Elementaroperationen** des Algorithmus in Abhängigkeit von der Größe *n* der Eingabe
  - Das algorithmische Verhalten wird als Funktion der benötigten Elementaroperationen dargestellt.
  - Die Charakterisierung dieser elementaren Operationen ist abhängig von der jeweiligen Problemstellung und dem zugrunde liegenden Algorithmus.
  - Beispiele für Elementaroperationen: Zuweisungen, Vergleiche, arithmetische Operationen, Zeigerdereferenzierungen oder Arrayzugriffe

Laufzeitanalyse (2)

- Das Maß für die Größe n der Eingabe ist abhängig von der Problemstellung, z.B.
  - Suche eines Elementes in einer Liste: n = Anzahl der Elemente
  - Multiplikation zweier Matrizen: n = Dimension der Matrizen
  - Sortierung einer Liste von Zahlen: n = Anzahl der Zahlen
  - Berechnung der k-ten Fibonacci-Zahl: $n = k$

- Laufzeit = benötigte Elementaroperationen bei einer bestimmten Eingabelänge n

Analog:

- Speicherplatz = benötigter Speicher bei einer bestimmten Eingabelänge n

Problem: Hardware-Abhängigkeit

- Laufzeit einzelner Elementar-Operation ist abhängig von der eingesetzten Rechner-Hardware.
- Frühere Rechner (incl. heutige PDAs, Mobiltelefone…):
  - „billig“: Fallunterscheidungen, Wiederholungen
  - „mittel“: Rechnen mit ganzen Zahlen (Multiplikation usw.)
  - „teuer“: Rechnen mit reellen Zahlen
- Heutige Rechner (incl. z.B. Spiele-Konsolen):
  - „billig“: Rechnen mit reellen und ganzen Zahlen
  - „teuer“: Fallunterscheidungen

Asymptotisches Laufzeitverhalten

- „Kleine“ Probleme (z.B. n=5) sind uninteressant:
  Die Laufzeit des Programms ist eher bestimmt durch die Initialisierungskosten (Betriebssystem, Programmiersprache etc.) als durch den Algorithmus selbst.
- Interessanter:
  - Wie verhält sich der Algorithmen bei sehr großen Problemgrößen?
  - Wie verändert sich die Laufzeit, wenn ich die Problemgröße variiere (z.B. Verdopplung der Problemgröße)

➔ asymptotisches Laufzeitverhalten

Definition *O*-Notation

- Mit der *O*-Notation haben Informatiker einen Weg gefunden, die asymptotische Komplexität (bzgl. Laufzeit oder Speicherplatzbedarf) eines Algorithmus zu charakterisieren.
- Definition *O*-Notation:

Seien $f : \mathrm{N} \rightarrow \mathrm{N}$ und $s : \mathrm{N} \rightarrow \mathrm{N}$ zwei Funktionen (s wie Schranke).
Die Funktion f ist von der Größenordnung $O(s)$, geschrieben $f \in O(s)$, wenn es $k \in \mathrm{N}$ und $m \in \mathrm{N}$ gibt, so dass gilt:
Für alle $n \in \mathrm{N}$ mit $n \geq m$ ist $f(n) \leq k * s(n)$.

Bemerkungen zur *O*-Notation

- k ist unabhängig von n:
  k muss dieselbe Konstante sein, die für alle $n \in \mathrm{N}$ garantiert, dass $f(n) \leq k * s(n)$.
- Existiert KEINE solche Konstante k, ist f nicht von der Größenordnung $O(s)$.
- $O(s)$ bezeichnet die Menge aller Funktionen, die bezüglich s die Eigenschaft aus der Definition haben.
- Man findet in der Literatur häufig $f = O(s)$ statt $f \in O(s)$. Da f nicht gleichzeitig Element von $O(s)$ und gleich zu $O(s)$ sein kann, ist das irreführend.

Rechnen mit der *O*-Notation

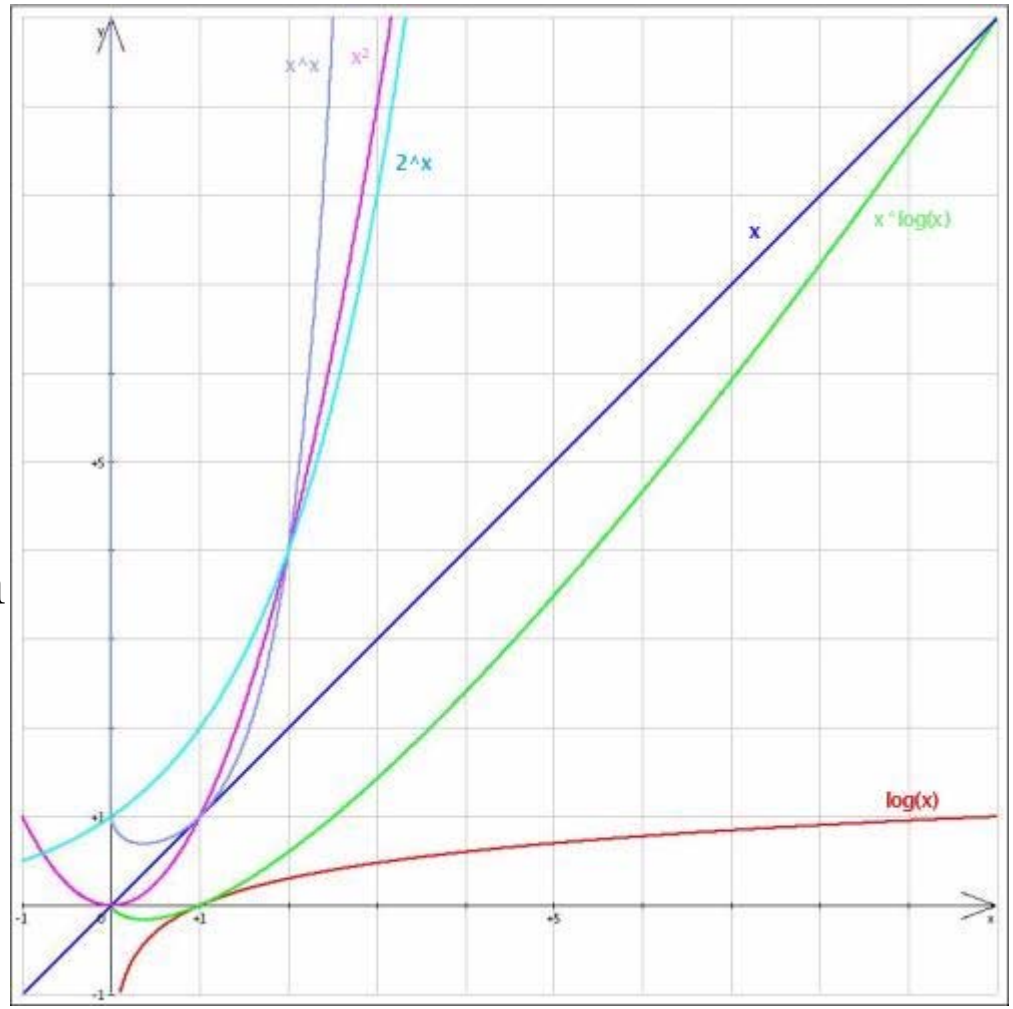
- Elimination von Konstanten:
  - $2 \cdot n \in O(n)$
  - $n/2 + 1 \in O(n)$
- Bei einer Summe zählt nur der am stärksten wachsende Summand (mit dem höchsten Exponenten):
  - $2n^3 + 5n^2 + 10n + 20 \in O(n^3)$
  - $O(1) \subseteq O(\log n) \subseteq O(n) \subseteq O(n \log n)$ $\subseteq O(n^2) \subseteq O(n^3) \subseteq \ldots \subseteq O(2^n)$
- Beachte: 1000 n^2 ist nach diesem Leistungsmaß immer „besser“ als 0,001 n^3, auch wenn das m, ab dem die $O(n^2)$-Funktion unter der $O(n^3)$-Funktion verläuft, sehr groß ist (m=1 Million)

Wichtige Klassen von Funktionen (1)

	Sprechweise	**Typische Algorithmen / Operationen**
$O(1)$	konstant	Addition, Vergleichsoperationen, rekursiver Aufruf, …
$O(\log n)$	logarithmisch	Suchen auf einer sortierten Menge
$O(n)$	linear	Bearbeiten jedes Elementes einer Menge
$O(n \cdot \log n)$		gute Sortierverfahren
$O(n \cdot \log^2 n)$		
…		
$O(n^2)$	quadratisch	primitive Sortierverfahren
$O(n^k)$, k≥2	polynomiell	
…		
$O(2^n)$	exponentiell	Ausprobieren von Kombinationen

Wichtige Klassen von Funktionen (2)

- Die *O*-Notation hilft insbesondere bei der Beurteilung, ob ein Algorithmus für großes *n* noch geeignet ist bzw. erlaubt einen Effizienz-Vergleich zwischen verschiedenen Algorithmen für große *n*.
- Schlechtere als polynomielle Laufzeit gilt als nicht effizient.



Beispiel: Laufzeitanalyse Fakultätsfunktion

```
public static void fak(int n) {
   int result = 1 ;
   for (int i=1 ; i<=n ; i++)
     result *= i ;
   return result ;
 }
```

- Für die Laufzeit T_{fak} von `fak` gilt:

$$T_{fak}(n) \in O(n)$$

- Begründung: Die Schleife wird n mal ausgeführt.

Größter gemeinsamer Teiler

- Der größte gemeinsame Teiler (ggT) zweier natürlicher Zahlen a und b ist die größte natürliche Zahl, durch die sowohl a als auch b teilbar (d.h. ohne Rest dividierbar) ist.
- Die Notation $t|a$ wird verwendet, um auszudrücken, dass t ein Teiler von a ist (d.h. es gibt ein $k \in \mathrm{N}$ mit $a = t * k$).
- Ist t gemeinsamer Teiler von a und b, schreibe $t|a$ und $t|b$.

Primfaktorzerlegung

- Der ggT von zwei natürlichen Zahlen a und b kann leicht aus der Primfaktorzerlegung von a und b ermittelt werden.
- Er ist das Produkt aller p^n mit:
  - die Primzahl p kommt in jeder der beiden Zerlegungen (einmal) vor, einmal mit Exponent n_1, einmal mit Exponent n_2.
  - n ist das Minimum von n_1 und n_2.
- Die Zerlegung einer natürlicher Zahl in Primfaktoren ist eine zeitaufwendige Aufgabe, so dass dieser Ansatz zur Berechnung des ggT zweier natürlicher Zahlen ziemlich ineffizient ist.

Effizienter Ansatz zur Berechnung des ggT zweier natürlicher Zahlen

Satz:
Seien $a \in \mathrm{N}$ und $b \in \mathrm{N}$ mit $a \geq b$.
Sei r der Rest der Ganzzahldivision von a durch b (d.h. $a = (b * c) + r$ für ein $c \in \mathrm{N}$). Sei $t \in \mathrm{N}$.

$t|a$ und $t|b$ genau dann, wenn $t|b$ und $t|r$.

- Daraus folgt, dass der $ggT(a, b)$ zweier natürlicher Zahlen a und b mit $a \geq b$ und $a = b * c + r$ gleich dem $ggT(b, r)$ von b und r ist.

Beweis: Notwendige Bedingung (von links nach rechts „=>“)

- Seien a, b, c und r wie im Satz definiert.
- Sei angenommen, dass $t|a$ und $t|b$.
- Zu zeigen ist, dass $t|b$ und $t|r$ gelten.
- Da nach Annahme $t|b$ gilt, reicht es, $t|r$ zu zeigen.

  Da $t|b$ gilt, gibt es t_b mit $b = t * t_b$.

  Da $t|a$ gilt, gibt es t_a mit $a = t * t_a$.
- Nach Annahme gilt $a = b * c + r$

  also $t * t_a = a = b * c + r = t * t_b * c + r$,

  also $r = t * t_a - t * t_b * c = t * (t_a - t_b * c)$, d.h. $t|r$

Beweis: Hinreichende Bedingung (von rechts nach links „<=“)

- Seien a, b, c und r wie im Satz definiert.
- Sei angenommen, dass $t|b$ und $t|r$.
- Zu zeigen ist, dass $t|a$ und $t|b$ gelten.
- Da nach Annahme $t|b$ gilt, reicht es, $t|a$ zu zeigen.

  Da $t|b$ gilt, gibt es t_b mit $b = t * t_b$.

  Da $t|r$ gilt, gibt es t_r mit $r = t * t_r$.
- Nach Annahme gilt $a = b * c + r$

  also $a = t * t_b * c + t * t_r = t * (t_b * c + t_r)$, d.h. $t|a$

qed.

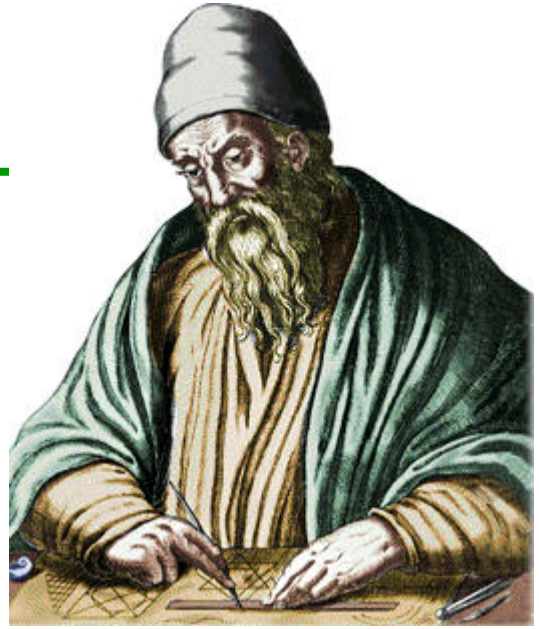
Endrekursive Funktion zur Berechnung des ggT

```
public static int gcd(int a, int b) {
    int h;
    if (a<b) {h=a ; a=b ; b=h}
    while (b != 0) {
       System.out.println ("a=" + a + " b=" + b) ;
       h = a%b ;
       a = b ;
       b = h ;
    }
    return a;
}
```

- Der Algorithmus, den die Funktion `ggT` implementiert, konvergiert sehr schnell:

```
ggT(150, 60):
    a=150 b=60
    a=60 b=30
```

Euklid



- Dieser Algorithmus zur Berechnung des ggT zweier natürlicher Zahlen wird Euklid (ca. 3. Jh. vor Christus) zugeschrieben, weil er in Euklids „Elemente der Mathematik“ erwähnt ist.
- Er gilt als der älteste bekannte Algorithmus, weil er im Gegensatz zu anderen überlieferten Algorithmen aus älteren oder sogar jüngeren Zeiten nicht mittels Beispielen, sondern abstrakt (mit Redewendungen anstelle von Variablen) spezifiziert ist.

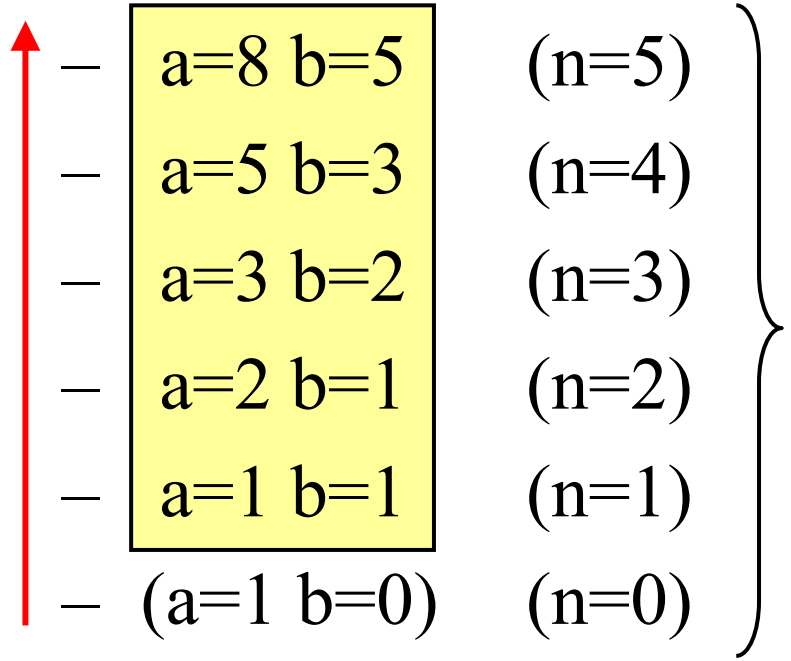
Abschätzung der Rechenzeit des Euklid´schen Algorithmus

Satz (Lamé)
Seien $a \in \mathrm{N}$ und $b \in \mathrm{N}$, so dass $a \geq b$ ist.
Benötigt der Euklid´sche Algorithmus zur Berechnung von $ggT(a, b)$ insgesamt n Iterationen, so gilt $b \geq fib(n)$,
wobei $fib(n)$ die n-te Fibonacci-Zahl ist.

- fib(0) = 0 ; fib(1) = 1 ;
- fib(n)=fib(n-1)+fib(n-2) ;
- Zahlenfolge: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1.597, 2.584, 4.181, 6.765,

Beweis des Satzes (1)

- Wir versuchen, die Ausgabe von unten nach oben zu konstruieren, so dass das minimal mögliche b und das minimal mögliche a entsteht:



Was ist das kleinste mögliche a_n in Zeile Nummer n, das bei Division durch b_n in Zeile n den Rest b_{n-1} in Zeile $(n\text{-}1)$ ergibt?

$$a_n = b_n + b_{n-1}$$
$$b_{n+1} = b_n + b_{n-1}$$

Beweis des Satzes (2)

- Dieses Ergebnis entspricht dem Bildungsgesetz für die Fibonacci-Zahlen.
- Deshalb kann man die Hypothese aufstellen, dass

  $b_0 \geq fib(n)$, wenn n die Anzahl der Schleifeniterationen der Funktion `ggT` während der Berechnung von (a_0, b_0) ist (wobei dieser allererste Aufruf nicht mitgezählt wird).

Bezug zum Goldenen Schnitt

- Benötigt der Euklid´sche Algorithmus zur Berechnung von `ggT(a,b)` genau n rekursive Aufrufe, so gilt nach dem Satz von Lamé:

  $b \geq fib(n) \approx \Phi^n$ ($\Phi = 1{,}618\ldots$ ist der Goldene Schnitt)

- Daraus folgt, dass asymptotisch, also für große n, gilt (mit der Schreibweise $\log_\Phi$ für den Logarithmus zur Basis Φ):

  $\log_\Phi b \geq n$

- Da es aber eine Konstante k gibt mit $\log_\Phi b \leq k * \ln\ b$, wobei `ln` den Logarithmus zur Basis e bezeichnet, gilt asymptotisch auch $n \leq k * \ln\ b$, also

  $ggT(a,b) \in O(\ln(b))$