Algorithmische Mathematik
Vorlesung WS 98/99

Winfried Hochstättler
Zentrum für paralleles Rechnen
Universität zu Köln

26. Oktober 2000

Inhaltsverzeichnis

Kapitel 1

Notation, allgemeine Grundlagen und Rechnerproblematik

Was ist ein *Algorithmus*? Zunächst einmal handelt es sich um eine Verunstaltung des Eigennamens *Abu 'Abdallah Muhammad ibn Musa al-Khwarizmi*. Dieser Mathematiker und Astronom, um 790 in Khorezm im heutigen Usbekistan geboren, leitete am Hofe des siebten Kalifen der Abbessiden 'Abdallah Al Ma'mun das Haus der Weisheiten. Von seinen Schriften sind zwei besonders interessant:

a) al-Kitab al-mukhtasar fi hisab al-jabr wa'l-muqabala (Rechnen durch Ergänzen und Ausgleichen in Kürze). Dieses Buch ist der Lösung linearer und quadratischer Gleichungen gewidmet, und ihm verdanken wir das Wort „Algebra".

b) Kitab hisab al'adad al-hindi, der lateinischen Übersetzung „Algoritmi de numero Indorum"verdanken wir neben dem Wort Algorithmus auch die irreführende Bezeichnung „arabische Zahlen". Al-Khwarizmi rechnet in 10er Notation mit Platzhalter für die Null.

Nach der Bereitstellung des rechnerischen Handwerkszeugs, beschäftigt sich das erste Algebrabuch mit Maßen und Erbschaftsangelegenheiten. Die lateinische Übersetzung des Buches über die Zahlen der Inder führte dazu, daß bei Rechenproblemen im Italien des ausgehenden Mittelalters die Kaufleute nachschauten, was *Algoritmi dixit*. In der Grundschule lernen wir die Algorithmen zur Addition, Subtraktion, Multiplikation und Division.

Ältere Prominente Algorithmen sind das Sieb des Erathostenes und der euklidische Algorithmus zur Bestimmung des größten gemeinsamen Teilers zweier Zahlen.

1.1 Algorithmus

Unter einem Algorithmus verstehen wir eine “Rechenvorschrift”, die bei *Eingabe* einer Probleminstanz eine *Ausgabe* berechnet und

a) in einem Text *endlicher Länge* beschrieben werden kann,

b) deren einzelne Schritte exakt und *eindeutig* festgelegt sind, dabei muß jede mögliche Situation erfaßt sein und die zugehörige Vorgehensweise genau spezifiziert und in endlicher Zeit durchführbar sein.

c) die in endlichen vielen Schritten *terminiert*

Diese informelle Definition eines Algorithmus soll uns hier genügen. In den Informatikvorlesungen haben Sie gelernt, daß man Algorithmen formal über geeignete Maschinenmodelle, wie RAMs oder Turingmaschinen definiert.

1.2 Etwas Notation

Der Allquantor ist $\forall$, der Existenzquantor $\exists$. Wir bezeichnen mir $\mathbb{N}$ $(\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C})$ die Mengen der natürlichen (ganzen, rationalen, reellen bzw. komplexen) Zahlen. Die Menge $\mathbb{N}$ enthält nicht die Null. Mit $\mathbb{Z}_+$, $(\mathbb{Q}_+, \mathbb{R}_+)$ bezeichnen wir die nichtnegativen ganzen (rationalen, reellen) Zahlen.

Für $n \in \mathbb{N}$ bezeichnet $\mathbb{N}^n$ $(\mathbb{Z}^n, \mathbb{Q}^n, \mathbb{R}^n, \mathbb{C}^n)$ die Menge der Vektoren mit n Komponenten mit Einträgen in $\mathbb{N}$ $(\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C})$. Sind E und R Mengen, so bezeichnet R^E die Menge aller Abbildungen von E nach R. Wenn E endlich ist, so betrachten wir die Elemente von R^E auch als $|E|$-Vektoren und schreiben

$$x = (x_e)_{e \in E}.$$

Soweit nicht explizit anders gesagt, sind Vektoren stets *Spaltenvektoren*. Ein hochgestelltes $\top$ bedeutet Transposition, also ist für $x, y \in \mathbb{R}^n$ $x^\top$ ein Zeilenvektor und als Matrixprodukt ist

$$x^\top y = \sum_{i=1}^{n} x_i y_i.$$

Ist α eine reelle Zahl, so bezeichnen wir mit

$$\lceil \alpha \rceil := \min\{x \in \mathbb{Z} \mid \alpha \leq x\}, \quad \lfloor \alpha \rfloor := \max\{x \in \mathbb{Z} \mid \alpha \geq x\}.$$

Sind $a, b \in \mathbb{R}^n$, so bedeutet

$$a \leq b \Leftrightarrow \forall i = 1, \ldots, n : a_i \leq b_i.$$

Sind M, N Mengen, so schreiben wir

$M \subseteq N$ falls M eine Teilmenge von N ist,

$M \subset N$ falls $M \neq N$ eine Teilmenge von N ist,

$M \setminus N := \{x \in M \mid x \notin N\}$ für die Differenz

$M \triangle N := (M \setminus N) \cup (N \setminus M)$ für die symmetrische Differenz

$2^M := \{X \mid X \subseteq M\}$ für die Potenzmenge von M.

Sind darüber hinaus $M, N \subseteq \mathbb{R}^n$ und $\alpha \in \mathbb{R}$, so gelte

$M + N := \{x + y \mid x \in M, y \in N\}$.

$M - N := \{x - y \mid x \in M, y \in N\}$.

$\alpha M := \{\alpha x \mid x \in M\}$.

$M^\perp := \{y \in \mathbb{R}^n \mid \forall x \in M : x^\top y = 0\}$.

Ist R eine Menge, so bezeichnen wir mit $R^{m\times n}$ die Menge der (m, n)-Matrizen mit Einträgen in R, das sind Matrizen mit m Zeilen und n Spalten. Ist $A \in R^{m\times n}$ so heißt der Eintrag in der i-ten Zeile und j-ten Spalte $a_{i,j}$ oder $A_{i,j}$ manchmal auch ohne Komma im Index. Wir schreiben auch $A = (a_{i,j})_{\substack{i=1,\ldots,m\\ j=1,\ldots,n}}$ oder kurz $A = (a_{i,j})$. Sind M, N die Zeilen- bzw. Spaltenindexmenge von A, $I \subseteq M$ und $J \subseteq N$, so bezeichnen wir mit $A_{I,J}$ die Matrix $(a_{i,j})_{i\in I, j\in J}$, statt $A_{I,N}$ $(A_{M,J})$ schreiben wir kurz $A_{I.}(A_{.J})$. Mit I_n oder kurz I bezeichnen wir die *Einheitsmatrix*, d.i. die $(n \times n)$-Matrix mit Einsen auf der Diagonale und sonst lauter Nullen.

Mit $e_i \in \mathbb{R}^n$ bezeichnen wir den i-ten Einheitsvektor, d.h. $(e_i)_j = 1$ falls $i = j$ und 0 sonst.

Ist $x \in \mathbb{R}^n$, so bezeichnet, falls nicht ausdrücklich anders definiert, $\|x\|$ stets die *euklidische Norm* oder L_2-Norm

$$\|x\|_2 := \sqrt{x^\top x} = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

Manchmal verwenden wir auch andere Normen, wie etwa

die L_1- oder 1-Norm: $\|x\|_1 := \sum_{i=1}^n |x_i|$

die L_∞- oder Maximumsnorm: $\|x\|_\infty := \max_{1\leq i\leq n} |x_i|$.

Ist $P \subseteq \mathbb{R}^n$ und $\varepsilon > 0$ so bezeichnen wir mit

$$U_\varepsilon(P) := \{x \in \mathbb{R}^n \mid \exists p \in P : \|x - p\| < \varepsilon\}$$

die offene ϵ-Umgebung der Menge P. Ist $P = \{p\}$ einelementig, so schreiben wir statt $U_\varepsilon(\{p\})$ kurz $U_\varepsilon(p)$. Zusätzlich definieren wir $U_\varepsilon(\{\infty\}) = \{x \in \mathbb{R}^n \mid \|x\| > \frac{1}{\varepsilon}\}$.

1.3 Kodierung von Zahlen

Wie wir bereits oben erwähnten, war eine der wesentlichen Leistungen von al-Khwarizmi, daß er die Stellennotation im Abendland bekannt machte. Mit römischen Zahlen läßt sich nämlich schlecht rechnen. Was steckt nun genau hinter dieser Zahldarstellung? Jede Stelle steht für eine Zehnerpotenz, wir haben Basis Zehn. Mit dem folgenden Satz werden wir beweisen, daß man bzgl. jeder beliebigen Basis eine eindeutige Darstellung jeder reellen Zahl gewinnen kann.

Satz 1.3.1 *Sei $B \in \mathbb{N}, B \geq 2$, und sei $x \in \mathbb{R} \setminus \{0\}$. Dann gibt es genau eine Darstellung der Gestalt*

$$x = \sigma B^n \sum_{i=1}^{\infty} x_{-i} B^{-i} \tag{1.1}$$

mit $\sigma \in \{+1, -1\}, n \in \mathbb{Z}$ und $x_{-i} \in \{0, 1, \ldots, B-1\}, x_{-1} \neq 0$ und zusätzlich $\forall j \in \mathbb{N} \exists k \geq j : x_{-k} \neq B - 1$.

Beweis. Existenz. Wir setzen

$$\sigma := \operatorname{sign}(x), n := \lfloor \log_B(|x|) \rfloor + 1 \text{ und } x_{-i} := \lfloor |x| B^{i-n} \rfloor \bmod B$$

und müssen nachweisen, daß dann die Reihe gegen x konvergiert. Dazu zeigen wir mittels vollständiger Induktion zunächst

$$x_{-i} = \lfloor (|x| - B^n \sum_{j=1}^{i-1} x_{-j} B^{-j}) B^{i-n} \rfloor. \tag{1.2}$$

Nach Definition von n ist $B^{n-1} \leq |x| < B^n$ und somit zunächst einmal

$$\begin{aligned} x_{-1} &= \lfloor |x| B^{1-n} \rfloor \bmod B \\ &= \lfloor |x| B^{1-n} \rfloor \\ &= \lfloor (|x| - B^n \sum_{j=1}^{1-1} x_{-j} B^{-j}) B^{1-n} \rfloor \\ &\geq 1 \end{aligned}$$

und somit (1.2) gezeigt für $i = 1$. Sei nun $i_0 \geq 2$ und die Behauptung für alle $i \leq i_0 - 1$ bewiesen. Nach Induktionsvoraussetzung ist dann

$$0 \leq (|x| - B^n \sum_{j=1}^{i_0-2} x_{-j} B^{-j}) B^{i_0-1-n} - x_{i_0-1} < 1.$$

Wir schließen hieraus

$$0 \leq (|x| - B^n \sum_{j=1}^{i_0-2} x_{-j} B^{-j}) B^{i_0-n} - B^{n-i_0+1+(i_0-n)} x_{i_0-1} < B$$

und somit

$$\begin{aligned} x_{-i_0} &= \lfloor |x| B^{i_0-n} \rfloor \bmod B \\ &= \lfloor (|x| - B^n \sum_{j=1}^{i_0-1} x_{-j} B^{-j}) B^{i_0-n} \rfloor \end{aligned}$$

womit (1.2) bewiesen ist.

Für den Konvergenzbeweis müssen wir nach Mathematik für Wirtschaftsinformatiker I zeigen, daß es zu jedem $\varepsilon > 0$ ein $i_0 \in \mathbb{N}$ gibt mit

$$\forall N \geq i_0 : |x - \sigma B^n \sum_{i=1}^{N} x_{-i} B^{-i}| < \varepsilon.$$

Sei also $\varepsilon > 0$ vorgegeben und $i_0 = \lceil n + 2 - \log_B \varepsilon \rceil$. Nach dem soeben Bewiesenen ist

$$0 \leq (|x| - B^n \sum_{j=1}^{i_0-1} x_{-j} B^{-j}) B^{i_0-n} < B$$

und somit

$$
\begin{aligned}
|x - \sigma B^n \sum_{j=1}^{N} x_{-j} B^{-j}| &= \left| |x| - B^n \sum_{j=1}^{N} x_{-j} B^{-j} \right| \\
&\leq \left| |x| - B^n \sum_{j=1}^{i_0-1} x_{-j} B^{-j} \right| + B^n \sum_{j=i_0}^{\infty} (B-1) B^{-j} \\
&< B^{n+1-i_0} + \frac{B^n (B-1)}{B^{i_0}(1-\frac{1}{B})} \\
&\leq B^{n+2-\lceil n+2-\log_B \varepsilon \rceil} \\
&= B^{\lfloor \log_B \varepsilon \rfloor} \leq \varepsilon.
\end{aligned}
$$

Nehmen wir nun an, es gäbe ein $j_0 \in \mathbb{N}$ mit $\forall k \geq j_0 : x_{-k} = B - 1$. Dann ist

$$
\begin{aligned}
x_{-(j_0-1)} &= \lfloor (|x| - B^n \sum_{k=1}^{j_0-2} x_{-k} B^{-k}) B^{j_0-1-n} \rfloor \\
&= \lfloor (B^n \sum_{i=1}^{\infty} x_{-i} B^{-i} - B^n \sum_{k=1}^{j_0-2} x_{-k} B^{-k}) B^{j_0-1-n} \rfloor \\
&= \lfloor B^{j_0-1} \sum_{i=j_0-1}^{\infty} x_{-i} B^{-i} \rfloor \\
&= \lfloor x_{-(j_0-1)} + (B-1) \sum_{i=j_0}^{\infty} B^{j_0-1-i} \rfloor \\
&= x_{-(j_0-1)} + \lfloor (B-1) \sum_{i=1}^{\infty} B^{-i} \rfloor \\
&= x_{-(j_0-1)} + \lfloor (B-1) \frac{1}{B} \frac{1}{1-\frac{1}{B}} \rfloor \\
&= x_{-(j_0-1)} + 1.
\end{aligned}
$$

Aus diesem Widerspruch folgt die Behauptung. Es bleibt die Eindeutigkeit zu zeigen. Angenommen also $\sigma_1 B^{n_1} \sum_{i=1}^{\infty} x_{-i} B^{-i} = \sigma_2 B^{n_2} \sum_{i=1}^{\infty} y_{-i} B^{-i}$. Offensichtlich muß wegen $x \neq 0$ dann $\sigma_1 = \sigma_2$ sein. Aus $x_{-1} \neq 0 \neq y_{-1}$ und

$$
\begin{aligned}
\sum_{i=2}^{\infty} z_{-i} B^{-i} &< \sum_{i=2}^{\infty} (B-1) B^{-i} \\
&= \frac{B-1}{B^2} \frac{1}{1-\frac{1}{B}} \\
&= B^{-1}
\end{aligned}
$$

falls alle $z_{-i} \in \{0, \ldots, B-1\}$ aber nicht alle identisch $B-1$ sind, schließen wir $n_1 = n_2$. Es bleibt zu zeigen $x_{-i} = y_{-i}$ für alle i. Angenommen dies wäre nicht so. Dann betrachten wir $0 = \sum_{i=1}^{\infty}(x_{-i} - y_{-i})B^{-i}$. Nach Annahme gibt es einen kleinsten Index i_0 mit $x_{-i_0} \neq y_{-i_0}$. Durch eventuelles Vertauschen der Namen können wir o.B.d.A. (ohne Beschränkung der Allgemeinheit) annehmen, daß $x_{-i_0} < y_{-i_0}$. Dann erhalten wir

$$\begin{aligned} 1 \leq y_{-i_0} - x_{-i_0} &= \sum_{i=i_0+1}^{\infty}(x_{-i} - y_{-i})B^{-i+i_0} \\ &= \sum_{i=1}^{\infty}(x_{-i-i_0} - y_{-i-i_0})B^{-i} \\ &\leq \sum_{i=1}^{\infty}(B-1)B^{-i} \\ &\leq (B-1)\frac{1}{B-1} = 1. \end{aligned}$$

Demnach muß in allen Ungleichungen dieser Kette Gleichheit herrschen. Hieraus folgt aber, daß $x_i = B - 1$ und $y_i = 0$ für alle $i > i_0$, was wir ausgeschlossen hatten. □

Im Rechner verwenden wir bekanntermaßen $B \in \{2, 8, 16\}$.

1.4 Fehlerquellen und Beispiele

Man kann keine Reihen mit unendlich vielen Gliedern bei vorgegebenem endlichen Speicher darstellen. Beim praktischen Rechnen ist man deshalb gezwungen zu *runden*. Es seien $x, \tilde{x} \in \mathbb{R}$ und $\tilde{x}$ eine Näherung für x sei. Dann heißen

a) $x - \tilde{x}$ der *absolute Fehler* und

b) für $x \neq 0$, $\frac{x-\tilde{x}}{x}$ der *relative Fehler*.

Bei der Darstellung von Zahlen bzgl. einer geraden Basis $B \in \mathbb{N}, B \geq 2$ wollen wir folgende Rundungsvorschrift für t-stellige Darstellung mit $x \in \mathbb{R} \setminus \{0\}$ für $x = \sigma B^n \sum_{i=1}^{\infty} x_{-i}B^{-i}$ benutzen:

$$Rd_t(x) := \begin{cases} \sigma B^n \sum_{i=1}^{t} x_{-i}B^{-i} & \text{falls } x_{-t-1} < \frac{B}{2} \\ \sigma B^n (B^{-t} + \sum_{i=1}^{t} x_{-i}B^{-i}) & \text{falls } x_{-t-1} \geq \frac{B}{2} \end{cases}$$

Sei nun eine *Maschinengenauigkeit* $t \in \mathbb{N}$ vorgegeben. Dann bezeichnen wir Zahlen mit einer Darstellung der Gestalt $\sigma B^n \sum_{i=1}^t x_{-i} B^{-i}$ als *Maschinenzahlen* oder *Gleitkommazahlen*. Dabei heißt n der *Exponent*, σ das *Vorzeichen* und $\sum_{i=1}^t x_{-i} B^{-i}$ die *Mantisse* der Zahl. Wir sagen auch die Zahl habe eine *t-stellige Mantisse*.

Satz 1.4.1 *Sei $k \in \mathbb{N}, B = 2k, t \in \mathbb{N}$ und $x = \sigma B^n \sum_{i=1}^{\infty} x_{-i} B^{-i} \neq 0$. Dann gilt:*

a) *$Rd_t(x)$ hat eine Darstellung der Gestalt $\sigma B^{n'} \sum_{i=1}^t x'_{-i} B^{-i} \neq 0$,*

b) *der absolute Fehler ist beschränkt durch $|x - Rd_t(x)| \leq \frac{B^{n-t}}{2}$,*

c) *der relative Fehler ist beschränkt durch $\left|\frac{x - Rd_t(x)}{x}\right| \leq \frac{B^{1-t}}{2}$,*

d) *der relative Fehler bzgl. $Rd_t(x)$ ist beschränkt durch $\left|\frac{x - Rd_t(x)}{Rd_t(x)}\right| \leq \frac{B^{1-t}}{2}$.*

Beweis. Übung.

Bei der Verknüpfung zweier Gleitkommazahlen, wie Addition, Subtraktion, Multiplikation und Division werden wir nun zunächst das Ergebnis möglichst genau berechnen und dann auf t Stellen runden. Betrachten wir dies zum Beispiel bei der Addition.

Beispiel 1.4.2 *Wir wollen mit 4-stelliger Arithmetik (d.h. 4-stelliger Mantisse) zur Basis 10 die Zahlen 12,34 und -0,09876 addieren.*

$$
\begin{array}{cccccccccccc}
+ & . & 1 & 2 & 3 & 4 & & & & & & 2 \\
- & . & 9 & 8 & 7 & 6 & & & & & & -1 \\
 & & & & & & & & & & & \\
+ & . & 1 & 2 & 3 & 4 & 0 & 0 & 0 & 0 & & 2 \\
- & . & 0 & 0 & 0 & 9 & 8 & 7 & 6 & 0 & & 2 \\
\hline
+ & . & 1 & 2 & 2 & 4 & 1 & 2 & 4 & 0 & & 2 \\
\hline
+ & . & 1 & 2 & 2 & 4 & & & & & & 2
\end{array}
$$

Das Runden von Zahlen liefert also ein zusätzliches Fehlerpotential bei der numerischen Lösung eines Problems. Andere Fehlerquellen sind *Datenfehler* oder *Verfahrensfehler*. Letzteres sind etwa Ungenauigkeiten, die dadurch enstehen, daß iterative, konvergente Prozesse nach endlich vielen Schritten abgebrochen werden. Alle diese Fehler können durch *Fehlerfortpflanzung* verstärkt werden, wie in der „Mathematik für Chemiker und Wirtschaftsinformatiker“ diskutiert.

In den Übungen wollen wir diskutieren, wie man durch geschickte Auswertung von Formeln die Auswirkungen von Rundungsfehlern verkleinern kann.

1.5 Komplexität

Unter der *Komplexität* eines Algorithmus verstehen wir üblicherweise das Laufzeitverhalten des Verfahrens. Um dies zu definieren müssen wir uns zunächst auf einen Satz von Elementarschritten einigen, deren Ausführung in einem Schritt bewältigt werden kann. Wir haben eben gesehen, daß die Addition zweier Maschinenzahlen konstanten Aufwand verursacht. Der Einfachheit halber woller wir $+, -, \cdot, :$ sowie Vergleichsoperationen $=, <, >, \leq$ etc. und Zuweisungen zu den Elementarschritten zählen. Mit diesen Definitionen können wir bei gegebenem Input die Laufzeit des Algorithmus A bei Input I definieren

$$L_A(I) := |\text{Elementarschritte, die } A \text{ ausführt, bis er terminiert.}|$$

Da wir bei Algorithmen üblicherweise an dem Verhalten bzgl. mehrer Inputbeispiele interessiert sind, betrachten wir das Laufzeitverhalten in Abhängigkeit von der Länge $\langle I \rangle$ der (sinnvoll kodierten) Inputdaten $\langle I \rangle$ und bezeichen als *worst-case Komplexität*

$$L_A(n) := \max\{L_A(I) \mid \langle I \rangle \leq n\}.$$

Beispiel 1.5.1 *Der euklidische Algorithmus zur Bestimmung des größten gemeinsamen Teilers zweier ganzer Zahlen m und n läßt sich wie folgt darstellen.*

```
#!/zpr/local/bin/python          # Pfad fuer den Interpreter
import sys                       # Modul fuer sys
import string                    # Modul fuer string

m=string.atoi(sys.argv[1])       # lese erste Zahl ein
n=string.atoi(sys.argv[2])       # lese zweite Zahl ein
r= m % n                         # r = m (mod) n
while r!=0:
  m=n
  n=r
  r= m % n                       # r = m (mod) n
print n
```

Sei m_i der Inhalt der Variablen m in der i-ten Iteration. Betrachten wir die Änderung der Variablen m beim Durchlauf zweier Schleifen. Offensichtlich ist stets $m_{i+1} < m_i$. Gilt nun $m_{i+1} \leq \frac{m_i}{2}$, so ist offensichtlich $m_{i+2} + m_{i+1} \leq 2m_{i+1} \leq m_i$. Andererseits folgt aber aus $m_{i+1} > \frac{m_i}{2}$ sofort $m_{i+2} = m_i - m_{i-1}$ also wiederum $m_{i+2} + m_{i+1} \leq m_i$. In zwei Schleifendurchläufen wird also die Größe der Zahl m mindestens halbiert. Hieraus schließen wir, daß die Gesamtlaufzeit des Algorithmus beschränkt ist durch $4(\log_2(m) + 1)$. *Also terminiert der Algorithmus.*

Wir schreiben $d|n$ falls d die Zahl n teilt, *bzw. $d \not| n$ im anderen Fall. Es bleibt zu zeigen, daß das Verfahren tatsächlich die größte Zahl $ggT(m,n)$ berechnet, die m und n teilt. Dafür zeigen wir, daß, wenn $m > n, n \not| m$ aber $d|n$ und $d|m$ auch gilt: $d|(m \bmod n)$. Denn $m = dk_1, n = dk_2$ und $m \bmod n = m - k_3 n$ impliziert $m \bmod n = (k_1 - k_2 k_3)d$. Der Algorithmus terminiert im r-ten Schritt, wenn $n_r|m_r$ und nach dem Gezeigten gilt dann $ggT(n,m)|n_r = m_{r-1} \bmod n_{r-1}$ und somit $ggT(n,m)|n_r$.*

Exaktes Buchhalten bei Algorithmen ist schwierig. Es erweist sich als nützlich für das Verhalten der Kompexitätsfunktion die sogenannten *Landau-Symbole* einzuführen. Seien dazu $f, g : D \to \mathbb{R}$ zwei Funktionen mit $D \subseteq \mathbb{R}$ und $x_0 \in \mathbb{R} \cup \{\infty\}$.

a) f heißt *von der Ordnung $O(g)$*, wir schreiben $f = O(g)$, *für x gegen x_0*, falls es $C > 0, \delta > 0$ gibt mit

$$|f(x)| \leq C|g(x)| \text{ für } x \in U_\delta(x_0).$$

b) f heißt *von der Ordnung $o(g)$*, wir schreiben $f = o(g)$, *für x gegen x_0*, falls für alle $C > 0$ es ein $\delta > 0$ gibt mit

$$|f(x)| \leq C|g(x)| \text{ für } x \in U_\delta(x_0).$$

Den wichtigen Spezialfall $n \to \infty$, den wir für die Laufzeitfunktion benötigen, wollen wir noch einmal extra notieren.

Proposition 1.5.2 *Sei A ein Algorithmus mit Laufzeitfunktion L_A und $g : \mathbb{N} \to \mathbb{N}$. Dann ist L_A in $O(g)$*[1] *genau dann, wenn*

$$\exists C > 0 \, \exists N_0 \in \mathbb{N} \, \forall N \geq N_0 : L_A(N) \leq Cg(N).$$

Beweis. $L_A = O(g)$ für x gegen ∞, wenn es ein $C > 0$, und ein $\delta > 0$ gibt mit $|L_A(N)| \leq C|g(N)|$ für $N \in U_\delta(\infty)$.

„⇒“ Wir setzen $N_0 = \frac{1}{\delta} + 1$. Dann ist $N \in U_\delta(\infty) \Leftrightarrow N > \frac{1}{\delta} = N_0 - 1 \Leftrightarrow N \geq N_0$.

„⇐“ Wir setzen $\delta = \frac{1}{N_0 - 1}$ und schließen wie eben.

□

[1] Warum ist hier nur $x_0 = \infty$ sinnvoll?

Beispiel 1.5.3 *Wir betrachten die Laufzeitfunktion des euklidischen Algorithmus. Die Inputgröße der Daten ist bei sinnvoller Kodierung* $\log_2(m) + \log_2(n) + 3$. *Wir dürfen annehmen* $n \geq 2$. *Für die Laufzeitfunktion* L_{euklid} *gilt also*

$$L_{euklid}(\log_2(m) + \log_2(n) + 3) \leq 4(\log_2(m) + 1).$$

Nach Definition ist die Laufzeitfuntion monoton wachsend und wir schließen

$$L_{euklid}(\log_2(m)) \leq 4\log_2(m) \textit{ für } m \geq n \geq 2.$$

Somit ist also $L_{euklid}(k) = O(k)$, *wir sagen, der euklidische Algorithmus ist linear. Üblich ist auch die Bezeichnung* $L_{euklid} = O(n)$, *weil mit n oft die Inputlänge bezeichnet wird, was in diesem Fall eine Fußfalle darstellt.*

Wir wollen auch die *Komplexität eines Problems* definieren und zwar als als die Laufzeitfunktion eines *besten* Algorithmus. Wir können also z.B. etwas salopp sagen, daß das Problem der Bestimmung des größten gemeinsamen Teilers zweier Zahlen $O(n)$ ist.

Üblicherweise definiert ein Problem im mathematische Sinne eine Funktion, die einer gegebenen Instanz des Problems die (eine) zugehörige Lösung zuordnet. Umgekehrt können wir jede gegebene Funktion f als Problem betrachten und nach einem Algorithmus fragen, der diese Funktion *berechnet*. Falls ein solcher existiert, nennen wir die Funktion *berechenbar* sonst *nicht berechenbar*. Der folgende Satz zeigt, daß es bei der Lösung mancher Probleme prinzipielle Schwierigkeiten geben kann.

Satz 1.5.4 *Es gibt eine nicht berechenbare Funktion* $\pi : \mathbb{N} \to \{0, 1\}$.

Beweis. Nach Definition läßt sich jeder Algorithmus in einem endlichen Text beschreiben, insbesondere also auch jeder Algorithmus, der eine Funktion mit Werten in $\{0, 1\}$ berechnet. Indem wir diese Algorithmen nach ihrer Länge und dann lexikographisch ordnen, können wir sie abzählen, also jedem $n \in \mathbb{N}$ einen Algorithmus A_n zuordnen. Wir definieren nun $\pi(n) = 1 - A_n(n)$. Dann stimmt π mit keiner Funktion überein, die von einem Algorithmus berechnet wird. $\square$