

Skript zur Vorlesung:
**Einführung in die
Programmierung**
WiSe 2009 / 2010

Skript © 2009 Christian Böhm, Peer Kröger, Arthur Zimek

Prof. Dr. Christian Böhm
Annahita Oswald
Bianca Wackersreuther

Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme



- Algorithmen verbrauchen zwei Ressourcen:
 - Rechenzeit
 - Speicherplatz
- Im folgenden betrachten wir, wie wir messen können, wieviel Rechenzeit und Speicherplatz verbraucht werden.

- **1. Ansatz:** Direktes *Messen* der Laufzeit (z.B. in ms).
 - Abhängig von vielen Parametern (Rechnerkonfiguration, Rechnerlast, Compiler, Betriebssystem, . . .)
 - Daher: 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, Arrayzugriffe, . . .

- Das Maß für die Größe n der Eingabe ist abhängig von der Problemstellung, z.B.
 - Suche eines Elementes in einem Array: $n =$ Länge des Arrays.
 - Multiplikation zweier Matrizen: $n =$ Dimension der Matrizen.
 - Sortierung einer Liste von Zahlen: $n =$ Anzahl der Zahlen.
- **Laufzeit:**
Benötigte Elementaroperationen bei einer bestimmten Eingabelänge n .
- **Speicherplatz:**
Benötigter Speicher bei einer bestimmten Eingabelänge n .

- 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

- “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 Algorithmus bei sehr großen Problemgrößen?
 - Wie verändert sich die Laufzeit, wenn ich die Problemgröße variere (z.B. Verdopplung der Problemgröße)?
- Das bringt uns zu dem Konzept
“**asymptotisches Laufzeitverhalten**”.

- Mit der *O -Notation* haben Informatiker einen Weg gefunden, die asymptotische Komplexität (bzgl. Laufzeit oder Speicherplatzbedarf) eines Algorithmus zu charakterisieren.
- Definition O -Notation:
Seien $f: \mathbb{N} \rightarrow \mathbb{N}$ und $s: \mathbb{N} \rightarrow \mathbb{N}$ zwei Funktionen (s wie *Schranke*)
Die Funktion f ist *von der Größenordnung* $O(s)$, geschrieben $f \in O(s)$, wenn es $k \in \mathbb{N}$ und $m \in \mathbb{N}$ gibt, so dass gilt:
Für alle $n \in \mathbb{N}$ mit $n \geq m$ ist $f(n) \leq k \cdot s(n)$.
- Man sagt auch: f wächst höchstens so schnell wie s .

- k ist unabhängig von n :
 k muss dieselbe Konstante sein, die für alle $n \in \mathbb{N}$ garantiert, dass $f(n) \leq k \cdot s(n)$.
- 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 $f \in O(s)$.

- 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):
 - $2n^3 + 5n^2 + 10n + 20 \in O(n^3)$
 - $O(1) \subseteq O(\log n) \subseteq O(n) \subseteq O(n \log n) \subseteq O(n^2) \subseteq O(n^3) \subseteq \dots \subseteq O(2^n)$
- Beachte:

$1000 \cdot n^2$ ist nach diesem Leistungsmaß immer “besser” als $0,001 \cdot n^3$, auch wenn das m , ab dem die $O(n^2)$ -Funktion unter der $O(n^3)$ -Funktion verläuft, in diesem Fall sehr groß ist ($m=1$ Million).

Wichtige Klassen von Funktionen

	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
$O(n^k), k \geq 2$	polynomiell	
		⋮
$O(2^n)$	exponentiell	Ausprobieren von Kombinationen

- 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, kann aber für viele Probleme das best-mögliche sein.

Wichtige Klassen von Funktionen

