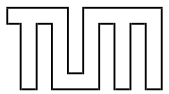
Grundlagen: Algorithmen und Datenstrukturen

Prof. Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Sommersemester 2010



Übersicht

Effizienzmessung

Hauptziel: Effiziente Algorithmen

Exakte Spezifikation der Laufzeit eines Algorithmus
(bzw. einer DS-Operation):

- I: Menge der Instanzen
- $T : I \mapsto \mathbb{N}$: Laufzeit des Algorithmus

Problem: T sehr schwer exakt bestimmbar

Lösung: Gruppierung der Instanzen nach einem bestimmten Kriterium, meist nach Größe

Eingabekodierung

Was ist die Größe einer Instanz?

- Speicherplatz in Bits oder Wörtern

Aber Vorsicht bei der Kodierung!

Beispiel: Primfaktorisierung

Gegeben: Zahl $x \in \mathbb{N}$

Gesucht: Primfaktoren von x, d.h. Primzahlen $p_1, \ldots, p_k$ mit $x = \prod_i p_i^{e_i}$

Bekannt als hartes Problem (wichtig für RSA!)

Eingabekodierung - Beispiel Primfaktorisierung

Trivialer Algorithmus

Teste von $y = 2$ bis $\lfloor \sqrt{x} \rfloor$ alle Zahlen, ob diese x teilen und wenn ja, dann bestimme wiederholt das Ergebnis der Division bis die Teilung nicht mehr ohne Rest möglich ist

Laufzeit: $\sqrt{x}$ Teilbarkeitstests und höchstens $\log_2 x$ Divisionen

- Unäre Kodierung von x (x Einsen als Eingabe): Linearzeitalgorithmus (Anzahl der Teilbarkeitstests und Divisionen ist sogar sublinear, aber die Eingabe muss einmal ganz gelesen werden)
- Binäre Kodierung von x ($\lceil \log_2 x \rceil$): Laufzeit exponentiell (bezüglich der Länge der Eingabe)

Eingabekodierung

Betrachtete Eingabegröße:

- Größe von Zahlen: binäre Kodierung
- Größe von Mengen/Folgen von Zahlen: hier wird oft nur die Anzahl der Elemente betrachtet

Beispiel (Sortieren)

Gegeben: Folge von Zahlen $a_1, \ldots, a_n \in \mathbb{N}$

Gesucht: sortierte Folge der Zahlen

Größe der Eingabe: n

Effizienzmessung

Für ein gegebenes Problem sei I_n die Menge der Instanzen der Größe n.

Effizienzmaße:

- Worst case: $t(n) = \max\{T(i) : i \in I_n\}$
- Average case: $t(n) = \frac{1}{|I_n|} \sum_{i \in I_n} T(i)$
- Best case: $t(n) = \min\{T(i) : i \in I_n\}$

Effizienzmaße: Vor- und Nachteile

- average case:
  muss nicht unbedingt übereinstimmen mit dem "typischen Fall" in der Praxis (dieser ist aber schwer formal zu erfassen)
- worst case:
  liefert Garantien für die Effizienz des Algorithmus
  (auch wichtig für Robustheit)
- best case:
  dient eher für Nachweis einer unzureichenden Performance

- exakte Formeln für $t(n)$ sehr aufwendig

$\Rightarrow$ betrachte asymptotisches Wachstum

Asymptotische Notation

Intuition:

- Zwei Funktionen $f(n)$ und $g(n)$ haben das gleiche Wachstumsverhalten, falls für genügend große n das Verhältnis der beiden nach oben und unten durch Konstanten beschränkt ist, d.h.

$$\exists c, d, n_0 \; \forall n \geq n_0 : \quad c < \frac{f(n)}{g(n)} < d \quad \text{und} \quad c < \frac{g(n)}{f(n)} < d$$

- Warum die Forderung nur für genügend große n?
  Ziel: Verfahren, die auch für große Instanzen noch effizient sind (man sagt "sie skalieren gut")

Beispiel

n^2 und $5n^2 - 7n$ haben das gleiche Wachstum, da für alle $n \geq 2$ $\frac{1}{5} < \frac{(5n^2-7n)}{n^2} < 5$ und $\frac{1}{5} < \frac{n^2}{(5n^2-7n)} < 5$ gilt.

Asymptotische Notation

Mengen zur Formalisierung des asymptotischen Verhaltens:

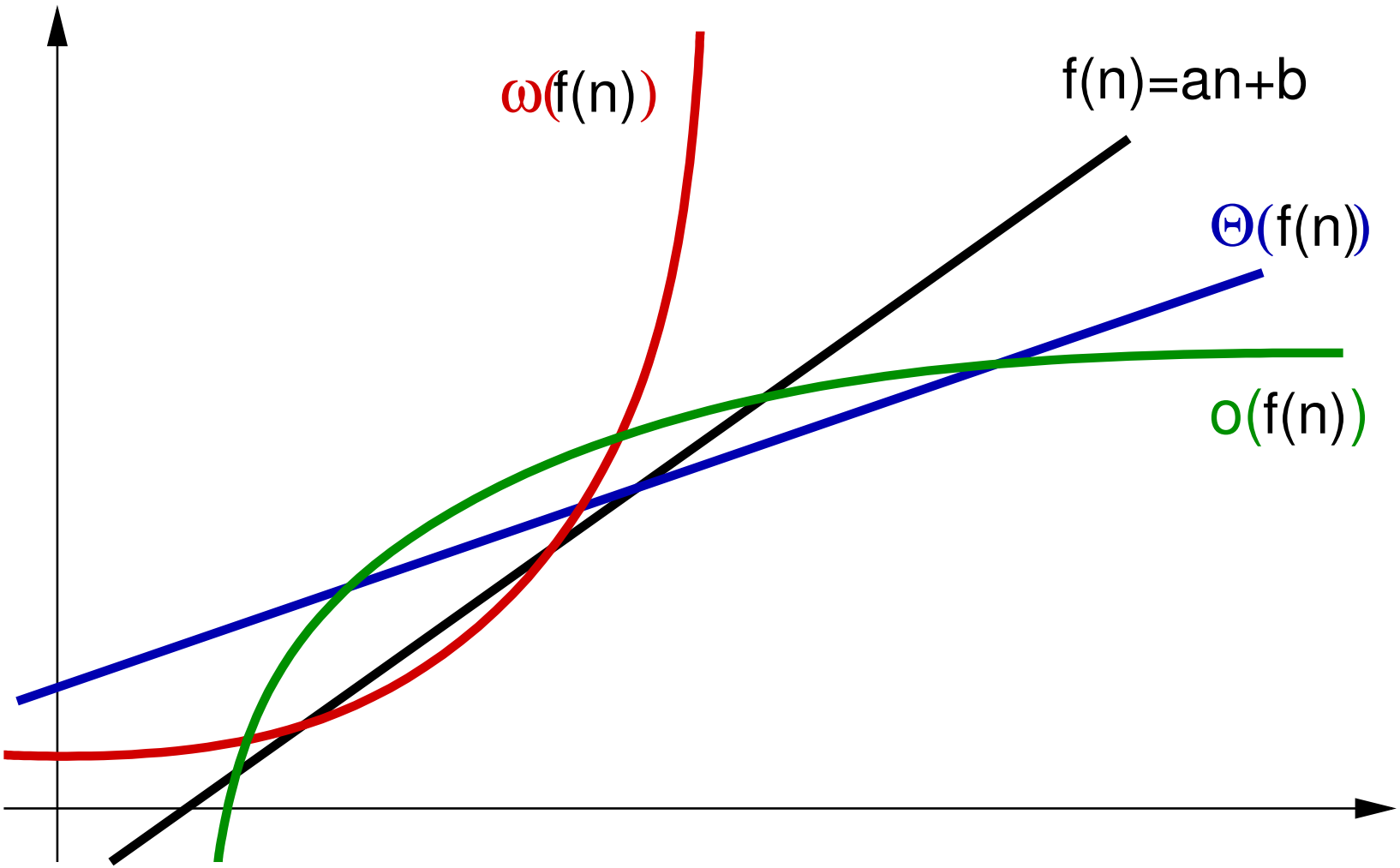
$$\begin{aligned}
\mathcal{O}(f(n)) &= \{g(n) \mid \exists c > 0\ \exists n_0 > 0\ \forall n > n_0 : g(n) \leq c \cdot f(n)\} \\
\Omega(f(n)) &= \{g(n) \mid \exists c > 0\ \exists n_0 > 0\ \forall n > n_0 : g(n) \geq c \cdot f(n)\}
\end{aligned}$$

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

$$\begin{aligned}
o(f(n)) &= \{g(n) \mid \forall c > 0\ \exists n_0 > 0\ \forall n > n_0 : g(n) \leq c \cdot f(n)\} \\
\omega(f(n)) &= \{g(n) \mid \forall c > 0\ \exists n_0 > 0\ \forall n > n_0 : g(n) \geq c \cdot f(n)\}
\end{aligned}$$

Weil die Funktionen Laufzeit- bzw. Speicherplatzschranken sein sollen, werden nur Funktionen $f(n)$ (bzw. g(n)) mit folgender Eigenschaft betrachtet: $\exists n_0 > 0\ \forall n > n_0 :\ f(n) > 0$

Asymptotische Notation



Asymptotische Notation

Beispiel

- $5n^2 - 7n,\ n^2/10 + 100n \in \mathcal{O}(n^2)$
- $n \log n \in \Omega(n),\ n^3 \in \Omega(n^2)$
- $\log n \in o(n),\ n^3 \in o(2^n)$
- $n^5 \in \omega(n^3),\ 2^{2n} \in \omega(2^n)$

Asymptotische Notation als Platzhalter

- statt $g(n) \in \mathcal{O}(f(n))$ schreibt man auch oft $g(n) = \mathcal{O}(f(n))$
- für $f(n) + g(n)$ mit $g(n) \in o(h(n))$ schreibt man auch $f(n) + g(n) = f(n) + o(h(n))$
- statt $\mathcal{O}(f(n)) \subseteq \mathcal{O}(g(n))$ schreibt man auch $\mathcal{O}(f(n)) = \mathcal{O}(g(n))$

Beispiel

$n^3 + n = n^3 + o(n^3) = (1 + o(1))n^3 = \mathcal{O}(n^3)$

$\mathcal{O}$-Notations"gleichungen" sollten nur von links nach rechts gelesen werden!

Wachstum von Polynomen

Lemma

Sei $p(n) = \sum_{i=0}^{k} a_i n^i$ *mit* $a_k > 0$.
Dann ist $p(n) \in \Theta(n^k)$.

Beweis.

Zu zeigen: $p(n) \in \mathcal{O}(n^k)$ und $p(n) \in \Omega(n^k)$

$p(n) \in \mathcal{O}(n^k)$:

Für $n > 1$ gilt:

$$p(n) \leq \sum_{i=0}^{k} |a_i| n^i \leq n^k \sum_{i=0}^{k} |a_i|$$

Also ist die Definition

$$\mathcal{O}(f(n)) = \{g(n) \mid \exists c > 0\ \exists n_0 > 0\ \forall n > n_0 : g(n) \leq c \cdot f(n)\}$$

Wachstum von Polynomen

Beweis.

$p(n) \in \Omega(n^k)$:

Sei $A = \sum_{i=0}^{k-1} |a_i|$.

Für positive n gilt dann:

$$p(n) \geq a_k n^k - A n^{k-1} = \frac{a_k}{2} n^k + n^{k-1} \left(\frac{a_k}{2} n - A \right)$$

Also ist die Definition

$$\Omega(f(n)) = \{g(n) \mid \exists c > 0 \; \exists n_0 > 0 \; \forall n > n_0 : g(n) \geq c \cdot f(n)\}$$

mit $c = a_k/2$ und $n_0 = 2A/a_k$ erfüllt. □