DHBW Lörrach
Duale Hochschule Baden-Württemberg

Algorithmen und Komplexität
TIF 21 A/B
Dr. Bruno Becker

3. Analyse von Algorithmen

Analyse von Algorithmen

- **Kosten eines Algorithmus**
- O-Notation
- Komplexitätsklassen

Kosten eines Algorithmus

- **Wie hängen die Kosten eines Algorithmus von der Problemgröße ab?**
- **Motivation:**
  - Wie groß kann meine Problemgröße sein, damit mein Algorithmus noch funktioniert?
  - Wieviel Platz benötigt der Algorithmus im schlimmsten Fall – Rechner-Ressourcen meistens nicht exklusiv
  - Ist die Antwortzeit eines Algorithmus für Anwender akzeptabel? – Beispiel Online-Ticketing
  - Ist mein Algorithmus und die verwendeten Datenstrukturen geschickt für die Problemlösung?

- **Modellierung der Kosten**
  - **Operationen zählen für Laufzeit-Bestimmung**
  - **Benötigter Speicherplatz**
  - **Modellierung heißt Vereinfachung**

Kosten eines Algorithmus

- **Vereinfachung in Formeln**
  - Nur die höchste Potenz zählt:
    $3 * n^3 + 42 * n^2 - 120 * n + 1345 \sim 3 * n^3$
  - $\mathbf{f\,(n) \sim g\,(n)} \Leftrightarrow \lim\limits_{n \to \infty} (f(n)/g(n)) = 1$
- **Vereinfachung in der Praxis manchmal unrealistisch**
  - $f(n) = 2 * n^3 + n - 1$
  - $g(n) = n^3 + 1000 * n^2$
  - $h(n) = 10^6 * n^2$
  - Welche Funktion nehme ich für n=1.000?; Ab wann lohnt sich h(n)?

Analyse von Algorithmen

- Kosten eines Algorithmus
- **O-Notation**
- Komplexitätsklassen

O-Notation: Vereinfachung zur Analyse

- **Ansatz: Konstante Faktoren ignorieren**
  - Konstanten werden von Hardware und Implementierung beeinflusst

- **O-Notation: Abschätzung nach oben**
- $f(n) \in O(g(n))$, wenn es positive Konstanten c und n_0 gibt, so dass gilt:

$$f(n) \leq c * g(n) \quad \text{für alle } n > n_0$$

  - $O(g(n))$ bezeichnet also eine *Menge* von Funktionen
  - **Asymptotische Notation**

O-Notation Beispiele

- $f(n) = 2 * n^3 + 100 * n^2 - 50 \ \epsilon$?
- $O(n) = O(1000 * n)$?
- $O(\log_2 n) = O(\log_2 n^2)$?
- $O(\log_2 n) > O(n^{1/10})$?

Übung

- Bringen Sie die folgenden Funktionen in eine Reihenfolge hinsichlich Ihrer Komplexität in O-Notation:

1. $f_1(n) = \sqrt{n/5} + 49$
2. $f_2(n) = n!$
3. $f_3(n) = log_2\, n^3$
4. $f_4\,(n) = 2^n$
5. $f_5\,(n) = 12n^2 - 7n + 5$
6. $f_6\,(n) = n^3 - 2n^2 + n$

Antwort $f_3 < f_1 < f_5 < f_6 < f_4 < f_2$

Ω und Θ-Notation

- **Ω -Notation: Abschätzung nach unten**
- $f(n) \in \Omega(g(n))$, wenn es positive Konstanten c und n_0 gibt, so dass gilt:

$$f(n) >= c * g(n) \quad \text{für alle } n > n_0$$

- **Θ -Notation: Abschätzung nach oben und unten**
- $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

Analyse von Algorithmen

- Kosten eines Algorithmus
- O-Notation
- **Komplexitätsklassen**

O-Notation in der Theorie der Algorithmen

→Komplexitätstheorie (gegen Ende der Vorlesung)

- **Effiziente (polynomiale) Algorithmen:** Laufzeit $O(n^k)$
- **Algorithmen mit exponentieller Worst case Laufzeit:** $\Omega(2^n)$
  - Im Worst case häufig Suche über alle Lösungsmöglichkeiten erforderlich
  - Beispiele: Travelling-Salesman-Problem, Erfüllbarkeit logischer Ausdrücke
- **Nachweisbar schwierige Probleme: Bewiesene untere Schranke Laufzeit:** $\Omega(2^n)$