# Algorithmen & Datenstrukturen

## Komplexitätsanalyse

**Wolfgang Auer**

# Komplexitätsanalyse

- Ziel
  - Möglichkeit zum Vergleich von verschiedenen Algorithmen, die dieselbe Aufgabe lösen
  - Bestimmung der Eigenschaften eines Algorithmus in Bezug auf:
    - Laufzeit
    - Speicherplatzbedarf
    - Statische Länge
    - ...

- Komplexität
  = Verhalten einer oder mehrerer Eigenschaften eines Algorithmus in Abhängigkeit von der Problemgröße

# Laufzeitkomplexität

- Laufzeit = *f (*Problemgröße*)*

  Problemgrößen
  - Listenlänge (Sortieren, Suchen)
  - Länge eines Textes (Suchen)
  - Anzahl der Elemente von Matrizen
  - Grad eines Polynoms
- Verschiedene Fragestellungen
  - Laufzeit eines Algorithmus
  - Minimale Laufzeit einer ganzen Problemklasse (Sortieren, Multiplizieren, ...)
- Bestimmung der Laufzeit
  - Messung
  - Berechnung

# Feinanalyse

- Bestimmung der Laufzeit eines Algorithmus anhand von gemessenen Ausführungszeiten elementarer Operationen
- Maschinen- und Compilerabhängige Werte

| **Operation** | **Zeit** |
|---|---|
| Zuweisung | 1 |
| Vergleich | 1.6 |
| Addition | 0.8 |
| Subtraktion | 0.8 |
| Division | 4.9 |
| Einfache Indizierung | 2.1 |
| ++ (Addition + Zuweisung) | 1.8 |
| Multiplikation | 2.3 |

# Beispiel: BinSearch (1)

```
BinSearch1(↓list, ↓n, ↓x, ↑y) {
   imin = 1; imax = n; y = 0;
   while ((y == 0) &&
          (imin <= imax)) {
      i = (imin + imax) / 2;
      if (x == list[i])
         y = i;
      else if (x < list[i])
         imax = i - 1;
      else
         imin = i + 1;
   }
}
```

| Anzahl | Dauer | Const | u | v |
|---|---|---|---|---|
| 1 | 3 | 3 | | |
| u + 1 (*) | 1.6 | 1.6 | 1.6u | |
| u | 1.6 | | 1.6u | |
| u | 6.7 | | 6.7u | |
| u | 3.7 | | 3.7u | |
| 1 (*) | 1 | 1 | | |
| u - 1 | 3.7 | -3.7 | 3.7u | |
| v | 1.8 | | | 1.8v |
| u - 1 - v | 1.8 | -1.8 | 1.8u | -1.8v |
| | | 0.1 | 19,1u | |

u..Anzahl der Schleifendurchläufe
v..Anzahl der Verzweigungen

(*) Annahme das gesuchte Element kommt vor

# Beispiel: BinSearch (2)

```
BinSearch2(↓list, ↓n, ↓x, ↑y) {
   imin = 1; imax = n;
   for (;;) {
      if (imin > imax) {
         y = 0;
         break;
      }
      i = (imin + imax) / 2
      if (x < list[i]) {
         imax = i - 1;
      else if (x > list[i])
         imin = i + 1;
      else {
         y = i;
         break;
      }
   } //end for
}
```

| Anzahl | Dauer | Const | u | v |
|---|---|---|---|---|
| 1 | 2 | 2 | | |
| u | 1.6 | | 1.6u | |
| 0 (*) | | | | |
| u | 6.7 | | 6.7u | |
| u | 3.7 | | 3.7u | |
| v | 1.8 | | | 1.8v |
| u - v | 3.7 | | 3.7u | -3.7v |
| u - v -1 | 1.8 | -1.8 | 1.8u | -1.8v |
| 1 (*) | 1 | 1 | | |
| | | 1.2 | 17.5u | -3.7v |

(*) Annahme das gesuchte Element kommt vor

# Gegenüberstellung

- *Annahme:* $v = \frac{u}{2}$

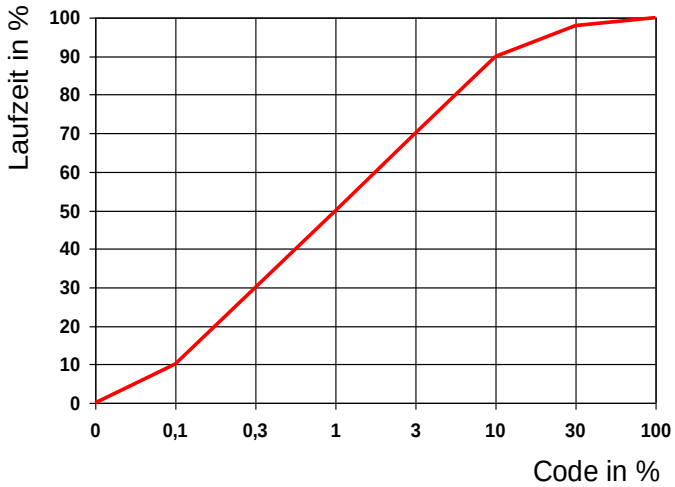
  d.h. die Wahrscheinlichkeit, dass das gesuchte Listenelement links oder rechts vom Prüfelement vorkommt ist gleich groß
- T(BinSearch1) = 19.1u + 0.1
- T(BinSearch2) = 15.65u + 1.2

| n | u | T(BinSearch1) | T(BinSearch2) |
|---|---|---|---|
| 10 | 4 | 76,7 | 63.8 |
| 100 | 7 | 134.0 | 110.75 |
| 1000 | 10 | 191.3 | 157.7 |
| 10000 | 14 | 267.7 | 220,3 |

# Zeitmessung

- Identische Fragestellung wie Feinanalyse
  - Wie oft werden bestimmte Programmzweige durchlaufen
  - Wie viel Zeit wird dafür benötigt

- *Profiling* dient hauptsächlich zum Auffinden des kritischen Pfads
- 1:50% Kurve



# Grobanalyse

- Weglassen der maschinen-abhängigen Faktoren
- Ausführungszeit von Anweisungen wird als konstant angesehen
- Bestimmen der Extremwerte und des Durchschnittswerts für die Laufzeit eines Algorithmus
  - best case
  - worst case
  - average case

# Beispiel

- Problem
  - Geg.: Int-Feld der Länge n (n ≥ 1), das nur die Werte „1“, „2“ und „3“ aufnehmen kann
  - Ges: Die Position des linkesten größten Elements
- Lösung

```
PositionOfMax(↓n, ↓f, ↑pos) { //f[1..n]
    pos = 1;
    i = 2;
    max = f[pos];

    while (i <= n) && (max < 3) {
        if (f[i] > max) {
           max = f[i];
           pos = i;
       }
       i++;
    }
}
```

Anzahl der Schleifendurchläufe *u*

- minimal: u = 0, wenn f[1] = 3
- maximal: u = n - 1, wenn f[n] = 3 $\vee$ 3 $\notin$ f
- durchschnittlich: $u = 2 - 3\left(\frac{2}{3}\right)^n$

# PositionOfMax

```
PositionOfMax(↓n, ↓f, ↑pos) { // f[1..n]
  pos = 1; i = 2;
  max = f[pos];

  while (i <= n) && (max < 3) {
     if (f[i] > max) {
        max = f[i];
        pos = i;
     }

     i++;
  }
}
```

# Asymptotische Komplexität

- Ziel
  Angabe der Größenordnung der Zeitkomplexität eines Algorithmus in Abhängigkeit von der Eingabe

- Idee
  Abstraktion von Konstanten und Potenzen niederer Ordnung
  z.B.: $O(an^2 + bn + c) = O(n^2)$

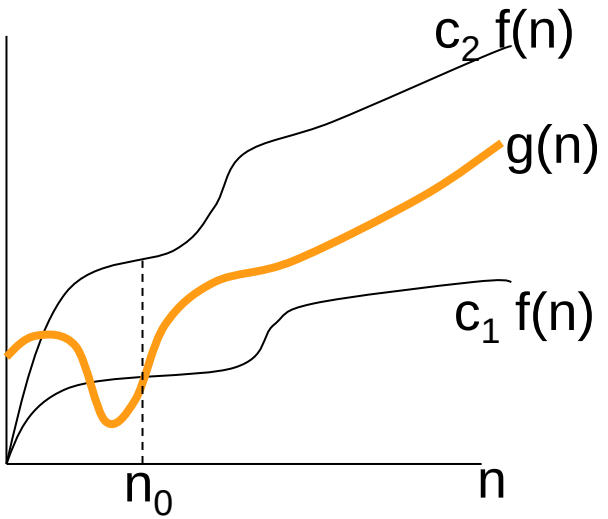
- Betrachtung der durchschnittlichen Laufzeit wird vernachlässigt
  - Bestimmung oft mathematisch kompliziert
  - Oftmals realistische Abschätzung der Eingabedaten schwierig

# Θ-Notation

- Betrachtung der Laufzeit in Bezug auf eine untere und obere Schranke

$$g(n) = \Theta(f(n))$$

$$\Theta(f(n)) = \left\{ \begin{array}{l} g(n) : \exists c_1, c_2, n_0 > 0 \\ \qquad \ni 0 \le c_1 f(n) \le g(n) \le c_2 f(n), \forall n \ge n_0 \end{array} \right\}$$




$$g(n) = \Theta(f(n))$$

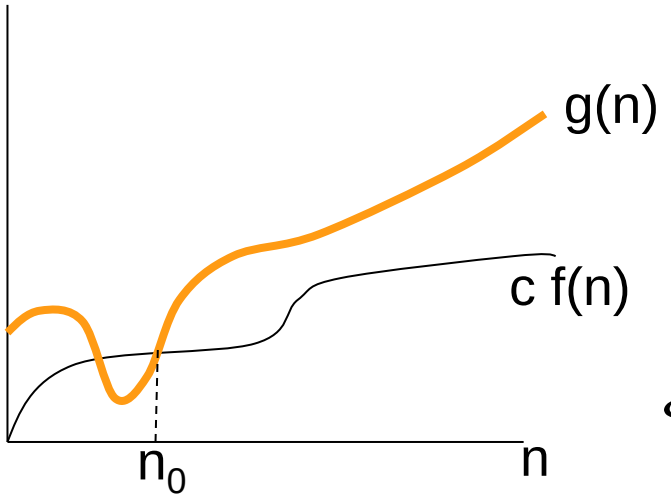
# Ω-Notation

- Betrachtung der Laufzeit nur in Bezug auf eine untere Schranke

$$\Omega(f(n)) \supseteq \Theta(f(n))$$

$$g(n) = \Omega(f(n))$$

$$\Omega(f(n)) = \left\{ \begin{array}{l} g(n) : \exists c, n_0 > 0 \\ \quad \ni c\, f(n) \leq g(n), \forall n \geq n_0 \end{array} \right\}$$




$$g(n) = \Omega(f(n))$$

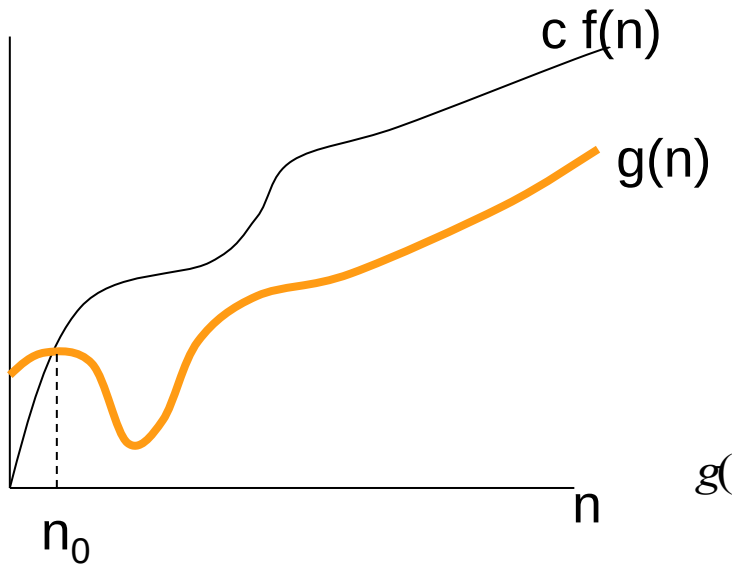
# O-Notation

- Betrachtung der Laufzeit nur in Bezug auf eine obere Schranke

$$O(f(n)) \supseteq \Theta(f(n))$$

$$g(n) = O(f(n))$$

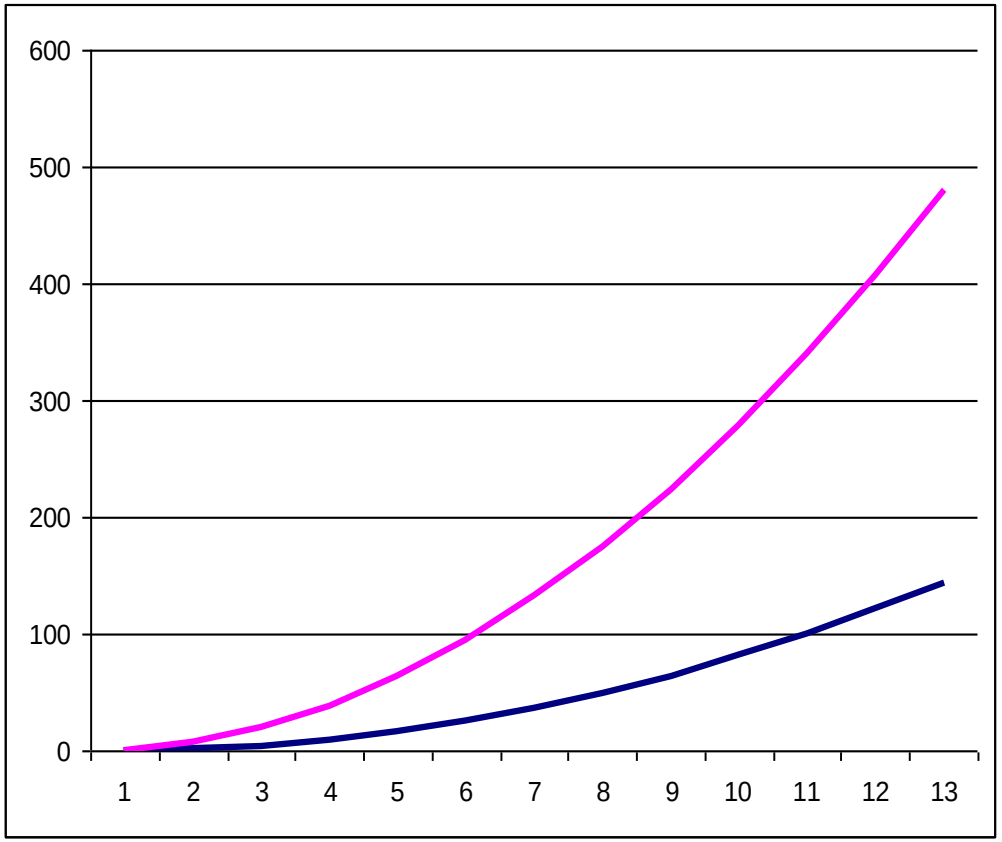
$$O(f(n)) = \left\{ \begin{array}{l} g(n) : \exists c, n_0 > 0 \\ \ni 0 \le g(n) \le c\, f(n), \forall n \ge n_0 \end{array} \right\}$$




$$g(n) = O(f(n)) \Leftrightarrow \lim_{n \to \infty} \left| \frac{g(n)}{f(n)} \right| \le c$$

# Dominanter Term und Koeffizienten

| n | $n^2$ | $3n^2+4n$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 7 |
| 2 | 4 | 20 |
| 3 | 9 | 39 |
| 4 | 16 | 64 |
| 5 | 25 | 95 |



# Einfluss des dominanten Terms

$$f(n) = \frac{3n^2 + 3n}{2}$$

| n | Formel | $\frac{3n^2}{2}$ | Anteil |
|---|---|---|---|
| 1 | 3 | 1,5 | 50,00% |
| 10 | 165 | 150 | 90,91% |
| 100 | 15150 | 15000 | 99,01% |
| 1000 | 1501500 | 1500000 | **99,90%** |

Dominanter Term ohne Koeffizienten

| n | Formel | $n^2$ | Anteil |
|---|---|---|---|
| 1 | 2,5 | 1 | 40,00% |
| 10 | 115 | 100 | 86,96% |
| 100 | 10150 | 10000 | 98,52% |
| 1000 | 1001500 | 1000000 | **99,85%** |

# Komplexitätsklassen

| | | |
|---|---|---|
| O(1) | Ideal | |
| O(lg n) | Sehr gut | Erstrebenswert |
| O(n) | Zufrieden stellend | |
| $O(n^2)$ | Unerwünscht | |
| $O(a^{n)}$ | Katastrophal | |