Informatik II

Vorlesung am D-BAUG der ETH Zürich

Felix Friedrich & Hermann Lehner

FS 2018

1. Einführung

Algorithmen und Datenstrukturen, erstes Beispiel

Ziele der Vorlesung

- Verständnis des Entwurfs und der Analyse grundlegender Algorithmen und Datenstrukturen.
- Grundlagen für das Design und die Implementation von Datenbanken.

Inhalte der Vorlesung

Datenstrukturen / Algorithmen

Begriff der Invariante, Kostenmodell, Landau Symbole

Algorithmenentwurf, Induktion

Suchen und Auswahl, Sortieren

Wörterbücher: Hashing und Suchbäume, AVL

Dynamic Programming

Graphen, Kürzeste Wege, Backtracking, Maximaler Fluss

Software Engineering

Files and Exceptions

Java Streams API

Datenbanken

ER-Modell, Relationales Modell, SQL

1.1 Algorithmen

[Cormen et al, Kap. 1;Ottman/Widmayer, Kap. 1.1]

Algorithmus

Algorithmus: wohldefinierte Berechnungsvorschrift, welche aus Eingabedaten (*input*) Ausgabedaten (*output*) berechnet.

Beispielproblem

Input : Eine Folge von n Zahlen $(a_1, a_2, \ldots, a_n)$

Beispielproblem

Input : Eine Folge von n Zahlen $(a_1, a_2, \ldots, a_n)$

Output : Eine Permutation $(a'_1, a'_2, \ldots, a'_n)$ der Folge $(a_i)_{1 \leq i \leq n}$, so dass $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Beispielproblem

Input : Eine Folge von n Zahlen $(a_1, a_2, \ldots, a_n)$
Output : Eine Permutation $(a'_1, a'_2, \ldots, a'_n)$ der Folge $(a_i)_{1 \leq i \leq n}$, so dass $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Mögliche Eingaben

$(1, 7, 3)$, $(15, 13, 12, -0.5)$, (1) ...

Beispielproblem

Input : Eine Folge von n Zahlen $(a_1, a_2, \ldots, a_n)$
Output : Eine Permutation $(a'_1, a'_2, \ldots, a'_n)$ der Folge $(a_i)_{1 \leq i \leq n}$, so dass $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Mögliche Eingaben

$(1, 7, 3)$, $(15, 13, 12, -0.5)$, (1) …

Jedes Beispiel erzeugt eine *Probleminstanz*.

Die Performanz (Geschwindigkeit) des Algorithmus hängt üblicherweise ab von der Probleminstanz. Es gibt oft “gute” und “schlechte” Instanzen.

Beispiele für Probleme in der Algorithmik

- Tabellen und Statistiken: Suchen, Auswählen und Sortieren

Beispiele für Probleme in der Algorithmik

- Tabellen und Statistiken: Suchen, Auswählen und Sortieren
- Routenplanung: Kürzeste Wege Algorithmus, Heap Datenstruktur

Beispiele für Probleme in der Algorithmik

- Tabellen und Statistiken: Suchen, Auswählen und Sortieren
- Routenplanung: Kürzeste Wege Algorithmus, Heap Datenstruktur
- DNA Matching: Dynamic Programming

Beispiele für Probleme in der Algorithmik

- Tabellen und Statistiken: Suchen, Auswählen und Sortieren
- Routenplanung: Kürzeste Wege Algorithmus, Heap Datenstruktur
- DNA Matching: Dynamic Programming
- Fabrikationspipeline: Topologische Sortierung

Beispiele für Probleme in der Algorithmik

- Tabellen und Statistiken: Suchen, Auswählen und Sortieren
- Routenplanung: Kürzeste Wege Algorithmus, Heap Datenstruktur
- DNA Matching: Dynamic Programming
- Fabrikationspipeline: Topologische Sortierung
- Autovervollständigung: Wörterbücher/Bäume

Beispiele für Probleme in der Algorithmik

- Tabellen und Statistiken: Suchen, Auswählen und Sortieren
- Routenplanung: Kürzeste Wege Algorithmus, Heap Datenstruktur
- DNA Matching: Dynamic Programming
- Fabrikationspipeline: Topologische Sortierung
- Autovervollständigung: Wörterbücher/Bäume
- Symboltabellen: Hash-Tabellen

Beispiele für Probleme in der Algorithmik

- Tabellen und Statistiken: Suchen, Auswählen und Sortieren
- Routenplanung: Kürzeste Wege Algorithmus, Heap Datenstruktur
- DNA Matching: Dynamic Programming
- Fabrikationspipeline: Topologische Sortierung
- Autovervollständigung: Wörterbücher/Bäume
- Symboltabellen: Hash-Tabellen
- Der Handlungsreisende: Dynamische Programmierung, Minimal aufspannender Baum, Simulated Annealing,

Beispiele für Probleme in der Algorithmik

- Tabellen und Statistiken: Suchen, Auswählen und Sortieren
- Routenplanung: Kürzeste Wege Algorithmus, Heap Datenstruktur
- DNA Matching: Dynamic Programming
- Fabrikationspipeline: Topologische Sortierung
- Autovervollständigung: Wörterbücher/Bäume
- Symboltabellen: Hash-Tabellen
- Der Handlungsreisende: Dynamische Programmierung, Minimal aufspannender Baum, Simulated Annealing,
- Zeichnen am Computer: Linien und Kreise Digitalisieren, Füllen von Polygonen

Beispiele für Probleme in der Algorithmik

- Tabellen und Statistiken: Suchen, Auswählen und Sortieren
- Routenplanung: Kürzeste Wege Algorithmus, Heap Datenstruktur
- DNA Matching: Dynamic Programming
- Fabrikationspipeline: Topologische Sortierung
- Autovervollständigung: Wörterbücher/Bäume
- Symboltabellen: Hash-Tabellen
- Der Handlungsreisende: Dynamische Programmierung, Minimal aufspannender Baum, Simulated Annealing,
- Zeichnen am Computer: Linien und Kreise Digitalisieren, Füllen von Polygonen
- PageRank: (Markov-Chain) Monte Carlo ...

Charakteristik

- Extrem grosse Anzahl potentieller Lösungen
- Praktische Anwendung

Datenstrukturen

- Organisation der Daten, zugeschnitten auf die Algorithmen die auf den Daten operieren
- Programme = Algorithmen + Datenstrukturen.

Ein Traum

- Wären Rechner unendlich schnell und hätten unendlich viel Speicher ...
- ... dann bräuchten wir die Theorie der Algorithmen (nur) für Aussagen über Korrektheit (incl. Terminierung).

Die Realität

Ressourcen sind beschränkt und nicht umsonst:

- Rechenzeit $\rightarrow$ Effizienz
- Speicherplatz $\rightarrow$ Effizienz

1.2 Altägyptische Multiplikation

Altägyptische Multiplikation

Altägyptische Multiplikation[2]

Berechnung von $11 \cdot 9$

$$\begin{array}{r|l} 11 & \;9 \end{array} \qquad\qquad \begin{array}{r|l} 9 & 11 \end{array}$$

[2] Auch bekannt als Russiche Bauernmulltiplikation

Altägyptische Multiplikation[2]

Berechnung von $11 \cdot 9$

$$11 \mid 9 \qquad\qquad 9 \mid 11$$

1. Links verdoppeln, rechts ganzzahlig halbieren.

[2] Auch bekannt als Russiche Bauernmulltiplikation

Altägyptische Multiplikation[2]

Berechnung von $11 \cdot 9$

$$\begin{array}{r|r} 11 & 9 \\ 22 & 4 \end{array} \qquad \begin{array}{r|r} 9 & 11 \\ 18 & 5 \end{array}$$

1. Links verdoppeln, rechts ganzzahlig halbieren.

[2] Auch bekannt als Russiche Bauernmulltiplikation

Altägyptische Multiplikation[2]

Berechnung von $11 \cdot 9$

$$\begin{array}{r|l} 11 & 9 \\ 22 & 4 \\ 44 & 2 \end{array} \qquad \begin{array}{r|r} 9 & 11 \\ 18 & 5 \\ 36 & 2 \end{array}$$

1. Links verdoppeln, rechts ganzzahlig halbieren.

[2] Auch bekannt als Russiche Bauernmulltiplikation

Altägyptische Multiplikation[2]

Berechnung von $11 \cdot 9$

$$
\begin{array}{r|l}
11 & 9 \\
22 & 4 \\
44 & 2 \\
88 & 1
\end{array}
\qquad\qquad
\begin{array}{r|r}
9 & 11 \\
18 & 5 \\
36 & 2 \\
72 & 1
\end{array}
$$

1. Links verdoppeln, rechts ganzzahlig halbieren.
2. Gerade Zahl rechts $\Rightarrow$ Zeile streichen.

[2] Auch bekannt als Russiche Bauernmulltiplikation

Altägyptische Multiplikation[2]

Berechnung von $11 \cdot 9$

11	9
~~22~~	~~4~~
~~44~~	~~2~~
88	1

9	11
18	5
~~36~~	~~2~~
72	1

1. Links verdoppeln, rechts ganzzahlig halbieren.
2. Gerade Zahl rechts $\Rightarrow$ Zeile streichen.

[2] Auch bekannt als Russiche Bauernmulltiplikation

Altägyptische Multiplikation[2]

Berechnung von $11 \cdot 9$

$$\begin{array}{r|r} 11 & 9 \\ \cancel{22} & \cancel{4} \\ \cancel{44} & \cancel{2} \\ 88 & 1 \end{array} \qquad \begin{array}{r|r} 9 & 11 \\ 18 & 5 \\ \cancel{36} & \cancel{2} \\ 72 & 1 \end{array}$$

1. Links verdoppeln, rechts ganzzahlig halbieren.
2. Gerade Zahl rechts $\Rightarrow$ Zeile streichen.
3. Übrige Zeilen links addieren.

[2] Auch bekannt als Russiche Bauernmulltiplikation

Altägyptische Multiplikation[2]

Berechnung von $11 \cdot 9$

11	9
~~22~~	~~4~~
~~44~~	~~2~~
88	1
99	–

9	11
18	5
~~36~~	~~2~~
72	1
99	

1. Links verdoppeln, rechts ganzzahlig halbieren.
2. Gerade Zahl rechts $\Rightarrow$ Zeile streichen.
3. Übrige Zeilen links addieren.

[2] Auch bekannt als Russiche Bauernmulltiplikation

Vorteile

- Kurze Beschreibung, einfach zu verstehen.
- Effizient für Computer im Dualsystem: Verdoppeln = Left Shift, Halbieren = Right Shift

Beispiel

left shift $9 = 01001_2 \to 10010_2 = 18$

right shift $9 = 01001_2 \to 00100_2 = 4$

Fragen

- Funktioniert das immer? (z.B. für negative Zahlen)
- Wenn nicht, wann?
- Wie beweist man die Korrektheit?
- Besser als die "Schulmethode"?
- Was heisst "gut"? Lässt sich Güte anordnen?
- Wie schreibt man das Verfahren unmissverständlich auf?

Beobachtung

Wenn $b > 1$, $a \in \mathbb{Z}$, dann:

$$a \cdot b = \begin{cases} 2a \cdot \frac{b}{2} & \text{falls } b \text{ gerade,} \\ a + 2a \cdot \frac{b-1}{2} & \text{falls } b \text{ ungerade.} \end{cases}$$

Terminierung

$$a \cdot b = \begin{cases} a & \text{falls } b = 1, \\ 2a \cdot \frac{b}{2} & \text{falls } b \text{ gerade}, \\ a + 2a \cdot \frac{b-1}{2} & \text{falls } b \text{ ungerade}. \end{cases}$$

Rekursiv funktional notiert

$$f(a,b) = \begin{cases} a & \text{falls } b = 1, \\ f(2a, \frac{b}{2}) & \text{falls } b \text{ gerade}, \\ a + f(2a, \frac{b-1}{2}) & \text{falls } b \text{ ungerade}. \end{cases}$$

Funktion programmiert

```
// pre: b>0
// post: return a*b
int f(int a, int b){
    if(b==1)
        return a;
    else if (b%2 == 0)
        return f(2*a, b/2);
    else
        return a + f(2*a, (b-1)/2);
}
```

Korrektheit

$$f(a,b) = \begin{cases} a & \text{falls } b = 1, \\ f(2a, \frac{b}{2}) & \text{falls } b \text{ gerade}, \\ a + f(2a \cdot \frac{b-1}{2}) & \text{falls } b \text{ ungerade}. \end{cases}$$

Zu zeigen: $f(a,b) = a \cdot b$ für $a \in \mathbb{Z}$, $b \in \mathbb{N}^+$.

Beweis per Induktion

Anfang: $b = 1 \Rightarrow f(a,b) = a = a \cdot 1$.
Hypothese: $f(a,b') = a \cdot b'$ für $0 < b' \leq b$
Schritt: $f(a,b+1) \overset{!}{=} a \cdot (b+1)$

$$f(a,b+1) = \begin{cases} f(2a, \overbrace{\frac{b+1}{2}}^{\leq b}) = a \cdot (b+1) & \text{falls } b \text{ ungerade,} \\ a + f(2a, \underbrace{\frac{b}{2}}_{\leq b}) = a + a \cdot b & \text{falls } b \text{ gerade.} \end{cases}$$

$\blacksquare$

Rekursion vs. Iteration

```
// pre: b>0
// post: return a*b
int f(int a, int b){
  if(b==1)
    return a;
  else if (b%2 == 0)
    return f(2*a, b/2);
  else
    return a + f(2*a, (b-1)/2);
}
```

→

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Invarianten!

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Sei $x := a \cdot b$.

Invarianten!

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Sei $x := a \cdot b$.

Hier gilt $x = \boxed{a \cdot b + res}$

Invarianten!

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Sei $x := a \cdot b$.

Hier gilt $x = \boxed{a \cdot b + res}$

Wenn hier $x = a \cdot b + res$ ...

Invarianten!

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Sei $x := a \cdot b$.

Hier gilt $x = \boxed{a \cdot b + res}$

Wenn hier $x = a \cdot b + res$ ...

... dann auch hier $x = a \cdot b + res$

Invarianten!

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Sei $x := a \cdot b$.

Hier gilt $x = \boxed{a \cdot b + res}$

Wenn hier $x = a \cdot b + res$ ...

... dann auch hier $x = a \cdot b + res$
b gerade

Invarianten!

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Sei $x := a \cdot b$.

Hier gilt $x = \boxed{a \cdot b + res}$

Wenn hier $x = a \cdot b + res$ ...

... dann auch hier $x = a \cdot b + res$

b gerade

Hier gilt $x = a \cdot b + res$

Invarianten!

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Sei $x := a \cdot b$.

Hier gilt $x = \boxed{a \cdot b + res}$

Wenn hier $x = a \cdot b + res$ ...

... dann auch hier $x = a \cdot b + res$
b gerade

Hier gilt $x = a \cdot b + res$
Hier gilt $x = a \cdot b + res$ und $b = 0$

Invarianten!

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Sei $x := a \cdot b$.

Hier gilt $x = \boxed{a \cdot b + res}$

Wenn hier $x = a \cdot b + res$ ...

... dann auch hier $x = a \cdot b + res$

b gerade

Hier gilt $x = a \cdot b + res$

Hier gilt $x = a \cdot b + res$ und $b = 0$

Also $res = x$.

Zusammenfassung

Der Ausdruck $a \cdot b + res$ ist eine *Invariante*.

- Werte von a, b, res ändern sich, aber die Invariante bleibt ”im Wesentlichen“ unverändert:
- Invariante vorübergehend durch eine Anweisung zerstört, aber dann darauf wieder hergestellt.
- Betrachtet man solche Aktionsfolgen als atomar, bleibt der Wert tatsächlich invariant
- Insbesondere erhält die Schleife die Invariante (*Schleifeninvariante*), wirkt dort wie der Induktionsschritt bei der vollständigen Induktion
- Invarianten sind offenbar mächtige Beweishilfsmittel!

Analyse

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Altägyptische Multiplikation entspricht der Schulmethode zur Basis 2.

$$1 \quad 0 \quad 0 \quad 1 \quad \times \quad 1 \quad 0 \quad 1 \quad 1$$

Analyse

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Altägyptische Multiplikation entspricht der Schulmethode zur Basis 2.

$$
\begin{array}{ccccccccccc}
1 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 & & \\
\hline
 & & & & & 1 & 0 & 0 & 1 & & (9)
\end{array}
$$

Analyse

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Altägyptische Multiplikation entspricht der Schulmethode zur Basis 2.

$$
\begin{array}{cccccccccccc}
1 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 & & \\
\hline
 & & & & & 1 & 0 & 0 & 1 & & (9) \\
 & & & & 1 & 0 & 0 & 1 & & & (18)
\end{array}
$$

Analyse

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Altägyptische Multiplikation entspricht der Schulmethode zur Basis 2.

$$
\begin{array}{cccccccccccc}
1 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 & \\
\hline
 & & & & & 1 & 0 & 0 & 1 & (9) \\
 & & & & 1 & 0 & 0 & 1 & & (18) \\
\hline
 & & & & 1 & 1 & 0 & 1 & 1 &
\end{array}
$$

Analyse

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Altägyptische Multiplikation entspricht der Schulmethode zur Basis 2.

$$
\begin{array}{cccccccccc}
1 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 & \\
\hline
 & & & & & 1 & 0 & 0 & 1 & (9) \\
 & & & & 1 & 0 & 0 & 1 & & (18) \\
\hline
 & & & & 1 & 1 & 0 & 1 & 1 & \\
 & & 1 & 0 & 0 & 1 & & & & (72)
\end{array}
$$

Analyse

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
  int res = 0;
  while (b > 0) {
    if (b % 2 != 0){
      res += a;
      --b;
    }
    a *= 2;
    b /= 2;
  }
  return res;
}
```

Altägyptische Multiplikation entspricht der Schulmethode zur Basis 2.

$$
\begin{array}{cccccccccccc}
1 & 0 & 0 & 1 & \times & 1 & 0 & 1 & 1 & & & \\
\hline
 & & & & & 1 & 0 & 0 & 1 & & & (9) \\
 & & & & 1 & 0 & 0 & 1 & & & & (18) \\
\hline
 & & & & 1 & 1 & 0 & 1 & 1 & & & \\
 & & 1 & 0 & 0 & 1 & & & & & & (72) \\
\hline
 & & 1 & 1 & 0 & 0 & 0 & 1 & 1 & & & (99)
\end{array}
$$

Effizienz

Frage: Wie lange dauert eine Multiplikation von a und b?

- Mass für die Effizienz
  - Gesamtzahl der elementaren Operationen: Verdoppeln, Halbieren, Test auf "gerade", Addition
  - Im rekursiven wie im iterativen Code: maximal 6 Operationen pro Aufruf bzw. Durchlauf
- Wesentliches Kriterium:
  - Anzahl rekursiver Aufrufe oder
  - Anzahl Schleifendurchläufe(im iterativen Fall)
- $\frac{b}{2^n} \leq 1$ gilt für $n \geq \log_2 b$. Also nicht mehr als $6\lceil \log_2 b \rceil$ elementare Operationen.

2. Effizienz von Algorithmen

Effizienz von Algorithmen, Random Access Machine Modell, Funktionenwachstum, Asymptotik [Cormen et al, Kap. 2.2,3,4.2-4.4 | Ottman/Widmayer, Kap. 1.1]

Effizienz von Algorithmen

Ziele

- Laufzeitverhalten eines Algorithmus maschinenunabhängig quantifizieren.
- Effizienz von Algorithmen vergleichen.
- Abhängigkeit von der Eingabegrösse verstehen.

Technologiemodell

Random Access Machine (RAM)

- Ausführungsmodell: Instruktionen werden der Reihe nach (auf einem Prozessorkern) ausgeführt.

Technologiemodell

Random Access Machine (RAM)

- Ausführungsmodell: Instruktionen werden der Reihe nach (auf einem Prozessorkern) ausgeführt.
- Speichermodell: Konstante Zugriffszeit.

Technologiemodell

Random Access Machine (RAM)

- Ausführungsmodell: Instruktionen werden der Reihe nach (auf einem Prozessorkern) ausgeführt.
- Speichermodell: Konstante Zugriffszeit.
- Elementare Operationen: Rechenoperation ($+,-,\cdot,\ldots$) , Vergleichsoperationen, Zuweisung / Kopieroperation, Flusskontrolle (Sprünge)

Technologiemodell

Random Access Machine (RAM)

- Ausführungsmodell: Instruktionen werden der Reihe nach (auf einem Prozessorkern) ausgeführt.
- Speichermodell: Konstante Zugriffszeit.
- Elementare Operationen: Rechenoperation ($+$,$-$,$\cdot$,...) , Vergleichsoperationen, Zuweisung / Kopieroperation, Flusskontrolle (Sprünge)
- Einheitskostenmodell: elementare Operation hat Kosten 1.

Technologiemodell

Random Access Machine (RAM)

- Ausführungsmodell: Instruktionen werden der Reihe nach (auf einem Prozessorkern) ausgeführt.
- Speichermodell: Konstante Zugriffszeit.
- Elementare Operationen: Rechenoperation ($+$,$-$,$\cdot$,...) , Vergleichsoperationen, Zuweisung / Kopieroperation, Flusskontrolle (Sprünge)
- Einheitskostenmodell: elementare Operation hat Kosten 1.
- Datentypen: Fundamentaltypen wie grössenbeschränkte Ganzzahl oder Fliesskommazahl.

Asymptotisches Verhalten

Genaue Laufzeit lässt sich selbst für kleine Eingabedaten kaum voraussagen.

- Betrachten das asymptotische Verhalten eines Algorithmus.
- Ignorieren alle konstanten Faktoren.

Beispiel

Eine Operation mit Kosten 20 ist genauso gut wie eine mit Kosten 1.
Lineares Wachstum mit Steigung 5 ist genauso gut wie lineares Wachstum mit Steigung 1.

2.2 Funktionenwachstum

$\mathcal{O}$, Θ, Ω [Cormen et al, Kap. 3; Ottman/Widmayer, Kap. 1.1]

Oberflächlich

Verwende die asymptotische Notation zur Kennzeichnung der Laufzeit von Algorithmen

Wir schreiben $\Theta(n^2)$ und meinen, dass der Algorithmus sich für grosse n wie n^2 verhält: verdoppelt sich die Problemgrösse, so vervierfacht sich die Laufzeit.

Genauer: Asymptotische obere Schranke

Gegeben: Funktion $g : \mathbb{N} \to \mathbb{R}$.

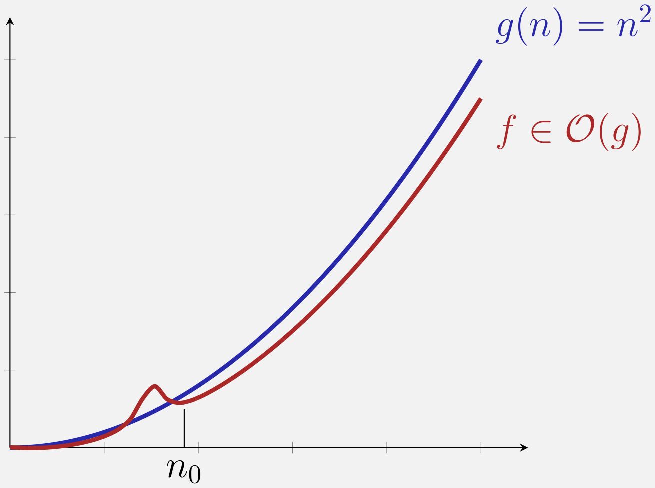
Definition:

$$\mathcal{O}(g) = \{f : \mathbb{N} \to \mathbb{R} | \\ \exists c > 0, n_0 \in \mathbb{N} : 0 \leq f(n) \leq c \cdot g(n) \; \forall n \geq n_0\}$$

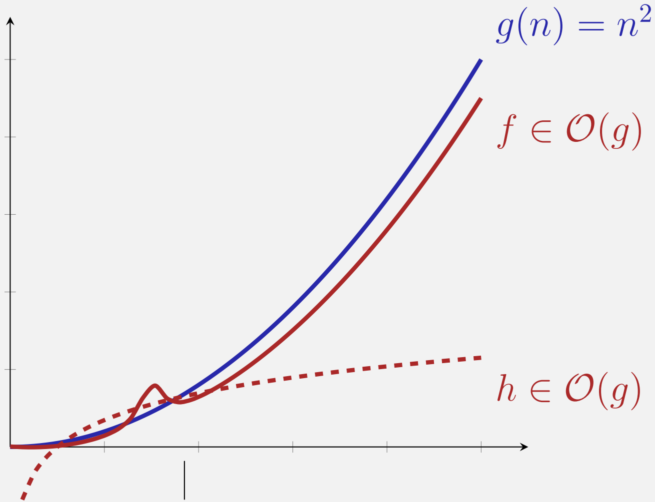
Schreibweise:

$$\mathcal{O}(g(n)) := \mathcal{O}(g(\cdot)) = \mathcal{O}(g).$$

Anschauung



Anschauung



Beispiele

$$\mathcal{O}(g) = \{f : \mathbb{N} \to \mathbb{R} | \;\; \exists c > 0, n_0 \in \mathbb{N} : 0 \leq f(n) \leq c \cdot g(n) \; \forall n \geq n_0\}$$

$f(n)$	$f \in \mathcal{O}(?)$	Beispiel
$3n + 4$		
$2n$		
$n^2 + 100n$		
$n + \sqrt{n}$		

Beispiele

$$\mathcal{O}(g) = \{f : \mathbb{N} \to \mathbb{R} \mid \ \exists c > 0, n_0 \in \mathbb{N} : 0 \leq f(n) \leq c \cdot g(n) \ \forall n \geq n_0\}$$

$f(n)$	$f \in \mathcal{O}(?)$	Beispiel
$3n+4$	$\mathcal{O}(n)$	$c=4, n_0=4$
$2n$		
n^2+100n		
$n+\sqrt{n}$		

Beispiele

$$\mathcal{O}(g) = \{f : \mathbb{N} \to \mathbb{R} | \;\; \exists c > 0, n_0 \in \mathbb{N} : 0 \leq f(n) \leq c \cdot g(n) \; \forall n \geq n_0\}$$

$f(n)$	$f \in \mathcal{O}(?)$	Beispiel
$3n + 4$	$\mathcal{O}(n)$	$c = 4, n_0 = 4$
$2n$	$\mathcal{O}(n)$	$c = 2, n_0 = 0$
$n^2 + 100n$		
$n + \sqrt{n}$		

Beispiele

$$\mathcal{O}(g) = \{f : \mathbb{N} \to \mathbb{R} | \quad \exists c > 0, n_0 \in \mathbb{N} : 0 \leq f(n) \leq c \cdot g(n) \; \forall n \geq n_0\}$$

$f(n)$	$f \in \mathcal{O}(?)$	Beispiel
$3n + 4$	$\mathcal{O}(n)$	$c = 4, n_0 = 4$
$2n$	$\mathcal{O}(n)$	$c = 2, n_0 = 0$
$n^2 + 100n$	$\mathcal{O}(n^2)$	$c = 2, n_0 = 100$
$n + \sqrt{n}$		

Beispiele

$$\mathcal{O}(g) = \{f : \mathbb{N} \to \mathbb{R} |\ \ \exists c > 0, n_0 \in \mathbb{N} : 0 \leq f(n) \leq c \cdot g(n)\ \forall n \geq n_0\}$$

$f(n)$	$f \in \mathcal{O}(?)$	Beispiel
$3n + 4$	$\mathcal{O}(n)$	$c = 4, n_0 = 4$
$2n$	$\mathcal{O}(n)$	$c = 2, n_0 = 0$
$n^2 + 100n$	$\mathcal{O}(n^2)$	$c = 2, n_0 = 100$
$n + \sqrt{n}$	$\mathcal{O}(n)$	$c = 2, n_0 = 1$

Eigenschaft

$$f_1 \in \mathcal{O}(g), f_2 \in \mathcal{O}(g) \Rightarrow f_1 + f_2 \in \mathcal{O}(g)$$

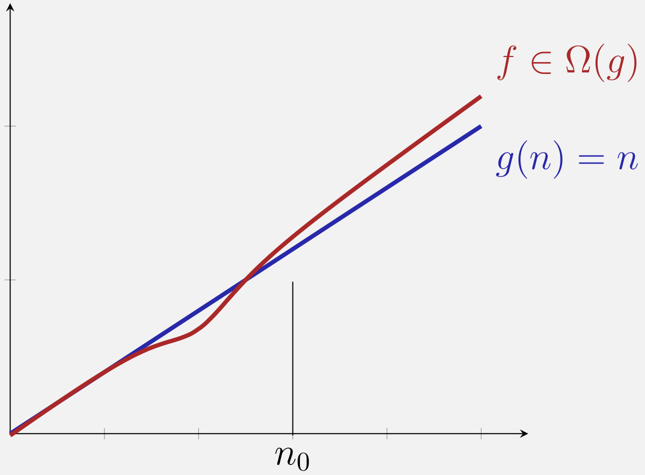
Umkehrung: Asymptotische untere Schranke

Gegeben: Funktion $g : \mathbb{N} \to \mathbb{R}$.

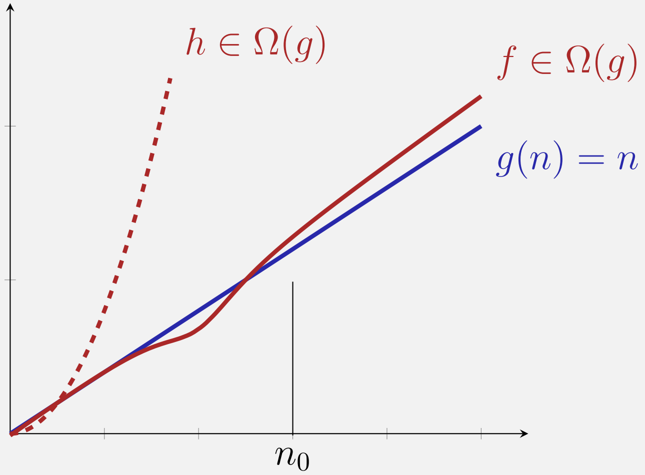
Definition:

$$\Omega(g) = \{f : \mathbb{N} \to \mathbb{R} | \\ \exists c > 0, n_0 \in \mathbb{N} : 0 \leq c \cdot g(n) \leq f(n) \; \forall n \geq n_0\}$$

Beispiel



Beispiel



Asymptotisch scharfe Schranke

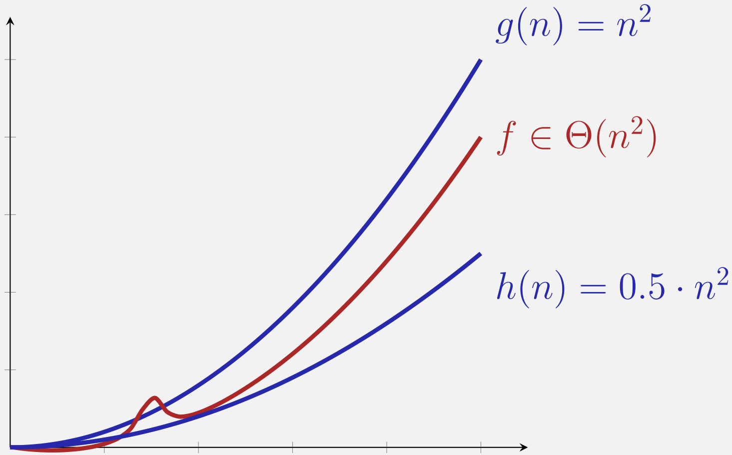
Gegeben Funktion $g : \mathbb{N} \to \mathbb{R}$.

Definition:

$$\Theta(g) := \Omega(g) \cap \mathcal{O}(g).$$

Einfache, geschlossene Form: Übung.

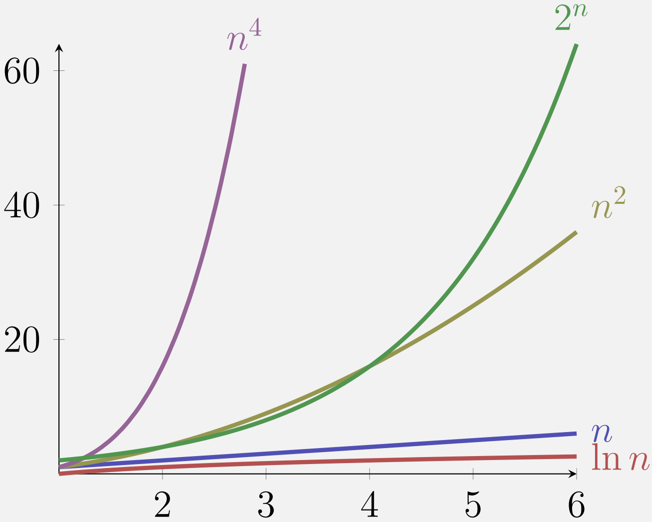
Beispiel



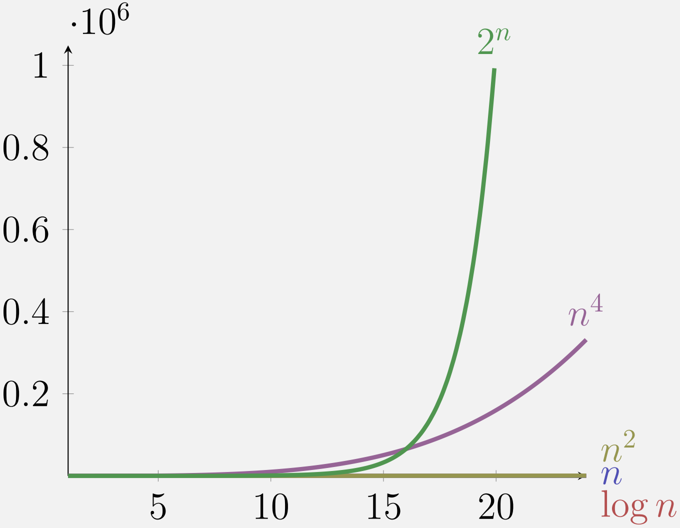
Wachstumsbezeichnungen

$\mathcal{O}(1)$	beschränkt	Array-Zugriff
$\mathcal{O}(\log\log n)$	doppelt logarithmisch	Binäre sortierte Suche interpoliert
$\mathcal{O}(\log n)$	logarithmisch	Binäre sortierte Suche
$\mathcal{O}(\sqrt{n})$	wie die Wurzelfunktion	Primzahltest (naiv)
$\mathcal{O}(n)$	linear	Unsortierte naive Suche
$\mathcal{O}(n\log n)$	superlinear / loglinear	Gute Sortieralgorithmen
$\mathcal{O}(n^2)$	quadratisch	Einfache Sortieralgorithmen
$\mathcal{O}(n^c)$	polynomial	Matrixmultiplikation
$\mathcal{O}(2^n)$	exponentiell	Travelling Salesman Dynamic Programming
$\mathcal{O}(n!)$	faktoriell	Travelling Salesman naiv

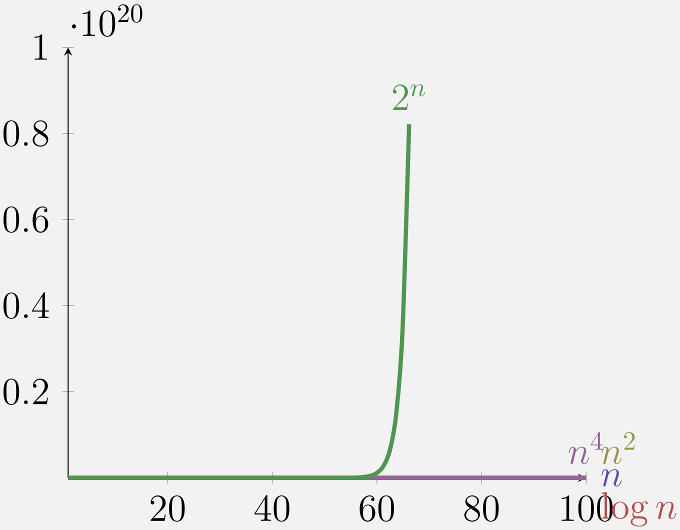
Kleine n



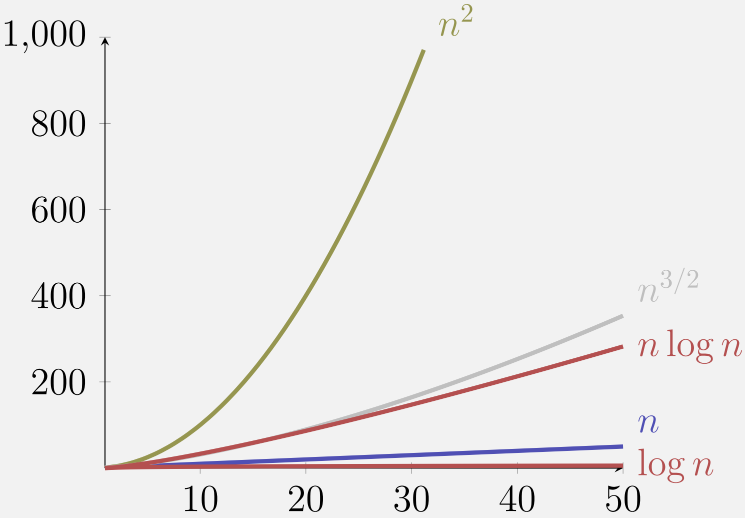
Grössere n



“Grosse” n



Logarithmen!



Beispiele

Beispiele

- $n \in \mathcal{O}(n^2)$

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.
- $2n^2 \in \mathcal{O}(n)$

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.
- $2n^2 \in \mathcal{O}(n)$ ist falsch:

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.
- $2n^2 \in \mathcal{O}(n)$ ist falsch: $\frac{2n^2}{cn} = \frac{2}{c}n \underset{n\to\infty}{\longrightarrow} \infty$!

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.
- $2n^2 \in \mathcal{O}(n)$ ist falsch: $\frac{2n^2}{cn} = \frac{2}{c}n \underset{n\to\infty}{\to} \infty$!

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.
- $2n^2 \in \mathcal{O}(n)$ ist falsch: $\frac{2n^2}{cn} = \frac{2}{c}n \underset{n\to\infty}{\to} \infty$!
- $\mathcal{O}(n) \subseteq \mathcal{O}(n^2)$

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.
- $2n^2 \in \mathcal{O}(n)$ ist falsch: $\frac{2n^2}{cn} = \frac{2}{c}n \underset{n\to\infty}{\to} \infty$!
- $\mathcal{O}(n) \subseteq \mathcal{O}(n^2)$ ist korrekt

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.
- $2n^2 \in \mathcal{O}(n)$ ist falsch: $\frac{2n^2}{cn} = \frac{2}{c}n \underset{n\to\infty}{\to} \infty$!
- $\mathcal{O}(n) \subseteq \mathcal{O}(n^2)$ ist korrekt

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.
- $2n^2 \in \mathcal{O}(n)$ ist falsch: $\frac{2n^2}{cn} = \frac{2}{c}n \underset{n\to\infty}{\to} \infty$!
- $\mathcal{O}(n) \subseteq \mathcal{O}(n^2)$ ist korrekt
- $\Theta(n) \subseteq \Theta(n^2)$

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.
- $2n^2 \in \mathcal{O}(n)$ ist falsch: $\frac{2n^2}{cn} = \frac{2}{c}n \underset{n\to\infty}{\to} \infty$!
- $\mathcal{O}(n) \subseteq \mathcal{O}(n^2)$ ist korrekt
- $\Theta(n) \subseteq \Theta(n^2)$ ist falsch:

Beispiele

- $n \in \mathcal{O}(n^2)$ korrekt, aber ungenau:
  $n \in \mathcal{O}(n)$ und sogar $n \in \Theta(n)$.
- $3n^2 \in \mathcal{O}(2n^2)$ korrekt, aber unüblich:
  Konstanten weglasssen: $3n^2 \in \mathcal{O}(n^2)$.
- $2n^2 \in \mathcal{O}(n)$ ist falsch: $\frac{2n^2}{cn} = \frac{2}{c}n \underset{n\to\infty}{\to} \infty$!
- $\mathcal{O}(n) \subseteq \mathcal{O}(n^2)$ ist korrekt
- $\Theta(n) \subseteq \Theta(n^2)$ ist falsch: $n \notin \Omega(n^2) \supset \Theta(n^2)$

Nützliches

Theorem

Seien $f, g : \mathbb{N} \to \mathbb{R}^+$ *zwei Funktionen. Dann gilt:*

1. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f \in \mathcal{O}(g), \mathcal{O}(f) \subsetneq \mathcal{O}(g)$.
2. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C > 0$ *(C konstant)* $\Rightarrow f \in \Theta(g)$.
3. $\frac{f(n)}{g(n)} \underset{n\to\infty}{\longrightarrow} \infty \Rightarrow g \in \mathcal{O}(f), \mathcal{O}(g) \subsetneq \mathcal{O}(f)$.

Zur Notation

Übliche Schreibweise

$$f = \mathcal{O}(g)$$

ist zu verstehen als $f \in \mathcal{O}(g)$.

Es gilt nämlich

$$f_1 = \mathcal{O}(g), f_2 = \mathcal{O}(g) \not\Rightarrow f_1 = f_2!$$

Beispiel

$n = \mathcal{O}(n^2), n^2 = \mathcal{O}(n^2)$ aber natürlich $n \neq n^2$.

Algorithmen, Programme und Laufzeit

Programm: Konkrete Implementation eines Algorithmus.

Laufzeit des Programmes: messbarer Wert auf einer konkreten Maschine. Kann sowohl nach oben, wie auch nach unten abgeschätzt werden.

Beispiel

Rechner mit 3 GHz. Maximale Anzahl Operationen pro Taktzyklus (z.B. 8). $\Rightarrow$ untere Schranke.
Einzelne Operation dauert mit Sicherheit nie länger als ein Tag $\Rightarrow$ obere Schranke.

Asymptotisch gesehen stimmen die Schranken überein.