FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Algorithmen und Datenstrukturen

Kapitel 4: Analyse von Algorithmen - Teil II

Prof. Ingrid Scholl

FH Aachen - FB 5

scholl@fh-aachen.de

20.04.2020

Analyse von Algorithmen - Teil II

Beispiel Fibonacci-Zahlen

Definition (Fibonacci-Zahlen)

Die Fibonacci-Zahlen sind definiert durch die folgenden Regeln:
$f_0 = 0$
$f_1 = 1$
$f_n = f_{n-1} + f_{n-2}$

Rekursiver Algorithmus:

```
// rekursiver Algorithmus zur
// Berechnung der Fibonacci Zahlen
int fib(int n) // n > 0
{
   if (n <= 1)
      return n;
   else
      return fib(n-1) + fib(n-2);
}
```

Fibonacci-Zahlen Laufzeitanalyse Rekursiver Algorithmus

Anzahl rekursiver Aufrufe: $T(n) > 2^{n/2}$

hier: $T(n)$ sei die Laufzeit zur Berechnung der n-ten Fibonacci-Zahl

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) \\
&> 2 \cdot T(n-2) \\
&> 2 \cdot 2 \cdot T(n-4) \\
&\vdots \\
&> 2 \cdot 2 \cdot \ldots \cdot 2 \cdot T(0) \\
&= 2^{n/2} \cdot T(0)
\end{aligned}
$$

Beweis: Mittels vollständiger Induktion

Fibonacci-Zahlen - Iterativer Algorithmus

```
// iterativer Algorithmus zur
// Berechnung der Fibonacci Zahlen

int fib(int n) // n > 0
{
   int i;
   int *f = new int[n];
   f[0] = 0;
   if (n > 0) {
     f[1] = 1;
     for (int i=2; i<=n; i++) {
        f[i] = f[i-1] + f[i-2];
     }
   }
   return f[n];
}
```

Laufzeitanalyse:

$T(n) = ?$

Vergleich Fibonacci-Algorithmen

Annahme: 1 arithm. Operation = 1ns

n	n+1	$2^{n/2}$	Rekursiv	Iterativ
40	41	1.048.576	1048 µs	41 ns
80	81	1.1×10^{12}	18 min	81 ns
100	101	1.1×10^{15}	13 Tage	101 ns
120	121	1.1×10^{18}	36 Jahre	121 ns
160	161	1.1×10^{24}	3.8×10^{7} Jahre	161 ns
200	201	1.1×10^{30}	4×10^{13} Jahre	201 ns

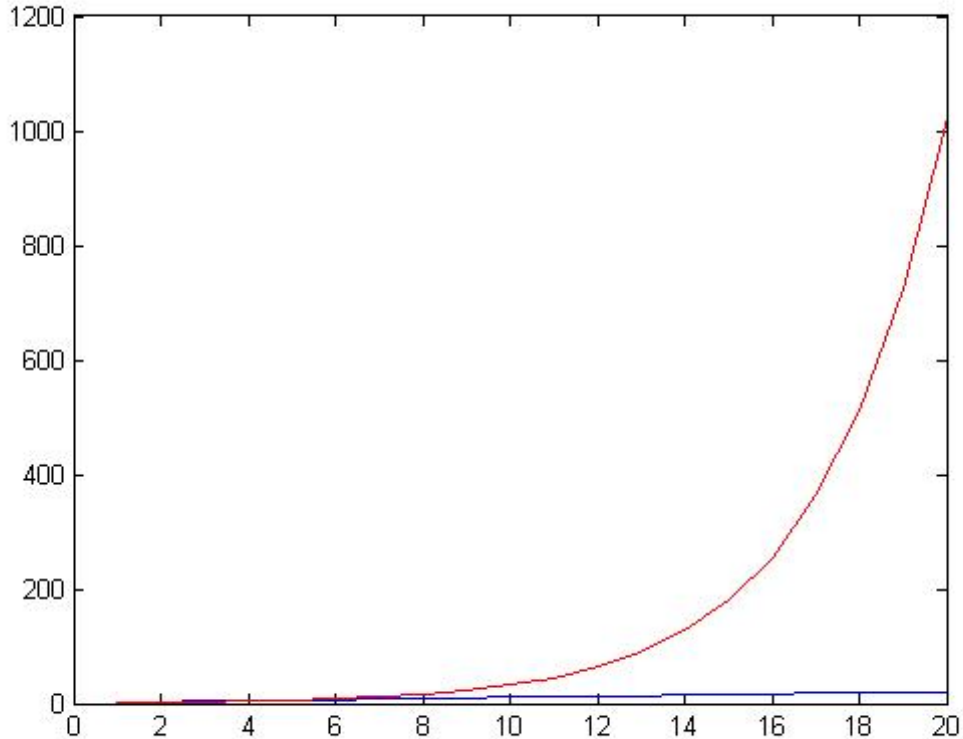
1 Mikrosek. = 1 µs = 10^{-6} s
1 Nanosek. = 1 ns = 10^{-9} s

Vergleich Fibonacci-Algorithmen

Annahme: 1 arithm. Operation = 1ns

n	n+1	$2^{n/2}$
40	41	1.048.576
80	81	1.1×10^{12}
100	101	1.1×10^{15}
120	121	1.1×10^{18}
160	161	1.1×10^{24}
200	201	1.1×10^{30}

$1\,\mu s = 10^{-6}\,s$
$1\,ns = 10^{-9}\,s$



Einstufung in Komplexitätsklassen

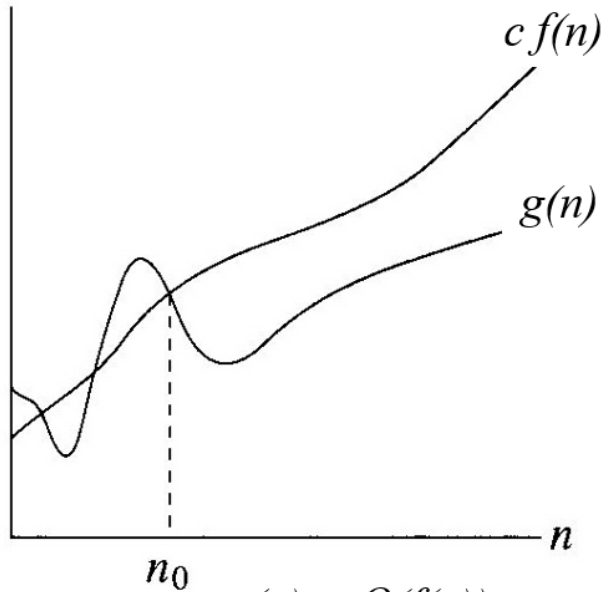
Beobachtung:

- Rekursiver Alg. Fibonacci-Zahlen: exponentielle Laufzeit
- Iterativer Alg. Fibonacci-Zahlen: lineare Laufzeit
- Einstufung der Algorithmen je nach Aufwand in verschiedene Komplexitätsklassen
- Suche dabei einfachere Vergleichsfunktion (Tilde-Approximation), die als obere Schranke für die Aufwandsfunktion eingesetzt wird.

Obere Schranke

Diese Schranke wird durch die sogenannte O-Notation angegeben.

O-Notation



Quelle: Introduction to Algorithms [Cor01]

O-Notation

Die O-Notation gibt für $g(n)$ eine obere Schranke mit einer Funktion $f(n)$ und einem konstanten Faktor c an:
$g(n) = O(f(n))$ genau dann, wenn $g(n) \leq c \cdot f(n)$ mit $n \geq n_0$

Beispiel:

Zeige $n^2 + 3n - 3 = O(n^2)$

- *$g(n)$ unser Algorithmus*
- *$f(n)$ Funktion als obere Schranke*

Definition O-Notation

Definition (O-Notation (Order of Magnitude))

Seien $f(n)$ und $g(n)$ zwei Funktionen ganzer positiver Zahlen. g ist von der Ordnung f, falls mit $c > 0$ und $n_0 \in \mathbb{N}_0$ für alle $n > n_0$ gilt:

$$g(n) \leq c \cdot f(n)$$

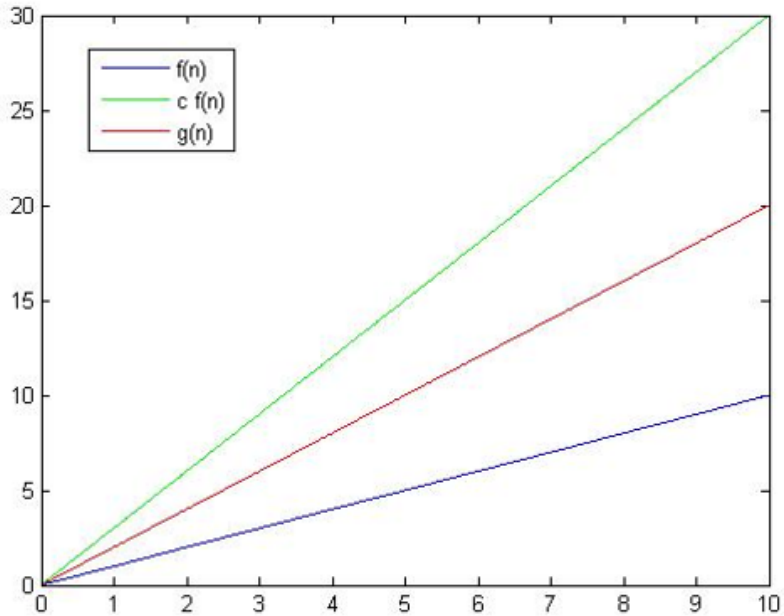
oder

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} \leq c$$

- $O(f(n))$ bezeichnet die Menge aller Funktionen g, die von der Ordnung f sind:
  $O(f) = \{g \mid \exists c > 0, \exists n_0 \in \mathbb{N}_0, \forall n > n_0 : g(n) \leq c \cdot f(n)\}$
- Man schreibt: $g(n) = O(f(n))$ *g ist von der Größenordnung f* oder *g wächst nicht wesentlich stärker als f*.

Veranschaulichung O-Notation (1)

$$O(f) = \{g \mid \exists c > 0, \exists n_0 \in \mathbb{N}_0, \forall n > n_0 : g(n) \leq c \cdot f(n)\}$$

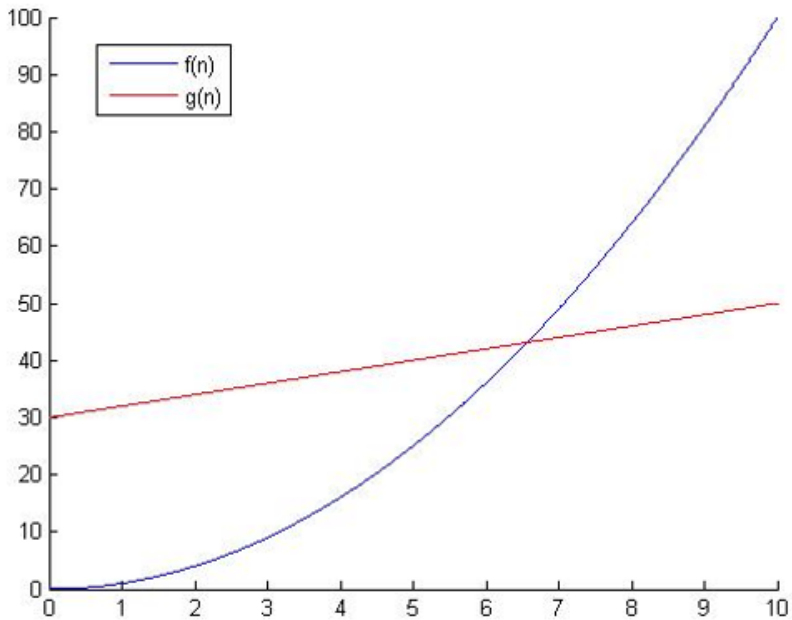


- $g(n) = 2n$, $f(n) = n$
  $\rightarrow g \in O(f)$
- $c = 3$, n_0 beliebig $\rightarrow$
  $g(n) = 2n \leq 3n = c \cdot f(n)$

Konstante Faktoren werden vernachlässigt!

Veranschaulichung O-Notation (2)

$$O(f) = \{g \mid \exists c > 0, \exists n_0 \in \mathbb{N}_0, \forall n > n_0 : g(n) \leq c \cdot f(n)\}$$



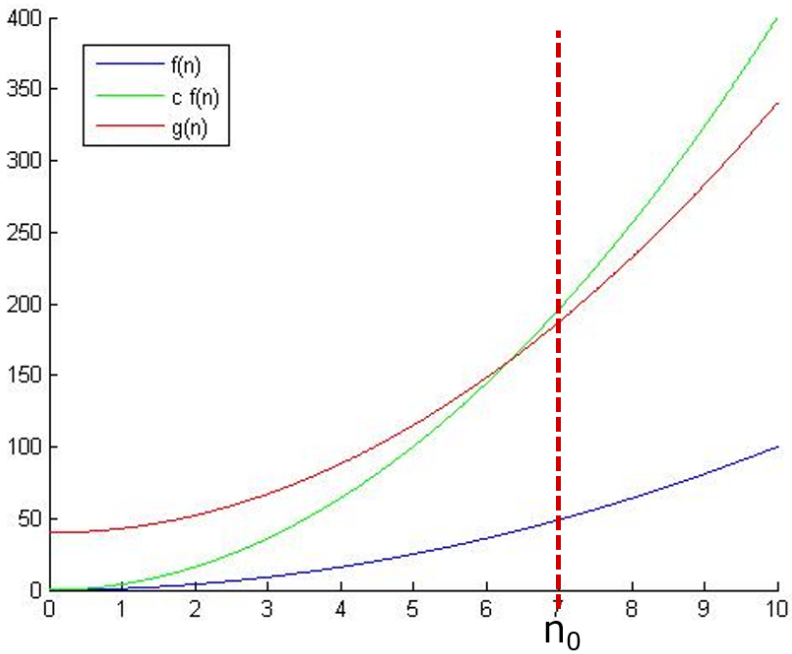
- $g(n) = 2n + 30$, $f(n) = n^2$ $\rightarrow g \in O(f)$
- $c = 1$, $n_0 = 7 \rightarrow$ $g(n) \leq f(n) \forall n \geq n_0$

Für kleine n kann die Laufzeit von $f(n)$ besser sein!

Bei asymptotischer Betrachtung wächst $g(n)$ langsamer!

Veranschaulichung O-Notation (3)

$$O(f) = \{g \mid \exists c > 0, \exists n_0 \in \mathbb{N}_0, \forall n > n_0 : g(n) \leq c \cdot f(n)\}$$



- $g(n) = 3n^2 + 40$, $f(n) = n^2$
  $\rightarrow g \in O(f)$
- $c = 4$, $n_0 = 7 \rightarrow$
  $g(n) \leq 4 \cdot f(n), \forall n \geq n_0 = 7$

Rechnen mit der O-Notation

Definiere Addition, Multiplikation, Maximumbildung:
Beispiel: $(f+g)(n) = f(n) + g(n)$

1. Addition: $f + g \in O(max\{f, g\}) = \begin{cases} O(g) & \text{falls } f \in O(g) \\ O(f) & \text{falls } g \in O(f) \end{cases}$
2. Multiplikation: $a \in O(f) \wedge b \in O(g) \Rightarrow a \cdot b \in O(f \cdot g)$
3. Falls

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} \leq c$$

existiert, ist $g \in O(f)$. (Umkehrschluß gilt nicht !)

Beispiele: Rechnen mit O-Notation

Example (Addition)

$T_1(n) \in O(n^2), T_2(n) \in O(n^3), T_3(n) \in O(n^2 log\, n)$:
dann $T_1(n) + T_2(n) + T_3(n) \in O(n^3)$

Example ($g \in O(f)$)

$f(n) = n^2, g(n) = 5n^2 + 100log(n)$
da

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} \leq c = \lim_{n\to\infty} \frac{5n^2 + 100logn}{n^2} = 5$$

folgt $g \in O(n^2)$

Weitere Wachstumsfunktionen

Definition (O-/Omega-/Theta-Notation)

Sei $f : \mathbb{N} \to \mathbb{R}^+$ eine Funktion, dann ist

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

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

$\Theta(f) = \{g : \mathbb{N} \to \mathbb{R}^+ \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 :$
$\frac{1}{c} \cdot f(n) \leq g(n) \leq c \cdot f(n)\}$

Übliche Sprechweise:

- **O-Notation** $g \in O(f)$:
  f ist obere Schranke von g. g wächst höchstens so schnell wie f.
- **Omega-Notation** $g \in \Omega(f)$:
  f ist untere Schranke von g. g wächst mindestens so schnell wie f.
- **Theta-Notation** $g \in \Theta(f)$:
  f ist die Wachstumsrate von g. g wächst wie f.

Wachstum für ausgewählte Komplexitätsklassen

f(n)	$n =$ 2	$2^4 =$ 16	$2^8 =$ 256	$2^{10} =$ 1024	$2^{20} =$ 1048576
ld n	1	4	8	10	20
n	2	16	256	1024	1048576
$n \cdot ldn$	2	64	2048	10240	20971520
n^2	4	256	65636	1048576	$\approx 10^{12}$
n^3	8	4096	16777200	$\approx 10^9$	$\approx 10^{18}$
2^n	4	65536	$\approx 10^{77}$	$\approx 10^{308}$	$\approx 10^{315\,653}$

Zeitaufwand für einige Problemgrößen

Wieviele Eingabeelemente können verarbeitet werden, wenn die Zeit durch eine maximale Dauer T begrenzt ist.
Annahme: 1 Schritt = 1 µs
g ist das größte lösbare Problem in der Zeit T.

g	1 Min	1 Std	1 Tag	1 Woche	1 Jahr
n	6×10^7	3×10^9	8.6×10^{10}	6×10^{11}	3×10^{13}
n^2	7750	6×10^4	2.9×10^5	7.8×10^5	5.6×10^5
n^3	391	1530	4420	8450	31600
2^n	25	31	36	39	44

Beobachtung

In 1 Jahr können bei exponentiellem Aufwand nur ca. 44 Eingabedaten verarbeitet werden!

Analyse von Algorithmen - Teil II

Vielen Dank!

Prof. Ingrid Scholl
FH Aachen
Fachbereich für Elektrotechnik und Informationstechnik
Graphische Datenverarbeitung und Grundlagen der Informatik
MASKOR Institut
Eupener Straße 70
52066 Aachen
T +49 (0)241 6009-52177
F +49 (0)241 6009-52190
scholl@fh-aachen.de

www.fh-aachen.de