Komplexität von Algorithmen

by

Dr. Günter Kolousek

Algorithmus

- Ursprung: Musa al-Chwarizmi (ca. 780 - 840) $\rightarrow$ Buch über indische Zahlensysteme und Rechenvorschriften (Addition, Subtraktion, Multiplikation, Division, Wurzelziehen)
- eine eindeutige Handlungsvorschrift wie eine Klasse von Problemen gelöst werden kann
- Hmm, was heißt das eigentlich genau? Was wird damit spezifiziert?

Algorithmus

- Ursprung: Musa al-Chwarizmi (ca. 780 - 840) $\rightarrow$ Buch über indische Zahlensysteme und Rechenvorschriften (Addition, Subtraktion, Multiplikation, Division, Wurzelziehen)
- eine eindeutige Handlungsvorschrift wie eine Klasse von Problemen gelöst werden kann
- Hmm, was heißt das eigentlich genau? Was wird damit spezifiziert?
  - Was wird als Eingabe für den Algorithmus akzeptiert?
  - Welche Schritte werden in welcher Reihenfolge durchgeführt?
  - Wann stoppt der Algorithmus und was wird ausgegeben?

Algorithmus

- Ursprung: Musa al-Chwarizmi (ca. 780 - 840) $\rightarrow$ Buch über indische Zahlensysteme und Rechenvorschriften (Addition, Subtraktion, Multiplikation, Division, Wurzelziehen)
- eine eindeutige Handlungsvorschrift wie eine Klasse von Problemen gelöst werden kann
- Hmm, was heißt das eigentlich genau? Was wird damit spezifiziert?
  - Was wird als Eingabe für den Algorithmus akzeptiert?
  - Welche Schritte werden in welcher Reihenfolge durchgeführt?
  - Wann stoppt der Algorithmus und was wird ausgegeben?
- Hmm, geht es etwas genauer?

Algorithmus

- Ursprung: Musa al-Chwarizmi (ca. 780 - 840) $\rightarrow$ Buch über indische Zahlensysteme und Rechenvorschriften (Addition, Subtraktion, Multiplikation, Division, Wurzelziehen)
- eine eindeutige Handlungsvorschrift wie eine Klasse von Problemen gelöst werden kann
- Hmm, was heißt das eigentlich genau? Was wird damit spezifiziert?
  - Was wird als Eingabe für den Algorithmus akzeptiert?
  - Welche Schritte werden in welcher Reihenfolge durchgeführt?
  - Wann stoppt der Algorithmus und was wird ausgegeben?
- Hmm, geht es etwas genauer?

  Eine Berechnungsvorschrift zur Lösung eines Problems heißt genau dann Algorithmus, wenn eine zu dieser Berechnungsvorschrift äquivalente Turingmaschine existiert, die für jede Eingabe, die eine Lösung besitzt, stoppt. – wikipedia

Algorithmus – 2

- Keine Angabe wie ein Algorithmus ausgeführt wird!
  - vom Menschen
  - einer mechanischen Rechenmaschine oder
  - einem Computer
- Ausführung: Algorithmus $\rightarrow$ Maschinencode
  - Spezifikation
    - mathematische (formale) Beschreibung
    - Pseudocode
    - Programmiersprache
  - Übersetzung
    - Maschinencode

Algorithmus – 3

- Algorithmus zur Berechnung der Fakultät
  - mathematische Beschreibung, fact $= f$

$$
\begin{aligned}
f &: \mathbb{N} \to \mathbb{N} \\
f(n) &= \begin{cases} 1 & \text{für } n = 0 \\ n \cdot f(n-1) & \text{für } n > 0 \end{cases}
\end{aligned}
$$

  - Pseudocode ;-)

```
# pre: type(n) == int and n >= 0
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```

Algorithmus – 4

- Algorithmus zur Berechnung der Fakultät
  - Programmiersprache

```
// pre: n >= 0
long long fact(long long n) {
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
```

Komplexität eines Algorithmus

- Fragestellung: Laufzeit oder Speicherbedarf eines Algorithmus?
  - unabhängig von der verwendeten Technologie!
  - abhängig von der ”Größe” der Eingabedaten
    - z.B. 100, 1000 oder 100 Millionen Daten zu sortieren
    - Symbol: n
- Arten
  - Speicherkomplexität
  - Zeitkomplexität
    - # der elementaren Operationen für Abarbeitung des Algorithmus
    - Annahme: Zeit je elementarer Operation ist bestimmte Zeit

Komplexität eines Algorithmus – 2

- Von nun an nur Zeitkomplexität!
- Beispiel: lineare Suche in `[1,7,4,2,9,3,6,5,8]`
- Fälle
  - Bester Fall (best-case)
    - Suche nach `1`
  - Durchschnittlicher Fall (average-case)
    - Suche nach `9`
  - Schlechtester Fall (worst-case)
    - Suche nach `8`
- meist: worst-case

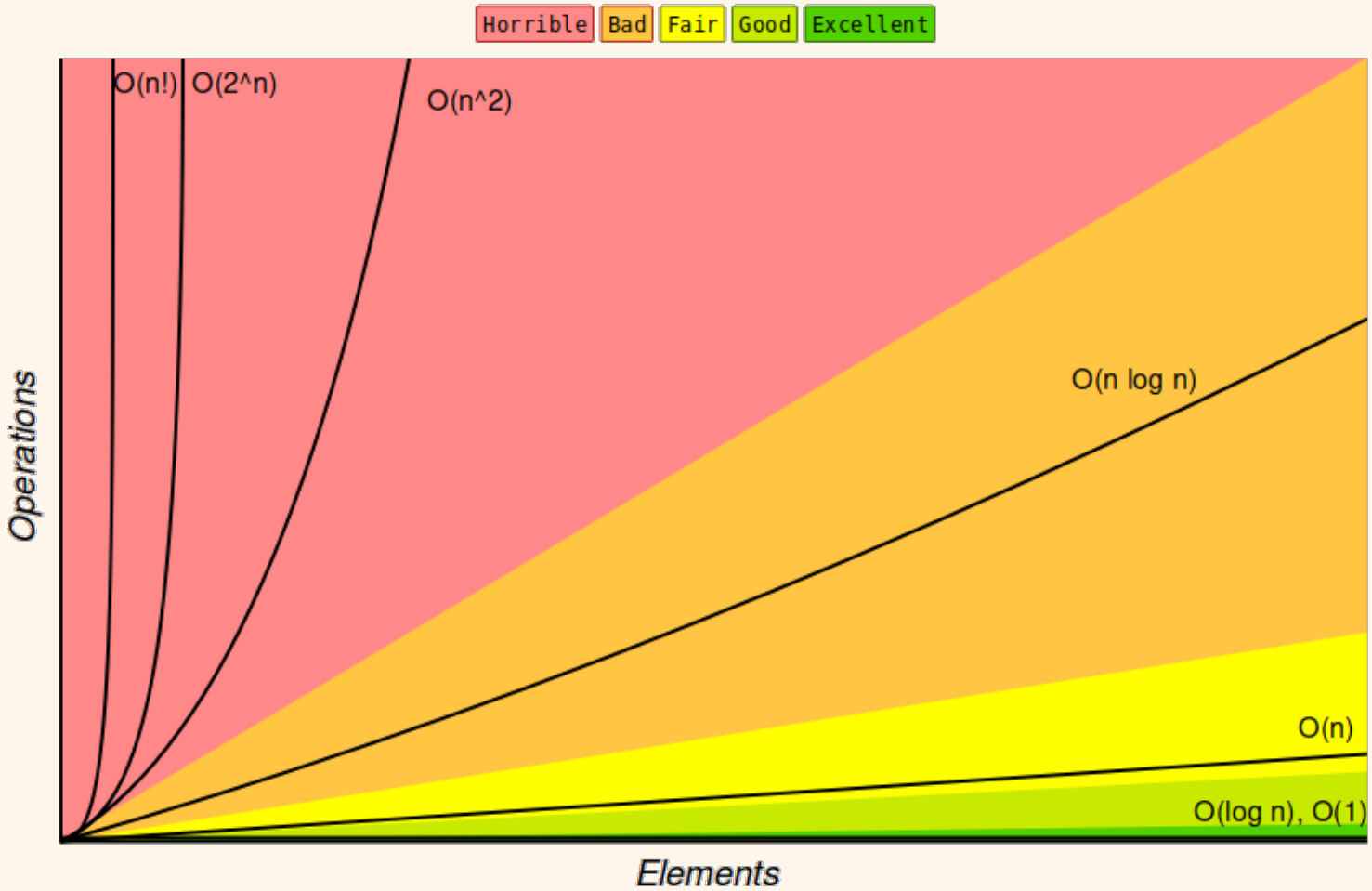
Komplexität eines Algorithmus – 3

- Landau-Symbol
  $O(g(n)) = \{f(n) | \exists C \geq 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : |f(n)| \leq C \cdot |g(n)|\}$
  - ”Groß-Oh Notation” (Big-O notation)
  - C und n_0 beliebig!
    - Konstanten spiegeln nur technologische Möglichkeiten wider!
  - statt $f(n) \in O(g(n))$ wird einfacher $f(n) = O(g(n))$ geschrieben
- z.B.:
  - $3n^2 + 10 \in O(n^2)$, da z.B. für $C = 4, n_0 = 4$ gilt, dass
    $\forall n \geq n_0 : 3n^2 + 10 \leq C \cdot n^2$
  - $0.001n^3 \notin O(n^2)$, da kein C und n_0 existiert, dass
    $\forall n \geq n_0 : 0.001n^3 \leq Cn^2$

Schrankenfunktionen

- $O(1)$ … konstant
  - z.B. Zugriff in Hasharray
- $O(\log(n))$ … logarithmisch
  - z.B. Suche in BSB
- $O(n)$ … linear
  - z.B. Lineare Suche in Liste
- $O(n \cdot \log(n))$ … quasi linear, acuh superlinear
  - z.B. Quicksort
- $O(n^k)$ für $k \geq 1$ … polynomial
  - $O(n^2)$ … quadratisch
    - z.B. Bubblesort
- $O(d^n)$ für $d > 1$ … exponentiell
  - z.B. Türme von Hanoi
- $O(n!)$ … faktoriell
  - z.B. TSP hat $O(n!)(mitbrute - forceSuche)$

Schrankenfunktionen – 2



Quelle: http://www.bigocheatsheet.com/

Optimierung

- Änderung der Anforderungen

  Always remember, however, that there's usually a simpler and better way to do something than the first way that pops into your head. – Donald Knuth

  - Können andere (weniger) Datensätze verwendet werden?
  - Wird die geforderte Genauigkeit wirklich benötigt?
  - Kann der Problemraum verkleinert werden?
- Ändern des Entwurfes

  People who are more than casually interested in computers should have at least some idea of what the underlying hardware is like. Otherwise the programs they write will be pretty weird. – Donald Knuth

  - Systemarchitektur (z.B. Client/Server vs. zentrale Architektur, peer-to-peer,…)
  - Technologien (z.B. Java vs. C++)

Optimierung – 2

- Optimierung der Algorithmen

  If you optimize everything, you will always be unhappy. – Donald Knuth

  - Laufzeit vs. Speicherbedarf?
    - meist widersprüchlich!
  - Algorithmen vs. Datenstrukturen?
  - Häufige Fälle vs. seltene Fälle?
  - Caching?
- Optimierung des Programmcodes

  Premature optimization is the root of all evil. – Donald Knuth

  - Laufzeit vs. Speicherbedarf?
  - Datenstrukturen?
  - Profiling um kritischen Pfad zu finden!
  - siehe Folien über Compilertechnologie!

Optimierung – Beispiel

- Potenzieren

$$f : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$$

$$f(x,n) = \begin{cases} 1 & \text{für } n = 0 \\ x \cdot f(x, n-1) & \text{für } n > 0 \end{cases}$$

- naive Umsetzung

```
double pow(double x, unsigned int n) {
    if (n == 0)
        return 1;
    else
        return x * pow(x, n - 1);
}
```

$\rightarrow$ Rekursion (Stack!, Funktionsaufrufe!)

$\rightarrow O(n)$

Optimierung – Beispiel – 2

- Umsetzen in iterativen Ansatz ($\rightarrow$ Programmcode!)

```
double pow(double x, unsigned int n) {
    double res{(n == 0) ? 1 : x};
    for (unsigned int i{1}; i < n; res *= x, ++i);
    return res;
}
```

  $\rightarrow$ keine Rekursion!

  $\rightarrow O(n)$

Optimierung – Beispiel – 3

- Optimierung des Algorithmus!
  - Unterscheidung in gerade und ungerade Exponenten

$$x^{2k} = x^k \cdot x^k = (x^k)^2$$
$$x^{2k+1} = x \cdot x^k \cdot x^k = x \cdot (x^k)^2$$

  - Indizes bilden eine geometrisch fallende Folge: $n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots$
  - $\rightarrow O(\log_2 n)$
- resultierender Programmcode

```
double pow(double x, unsigned int n) {
    double res{1};
    if (n) {
        res = pow(x, n / 2);
        res *= res;
        if (n % 2) res *= x;
    }
    return res;
}
```

$\rightarrow$ Rekursion!

Optimierung – Beispiel – 4

- Umsetzen in iterativen Ansatz

```
double pow(double x, unsigned int n) {
    double res{1};
    do {
        if (n & 1) res *= x;  // n % 2 == 1?
        x *= x;
    } while (n >>= 1);  // n / 2 > 0?
    return res;
}
```

$\rightarrow$ keine Rekursion

$\rightarrow$ Verwendung von Bit-Operationen anstelle von arithmetischen Operationen (mit guten Compilern…)

… Ansatz von Donald E. Knuth, The Art of Computer Programming, Vol 2, Chapter 4

- Bände 1, 2, 3 und 4A sind erschienen (seit 1962)
- Bände 4B, 4C, 4D, 5, 6, 7 sind in Planung (Knuth ist dzt. über 80 Jahre alt!)