Algorithmische Komplexität

Inhaltsverzeichnis

Weiterführende Literatur:

- Saake und Sattler, *Algorithmen und Datenstrukturen*, Seite 196-206 (PDF 214-224)
- *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 19-35 (PDF 11-24)

Die Algorithmische Komplexität macht Aussagen über den *Ressourcenverbrauch* (Laufzeit und Resourcenverbrauch) eines Algorithmus. Aussagen zur Algorithmischen Komplexität sind unabhängig von: Ressourcenverbrauch

- Programmiersprache
- installiertem Betriebssystem
- Prozessorleistung
- Speicherausstattung
- zugrunde liegender Computerarchitektur

Die Algorithmische Komplexität beschreibt die *Effizienz* eines eingesetzten Algorithmus und entscheidet über die *praktische Anwendbarkeit* eines Verfahrens.[1] Effizienz praktische Anwendbarkeit

Zeitkomplexität

Die Speicherkapazität wächst schneller als die Taktfrequenz. Die Zeit ist die meist beschränkende Größe. Die Anzahl der für seine Ausführung benötigten elementaren Operationen eines Algorithmus in Abhängigkeit von der Eingabelänge n heißt, Laufzeit $T(n)$. Ziel ist die Laufzeitanalyse für große Eingaben. Mit Fachbegriff nennt man diesen Sachverhalt *„asymptotische Komplexität“*.[2]

Da der Aufwand schon vom Ansatz her ohnehin nur grob abgeschätzt werden kann, brauchen einige Informationen, wie etwa Konstanten nicht berücksichtig werden.

[1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 20 (PDF 12).
[2] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 21 (PDF 12).

$T_n = \mathcal{O}(\log n)$, bedeutet zum Beispiel, dass T_n *„von der Ordnung log n“* ist, wobei multiplikative und additive Konstanten sowie die Basis des Logarithmus unspezifiziert bleiben.[3]

Da die $\mathcal{O}(g)$-Notation eine obere Grenze für eine Klasse von Funktionen definiert, kann die Funktion g jeweils vereinfacht werden, indem konstante Faktoren weggelassen werden sowie nur der höchste Exponent berücksichtigt wird.[4]

$\mathcal{O}$-Notation

Offensichtlich gilt für große n: $1 < \log_2(n) < n < n^2 < n^3 < ... < 2^n$

Bestimmung einer asymptotischen oberen Schranke g.

$T(n) \in \mathcal{O}(g(n))$ bedeutet:

- $T(n)$ wächst höchstens so schnell wie die Funktion $g(n)$
- Es gibt eine Konstante c, sodass für große n immer gilt: $T(n) \leq c \cdot g(n)$
- $c \cdot g(n)$ ist damit eine obere Schranke für $T(n)$

Definition (O-Notation)

$\mathcal{O}(g(n)) = \{T(n)|$ Es gibt ein $c > 0$ und ein n_0 aus, $\mathbb{N}$ sodass gilt: $0 \leq T(n) \leq c \cdot g(n)$ für alle $n \geq n_0\}$

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

```
max = messwert[0]; // 1
for (int i = 0; i < anzMesswerte; i++) { // n-mal
  System.out.println(i + ":" + messwert[i]); // konstanter Bedarf
  if (messwert[i] > max) {
    max = messwert[i];
  }
}
```

$T(n) = 1 + \text{const} * n \rightarrow$ Laufzeit ist in $\mathcal{O}(n)$

```
for (int k = 0; k < anzVersuchsreihen; k++) { // n-mal
  System.out.println(k + "-te Versuchsreihe: "); // konstant
  for ( int i = 0; i < anzMesswerte; i++) {
    System.out.println(k + "." + i + ":" + messwert[k][i]); // n-mal
  }
}
```

$T(n) = n + n^2 \rightarrow$ wächst quadratisch mit $n \rightarrow T(n)$ ist in $\mathcal{O}(n^2)$[5]

Definition (Best Case, Worst Case, Average Case)

best case

Der günstigste Fall (*best case*) für einen Algorithmus liegt vor, wenn sein Ergebnis mit minimalem Aufwand erzielt werden kann.

worst case

Es handelt sich um den ungünstigsten Fall (*worst case*), wenn der Aufwand zur Erzielung des Ergebnisses maximal ist.

Im Durchschnittsfall (*average case*) ist ein mittlerer erwarteter Aufwand erforderlich.[6]

Beispiel

Ist eine gegebene Zahl n eine Primzahl?

best case $n = 1467294058367694867239795 6$ (Die Zahl ist gerade und daher durch zwei teilbar)

worst case n ist eine Primzahl

average case (Um den *average case* zu bestimmen, ist eine aufwändige statistische Berechnung nötig.)

Beispiel einer Komplexitätsberechnung

```
1  int summe = 0;
2  for (int j = 1 ; j < n ; j ++) {
3    for (int i = 1 ; i < j ; i ++) {
4      summe = summe + 1;
5    }
6  }
7  for (int k = 0 ; k <= n ; k ++) {
8    a[k] = k;
9  }
```

Offenbar fallen die Fälle *Best Case*, *Worst Case* und *Average Case* zusammen, da die Anzahl der Durchläufe der Wiederholungsanweisungen fest ist.[7]

ersten Zeile

```
1  int summe = 0;
```

Die Zuweisung in der ersten Zeile benötigt die konstante Zeit c_1, also gilt: $T_{\text{Erste Zuweisung}}(n) = \mathcal{O}(1)$[8]

erste for-Anweisung

```
2  for (int j = 1 ; j < n ; j ++) {
3    for (int i = 1 ; i < j ; i ++) {
4      summe = summe + 1;
5    }
6  }
```

Die innere Zuweisung benötigt einen konstanten Zeitbedarf c_2. Die innere Wiederholungsanweisung eine j-fache Ausführung, d. h. den Zeitbedarf $j \cdot c_2$. Die

[3] Saake und Sattler, *Algorithmen und Datenstrukturen*, Seite 200 (PDF 2018).
[4] Saake und Sattler, *Algorithmen und Datenstrukturen*, Seite 201 (PDF 2019).
[5] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 25 (PDF 17).
[6] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 26 (PDF 18).
[7] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 27 (PDF 19).
[8] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 28 (PDF 20).

äußere Wiederholungsanweisung eine n-fache Ausführung. Bei jedem Durchlauf wird j um 1 inkrementiert. Die Laufzeit für die verschachtelte for-Anweisung beträgt:

$$c_2 \cdot (1 + 2 + \ldots + n) = c_2 \cdot \frac{n \cdot (n+1)}{2} = c_2 \cdot \frac{(n^2 + n)}{2} = c_2/2 \cdot (n^2 + n)$$

[9]

$$\mathcal{O}(c_2/2 \cdot (n_2 + n)) = \mathcal{O}(n^2 + n) = \mathcal{O}(n^2)$$

[10]

zweite for-Anweisung

Für die zweite for-Anweisung gilt:

```
7  for (int k = 0 ; k <= n ; k ++) {
8    a[k] = k;
9  }
```

$$T_{\text{Zweite for-Anweisung}}(n) = (n+1) * c_3 = \mathcal{O}(n)$$

Laufzeitverhalten für das gesamte Codefragment:

$$\begin{aligned} T_{\text{gesamt}} \\ &= T_{\text{Erste Zuweisung}}(n) + T_{\text{Erste for-Anweisung}}(n) + T_{\text{Zweite for-Anweisung}}(n) \\ &= \mathcal{O}(1) + \mathcal{O}(n^2) + \mathcal{O}(n) \\ &= \mathcal{O}(n^2) \end{aligned}$$

Insgesamt ist die Laufzeit für das gesamte Codefragment also quadratisch.[11]

Typische Komplexitätsklassen

Quellen[12][13][14]

konstant	$\mathcal{O}(1)$
logarithmisch	$\mathcal{O}(\log n)$
linear	$\mathcal{O}(n)$
linearithmisch	$\mathcal{O}(n \log n)$
quadratisch	$\mathcal{O}(n^2)$
polynomial	$\mathcal{O}(n^k)$ für $k \geq 1$
exponentiell	$\mathcal{O}(d^n)$ für $d > 1$

[9] *Wikipedia-Artikel „Gaußsche Summenformel“*.
[10] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 29 (PDF 21).
[11] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 30.
[12] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 24 (PDF 16).
[13] Saake und Sattler, *Algorithmen und Datenstrukturen*, Seite 202 (PDF 220).
[14] *Wikipedia-Artikel „Komplexitätstheorie“*, Absatz „Anforderungen an Schrankenfunktionen“.

Landau-Symbole

Landau-Symbole werden in der Informatik verwendet, um das asymptotische Verhalten von Funktionen und Folgen zu beschreiben. In der Informatik werden sie bei der Analyse von Algorithmen verwendet und geben ein Maß für die Anzahl der Elementarschritte oder der Speichereinheiten in Abhängigkeit von der Größe der Eingangsvariablen an. Die Komplexitätstheorie verwendet sie, um verschiedene Probleme danach zu vergleichen, wie „schwierig“ oder aufwendig sie zu lösen sind. Man sagt, „schwere Probleme“ wachsen exponentiell mit der Instanz oder schneller und für „leichte Probleme“ existiert ein Algorithmus, dessen Laufzeitzuwächse sich durch das Wachstum eines Polynoms beschränken lassen. Man nennt sie (nicht) polynomiell lösbar.[15]

Notation	Anschauliche Bedeutung
$f \in o(g)$	f wächst langsamer als g
$f \in \mathcal{O}(g)$	f wächst nicht wesentlich schneller als g
$f \in \Theta(g)$	f wächst genauso schnell wie g
$f \in \Omega(g)$	f wächst nicht wesentlich langsamer als g
$f \in \omega(g)$	f wächst schneller als g

Asymptotische untere Schranke: Ω-Notation

- $f(n) \in \Omega(g(n))$: $f(n)$ wächst mindestens so schnell wie $g(n)$
- $\Omega(g(n)) = \{ f(n) \mid \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : 0 \leq c \cdot g(n) \leq f(n) \}$

Asymptotisch scharfe Schranke ermitteln: Θ-Notation

- $f(n) \in \Theta(g(n))$: $f(n)$ wächst ebenso schnell wie $g(n)$
- $\Theta(g(n)) = \{f(n) | \exists c_1 > 0 \exists c_2 > 0 \exists n_0 > 0 \forall n \geq n_0 : 0 \leq c1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}$

„Sandwich-Gedanke“

- $\mathcal{O}$ ist eine (möglicherweise viel zu große obere Schranke).
- Ω ist eine (möglicherweise viel zu kleine untere Schranke). Beide helfen bei der Charakterisierung von Aufwänden nur bedingt.
- Es gilt: $f(n) \in \Theta(g(n))$ genau dann, wenn $f(n) \in \mathcal{O}(g(n))$ und $f(n) \in \Omega(g(n))$.
- Aufwandsabschätzungen mit Hilfe von Θ wären ideal, aber leider lassen sich Beweise oft nur schwer führen.

Daher werden oft in der Literatur nur $\mathcal{O}$-Abschätzungen verwendet, aber mit der Maßgabe, dass die kleinstmögliche obere Schranke angegeben wird.[16]

Vergleichsbasiertes Sortieren

Allgemeine Verfahren basieren auf dem paarweisen Vergleich der zu sortierenden Elemente, ob das eine Element „kleiner“ als, „größer“ als oder „gleich(groß)“

[15] *Wikipedia-Artikel „Landau-Symbole“*.

[16] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 37 (PDF 26).

wie das andere Element ist. Bei der Komplexitätsanalyse wird davon ausgegangen, dass der Aufwand zum Vergleich zweier Elemente konstant ist.[17][18]

Sortierverfahren	Best-Case	Average-Case	Worst-Case
Binary Tree Sort	$\Theta(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$	$\Theta(n^2)$
Bubblesort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
Heapsort	$\Theta(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$
Insertionsort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
Mergesort	$\Theta(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$
Quicksort	$\Theta(n \cdot \log(n))$	$\Theta(n \cdot \log(n))$	$\Theta(n^2)$
Selectionsort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$

Exkurs: Logarithmus
Der Logarithmus zur Basis b ist die Umkehrfunktion der allgemeinen Exponentialfunktion zur positiven Basis b:

$x \mapsto b^x$.

Die Funktionen b^x und $\log_b x$ sind also Umkehrfunktionen voneinander, d. h. Logarithmieren macht Exponenzieren rückgängig und umgekehrt:

$b^{\log_b x} = x \quad \text{und} \quad \log_b(b^x) = x$.

Der natürliche Logarithmus ergibt sich mit der Basis $b = \mathrm{e}$, wobei $\mathrm{e} = 2{,}718281828459\ldots$ die Eulersche Zahl ist.[a]

[a] **wiki:logarithmus**.

Exkurs: asymptotische Abschätzung
Unter der Zeitkomplexität eines Problems wird in der Informatik die Anzahl der Rechenschritte verstanden, die ein optimaler Algorithmus zur Lösung dieses Problems benötigt, in Abhängigkeit von der Länge der Eingabe. Man spricht hier auch von der *asymptotischen Laufzeit* und meint damit, in Anlehnung an eine *Asymptote*, das Zeitverhalten des Algorithmus für eine *potenziell unendlich große Eingabemenge*.

Es interessiert also *nicht* der *Zeitaufwand eines konkreten Programms* auf einem bestimmten Computer, sondern viel mehr, wie der Zeitbedarf wächst, wenn mehr Daten zu verarbeiten sind, also z. B. ob sich der Aufwand für die doppelte Datenmenge verdoppelt oder quadriert (Skalierbarkeit).[a]

[a] *Wikipedia-Artikel „Zeitkomplexität“*.

Literatur

[1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2. Sortieren, Suchen, Komplexität*. `https://www.studon.fau.de/file2566441_download.html`.

[2] Gunter Saake und Kai-Uwe Sattler. *Algorithmen und Datenstrukturen. Eine Einführung in Java*. 2014.

[3] *Wikipedia-Artikel „Gaußsche Summenformel“*. `https://de.wikipedia.org/wiki/Gaußsche_Summenformel`.

[17] *Wikipedia-Artikel „Sortierverfahren“*.

[18] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 35 (PDF 24).

[4] *Wikipedia-Artikel „Komplexitätstheorie“*. https://de.wikipedia.org/wiki/Komplexitätstheorie.

[5] *Wikipedia-Artikel „Landau-Symbole“*. https://de.wikipedia.org/wiki/Landau-Symbole.

[6] *Wikipedia-Artikel „Sortierverfahren“*. https://de.wikipedia.org/wiki/Sortierverfahren.

[7] *Wikipedia-Artikel „Zeitkomplexität“*. https://de.wikipedia.org/wiki/Zeitkomplexität.