

Abschätzung der Laufzeit von Algorithmen

Einstiegsaufgabe: Wenn Sie in der Lerngruppe jeder für sich oder in Teams alle eine Lösung für die gleiche Problemstellung suchen, finden Sie vermutlich häufig ganz unterschiedliche Algorithmen und Implementierungen. Zum Beispiel haben Sie vielleicht schon einmal einen Algorithmus entwickelt, der die Werte in einer Reihung sortiert und dabei ganz unterschiedliche Vorgehensweisen entdeckt. Diskutieren Sie, welche Kriterien Sie anlegen würden, um zu entscheiden, welcher von allen Algorithmen, die das Problem lösen, der Beste ist.

Komplexität von Algorithmen

Häufig gibt es für ein Problem verschiedene Algorithmen zur Lösung. Auch wenn alle Algorithmen zum gleichen Ergebnis führen, können sie sich z. B. in der Zeit, die sie für die Berechnung benötigen, oder in dem während der Berechnung benötigten Speicherplatz unterscheiden. Beides hängt in der Regel von der Menge der zu verarbeitenden Daten ab. Suchen und Sortieren sind Beispiele für Operationen, die sehr häufig und in der Regel für sehr große Datenmengen ausgeführt werden müssen. Man ist daher bestrebt, für diese Operationen einen möglichst effizienten Algorithmus zu verwenden. Um die Algorithmen vergleichen zu können, schätzt man ihre **Laufzeit** (Rechenzeit für die Ausführung) und ihren Speicherbedarf in Abhängig von der Größe der Eingabe ab. Man spricht dabei auch von der **Komplexität** eines Algorithmus. Im Folgenden konzentrieren wir uns auf die Laufzeitkomplexität eines Algorithmus.

Aufgabe 1: Diskutieren Sie, weshalb es schwierig ist, die Rechenzeit, die ein Algorithmus benötigt, exakt zu bestimmen.

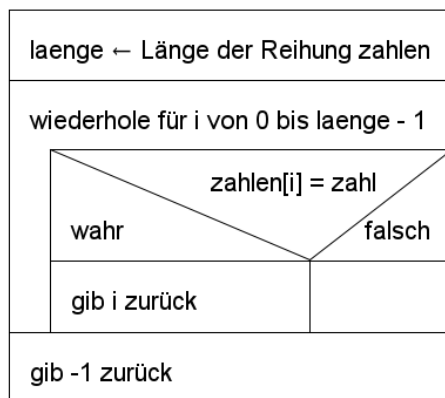
Da die exakte Rechenzeit eines Algorithmus von vielen Faktoren abhängt, nimmt man zur Einordnung der Effizienz nur eine grobe Abschätzung vor. Als Maß für die Laufzeit eines Algorithmus dient die Anzahl der **elementaren Operationen**, die in Abhängigkeit von der **Größe n der Eingabe** ausgeführt werden müssen. Elementare Operationen sind z. B. Wertzuweisungen an Variablen, Vergleichsoperationen oder arithmetische Operationen. Arbeitet ein Algorithmus auf einer Reihung, könnte die Größe der Eingabe beispielsweise die Anzahl der Elemente in der Reihung sein. Mit Größe der Eingabe ist also nicht unbedingt der Wert der Eingabe gemeint, sondern es kann z. B. die Anzahl der zu verarbeitenden Elemente entscheidend sein. Man spricht daher auch allgemein von der Größe des Problems.

Bei der Betrachtung der Effizienz von Algorithmen ordnet man die Laufzeit einer Klasse von Funktionen zu. Beispielsweise gehören Algorithmen, die $6n$, $7n + 4$ oder $6n + 3$ elementare Rechenoperationen benötigen, zur gleichen Klasse. Entscheidend ist, dass sich die Laufzeit mit einer linearen Funktion beschreiben lässt und diese für große Eingaben damit langsamer wächst, als beispielsweise die Laufzeit der Algorithmen in der Klasse der Algorithmen mit quadratischer Laufzeit.

Aufgabe 2: Abbildung 1 zeigt zwei verschiedenen algorithmische Umsetzungen einer Suche nach einer Zahl in einer global definierten Reihung `zahlen` vom Inhaltstyp `Ganzzahl`. Die relevante Eingabegröße n ist die Anzahl der Werte in der Reihung `zahlen`.

- Bestimmen Sie jeweils die durchschnittliche Laufzeit (Anzahl an benötigten elementaren Operationen) in Abhängigkeit von n .
- Begründen Sie, dass beide Laufzeiten zu der gleichen Funktionsklasse gehören.

suche1(zahl: Ganzzahl): Ganzzahl



suche2(zahl: Ganzzahl): Ganzzahl

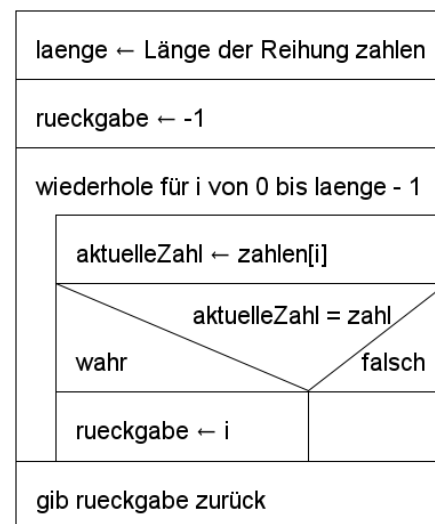


Abbildung 1: zwei verschiedene algorithmische Umsetzungen für eine Suche

Vergleich der Komplexitätsklassen

Durch die Zuordnung der Laufzeit zu einer Funktionsklasse ist es möglich, die Laufzeit verschiedener Algorithmen für sehr große Eingaben zu vergleichen und zu entscheiden, ob ein Algorithmus in diesem Fall effizienter ist als ein anderer, ohne dabei die exakte Anzahl der Operationen oder die exakte Rechenzeit bestimmen zu müssen. Möglicherweise stellt man dabei sogar fest, dass die Ausführung eines Algorithmus ab einer gewissen Eingabegröße zu lange dauern würde, um ihn praktisch anzuwenden.

Im Zusammenhang mit der Laufzeit eines Algorithmus spricht man von Komplexitätsklassen. Man sagt ein Algorithmus gehört zu einer Komplexitätsklasse oder liegt in einer Komplexitätsklasse. Die Komplexitätsklasse wird nach dem Mathematiker Edmund Landau angegeben als $O(g(n))$, z. B. $O(n)$ oder $O(\log n)$. Das O steht für „Ordnung von“.

Betrachtet man das Wachstumsverhalten der verschiedenen Funktionsklassen für große Werte von n , kann man die entsprechenden Komplexitätsklassen in eine Reihenfolge von langsam bis schnell wachsend bringen.

Aufgabe 3: In der Informatik unterscheidet man häufig die folgenden Komplexitätsklassen: $O(n)$, $O(2^n)$, $O(n^2)$, $O(1)$, $O(\log n)$, $O(n \cdot \log n)$, $O(n^3)$.

Vergleichen Sie das Verhalten der Laufzeiten für große Eingaben, d. h. große Werte für n und ordnen Sie die Komplexitätsklassen von der besten bis zur schlechtesten Laufzeit. Betrachten Sie dazu den Verlauf exemplarischer Funktionsgraphen der jeweiligen Funktionsklasse in Abbildung 2.

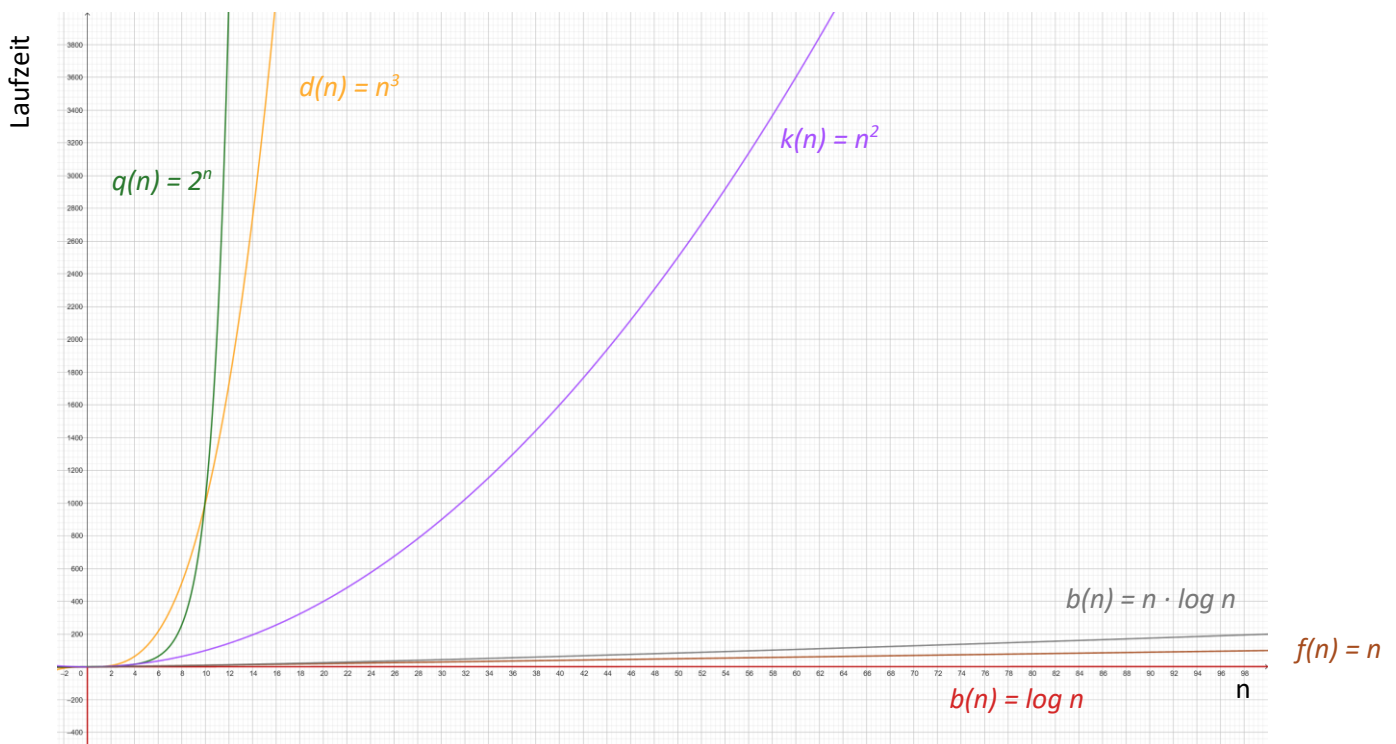


Abbildung 2: Verlauf exemplarischer Funktionsgraphen der verschiedenen Komplexitätsklassen

Man kann eine Komplexitätsklasse auch als obere Grenze der Laufzeit verstehen. Liegt ein Algorithmus z. B. in der Komplexitätsklasse $O(n^2)$, heißt das, dass es eine quadratische Funktion $c \cdot n^2$ mit einem $c \in \mathbb{N}$ gibt, so dass diese ab irgendeiner großen Zahl $m \in \mathbb{N}$ für alle $x \geq m$ schneller wächst als die Laufzeit des Algorithmus. Mit dieser Sichtweise lassen sich die beiden folgenden Regeln für die Zuordnung einer Funktion zu einer Komplexitätsklasse begründen:

- (1) konstante Faktoren können ignoriert werden: Die Laufzeiten $3n$, $5n$ und $1000n$ liegen alle in $O(n)$.
- (2) Bei einer Summe ist der Summand, der am schnellsten wächst, entscheidend:
Die Laufzeiten $3n + n^2$ und $n^2 + 70 \cdot \log(n)$ liegen beide in $O(n^2)$.

Aufgabe 4: Ordnen Sie die folgenden Funktionsgleichungen jeweils der passenden Komplexitätsklasse zu.

Komplexitätsklasse	Funktionsgleichungen
konstant: $O(1)$	$s(n) = 6 \cdot n^3 + 4 \cdot n + 6$ $f(n) = 3 \cdot n + 7$
linear: $O(n)$	$t(n) = 200$ $g(n) = 4 \cdot \log(n)$ $q(n) = 2^n + 5$
logarithmisch: $O(\log n)$	$j(n) = 6n + 2^n$ $k(n) = 6 \cdot n^2 + 3 \cdot n + 7$
quadratisch: $O(n^2)$	$h(n) = \log_2(n)$ $r(n) = 5 \cdot n^6 + 2 \cdot n^2 + 7$
polynomiell: $O(n^k)$ mit $k > 2$	$p(n) = 12 \cdot n + \log(n)$ $v(n) = 3658$
exponentiell: $O(2^n)$	

Aufgabe 5:

- a) Zeigen Sie an einem Beispiel, dass ein Algorithmus mit konstanter Laufzeit, der in $O(1)$ liegt, für kleine Eingaben (z. B. $n = 5$) langsamer sein kann als ein Algorithmus mit linearer Laufzeit, der in $O(n)$ liegt.
- b) Zeigen Sie an einem Beispiel, dass ein Algorithmus, der in $O(2^n)$ liegt, für kleine Eingaben, z. B. $n = 5$ schneller sein kann, als ein Algorithmus, der in $O(n^2)$ liegt.
- c) Begründen Sie, warum bei einem Vergleich von Algorithmen vor allem die Laufzeit für sehr große Eingaben betrachtet wird.

Aufgabe 6: Die Tabelle in Abbildung 3 veranschaulicht, wie stark exemplarische Funktionen der verschiedenen Komplexitätsklassen wachsen. Gehen Sie davon aus, dass eine elementare Operation 1ns ($10^9 \text{ ns} = 1 \text{ s}$) dauert. Markieren Sie die Zellen, bei denen die Ausführung mehr als 1 Stunde dauern würde, also mehr als 3600 Sekunden. Rechnen Sie die Dauer in eine geeignete Einheit um.

Problem- größe n	10	100	1000	10.000	100.000	1.000.000
Laufzeit $f(n)$						
$\log(n)$	1	2	3	4	5	6
n	10	100	1000	10.000	100.000	1.000.000
$n \cdot \log n$	10	200	3.000	40.000	500.000	6.000.000
n^2	100	10.000	1.000.000	100.000.000	10^{10}	10^{12}
n^3	1000	1.000.000	10^9	10^{12}	10^{15}	10^{18}
2^n	1024	10^{30}	$\approx 10^{301}$	$\approx 10^{3010}$		

Abbildung 3: Exemplarische Laufzeiten für verschiedene Problemgrößen und Funktionen

Aufgabe 7: Bei der Einordnung der Laufzeit kann es einen Unterschied zwischen dem besten, dem durchschnittlichen und dem schlechtesten Fall geben.

Untersuchen Sie, was man bei den Suchalgorithmen in Abbildung 1 unter dem besten, dem durchschnittlichen und dem schlechtesten Fall verstehen könnte und in welcher Komplexitätsklasse die Laufzeit in diesen Fällen jeweils liegt.

Aufgabe 8: Der Suchalgorithmus in Abbildung 1 kann auf beliebige Reihungen angewendet werden. Sind die Werte in der Reihung sortiert, kann auch der Algorithmus in Abbildung 4 verwendet werden.

- Dokumentieren Sie den Ablauf des Algorithmus in Abbildung 4 jeweils für die Suche nach den Zahlen 11, 19 und 30 in der global definierten Reihung `zahlen: 11, 13, 14, 17, 19, 20, 25, 27`. Erläutern Sie allgemein die Vorgehensweise des Algorithmus.
- Zeigen Sie, dass der Algorithmus in Abbildung 4 im schlechtesten Fall eine bessere Laufzeit hat als die Algorithmen in Abbildung 1.
- Der Algorithmus in Abbildung 4 wird als binäre Suche bezeichnet, während die Algorithmen in Abbildung 1 eine lineare Suche beschreiben. Erläutern Sie.
- Abbildung 5 zeigt eine rekursive Umsetzung des Algorithmus aus Abbildung 4. Zeigen Sie, dass die rekursive Variante des Algorithmus die gleiche Laufzeitkomplexität hat wie die iterative Variante.

suche(zahl: Ganzzahl): Ganzzahl

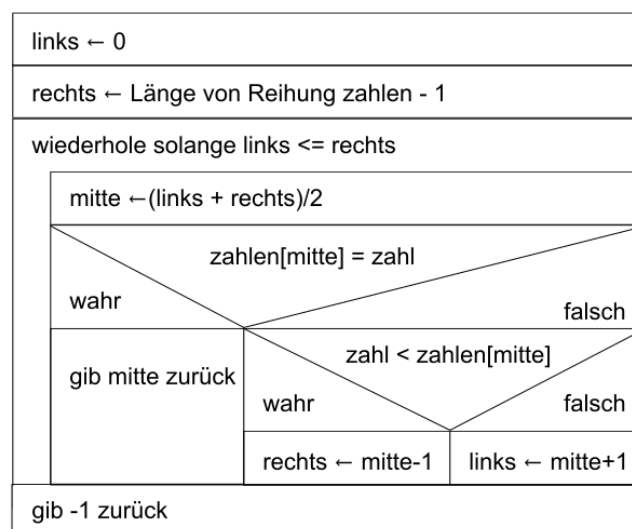


Abbildung 4: Suchalgorithmus für sortierte Reihungen

suche(zahl, links, rechts: Ganzzahl): Ganzzahl

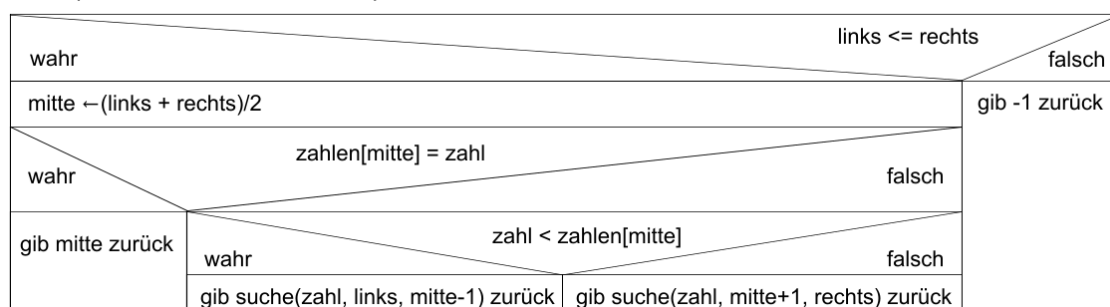


Abbildung 5: rekursive Umsetzung der binären Suche

Laufzeitanalyse von Algorithmen

Für die algorithmischen Grundstrukturen Sequenz, Schleife und Verzweigung kann man Regeln aufstellen, die die Einordnung der Laufzeit eines Algorithmus in eine Komplexitätsklasse erleichtern.

Regeln

1. Zuweisung und Vergleiche

Laufzeit: $O(1)$ Der Aufwand für Zuweisungen und Vergleiche ist konstant.

2. Nacheinanderausführung (Sequenz)

Für die Komplexitätsklassen gilt: $O(1) \subset O(\log(n)) \subset O(n) \subset O(n \cdot \log(n)) \subset \dots \subset O(2^n) \dots$

Deshalb bestimmt bei Hintereinanderausführungen die größte Laufzeitkomponente die Gesamtlaufzeit.

Beispiel: $O(n) + O(2^n) + O(1) \triangleq O(2^n)$

3. Schleifen

Laufzeit := Anzahl der Iterationen \cdot maximale Laufzeit der inneren Anweisung

4. Geschachtelte Schleifen

Laufzeit := Anzahl der Iterationen der äußeren Schleife \cdot Laufzeit der inneren Schleife

5. Verzweigung (if-else-Bedingung)

Laufzeit := Aufwand für den Test + $\max(\text{Aufwand für wahr-Zweig}, \text{Aufwand für falsch-Zweig})$

Aufgabe 9: Abbildung 6 und Abbildung 7 zeigen Ausschnitte aus Algorithmen mit verschiedenen Kontrollstrukturen. Ordnen sie die einzelnen Ausschnitte mithilfe der Regeln 1 bis 5 jeweils in eine Komplexitätsklasse ein. Die Laufzeit soll dabei jeweils in Abhängigkeit des Wertes der Variablen n bzw. m bestimmt werden.

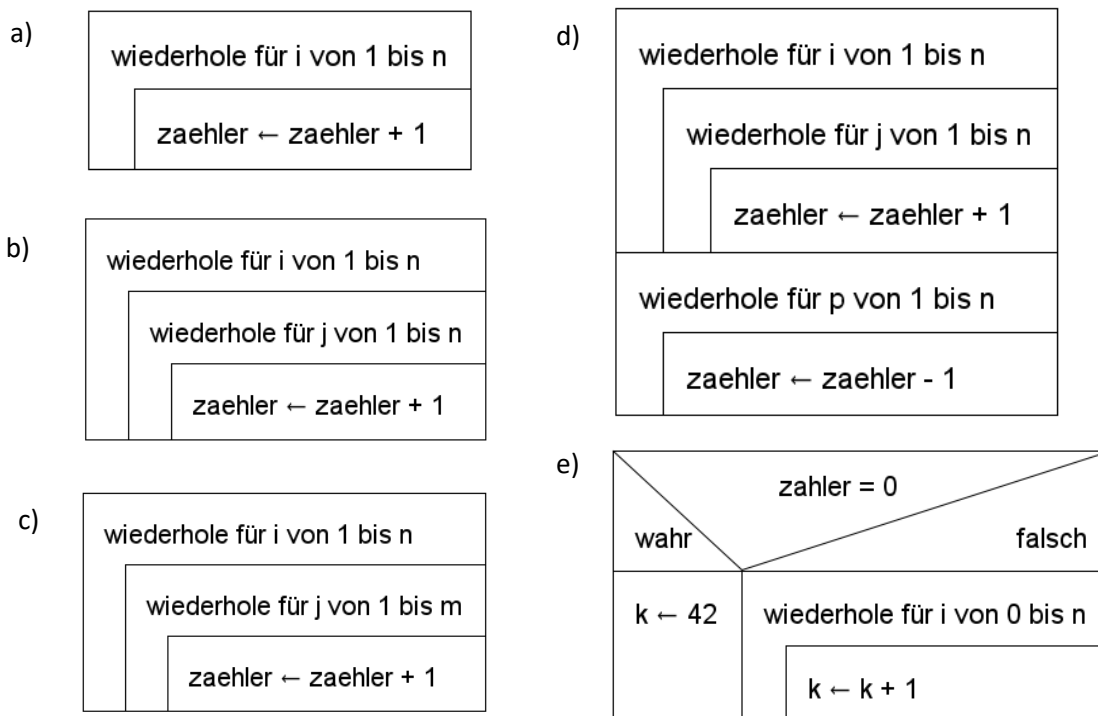


Abbildung 6: einfache Ausschnitte aus Algorithmen

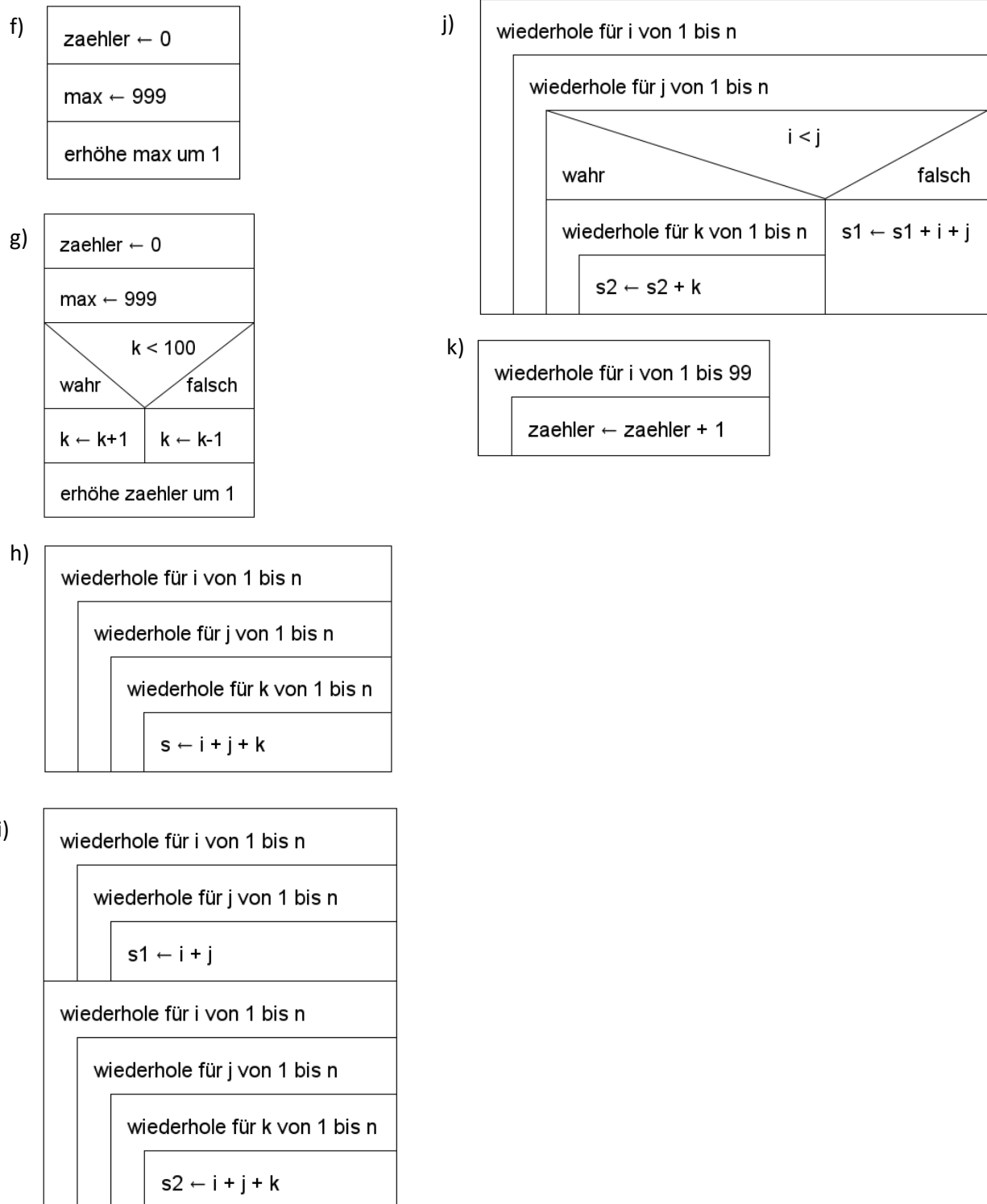


Abbildung 7: komplexere Algorithmenausschnitte

Aufgabe 10: Abbildung 8 und 9 zeigen zwei Algorithmen, die für ein Schwarz-Weiß-Bild die Anzahl der weißen Pixel bestimmen. Die Farben Schwarz und Weiß werden jeweils mit den Ganzzahl-Werten 0 bzw. 1 codiert. Bei dem Algorithmus in Abbildung 8 liegt das Bild zeilenweise in einer eindimensionalen Reihung vom Inhaltstyp Ganzzahl vor. Bei dem Algorithmus in Abbildung 9 liegt das Bild in einer zweidimensionalen Reihung vom Inhaltstyp Ganzzahl vor. Die Dimensionen der Reihung entsprechen der Höhe und Breite des zu verarbeitenden Bildes.

Diskutieren Sie, ob sich die Laufzeit der Algorithmen unterscheidet, wenn Sie jeweils für ein 400 x 300 Pixel großes Bild ausgeführt werden.

zaehleWeiß(pixel: Reihung vom Inhaltstyp Ganzzahl): Ganzzahl

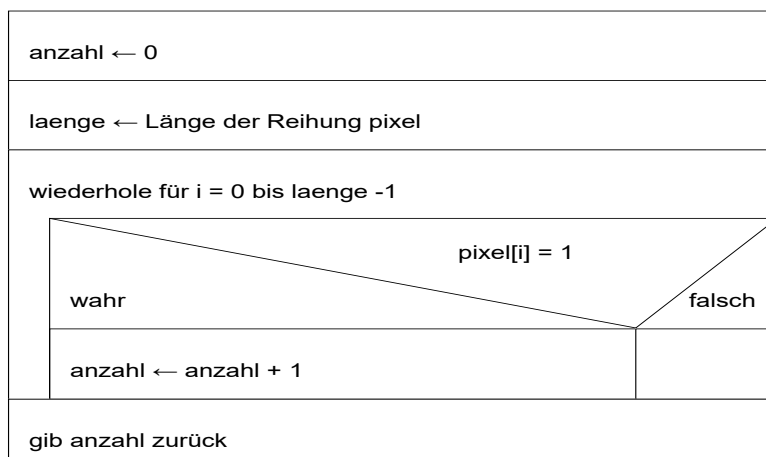


Abbildung 9: Algorithmus zum Zählen der weißen Pixel in einem Bild, das als eindimensionale Reihung vorliegt.

zaehleWeiß(pixel: 2D-Reihung vom Inhaltstyp Ganzzahl): Ganzzahl

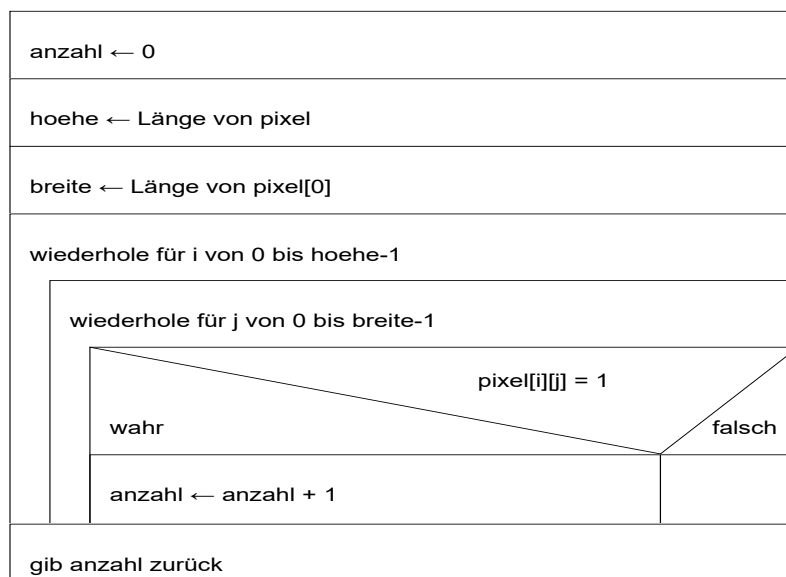


Abbildung 8: Algorithmus zum Zählen der weißen Pixel in einem Bild, das als zweidimensionale Reihung vorliegt.

Aufgabe 11: Wenn es um Sicherheitsfragen geht, kann eine hohe Laufzeitkomplexität von Vorteil sein. Das soll in dieser Aufgabe am Beispiel des Knackens eines Passworts untersucht werden.

Das Struktogramm in Abbildung 10 zeigt zunächst eine rekursive Operation, die systematisch alle Zeichenkombinationen bis zu einer bestimmten Länge erzeugt und auf dem Bildschirm ausgibt. Die Auswahl der möglichen Zeichen ist in einer global definierten Reihung `zeichen` vom Typ Zeichen gegeben. Beim ersten Aufruf der Operation wird die maximale Länge der zu erzeugenden Zeichenkombinationen als Parameter `laenge` übergeben. Der Parameter `anfang` ist beim ersten Aufruf eine leere Zeichenkette.

erzeugeWoerter(anfang: Zeichenkette, laenge: Ganzzahl)

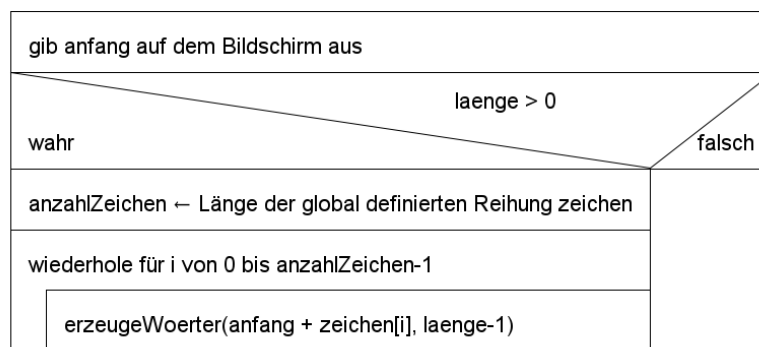


Abbildung 10: Struktogramm der Operation `erzeugeWoerter`

- Geben Sie die rekursiven Aufrufe an, die für den initialen Aufruf `erzeugeWoerter("", 3)` ausgeführt werden, wenn die global definierte Reihung `zeichen` die Zeichen 'A', '1', und '+' enthält. Listen Sie die Zeichenkombinationen auf, die für den initialen Aufruf `erzeugeWoerter("", 3)` ausgegeben werden.
- Erläutern Sie die Arbeitsweise der Operation `erzeugeWoerter`.

Der Algorithmus in Abbildung 10 kann zu einer Operation `knackePasswort` verändert werden, die ein zufällig gewähltes Passwort knackt (s. Abbildung 11). Die Operation `knackePasswort` prüft für jede mögliche Zeichenkombination, ob die Zeichenkette `anfang` das gesuchte Passwort ist. Nur wenn diese Bedingung erfüllt ist, wird die Zeichenkette `anfang` als das gesuchte Passwort ausgegeben.

knackePasswort(anfang: Zeichenkette, laenge: Ganzzahl)

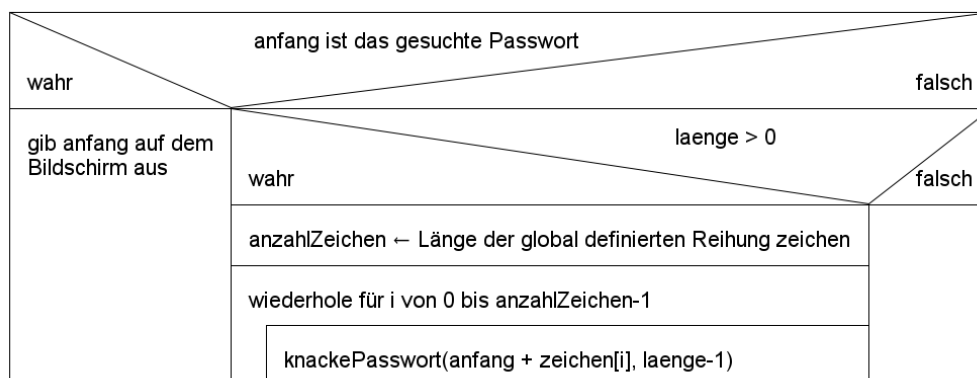


Abbildung 11: Struktogramm der Operation `knackePasswort`

Die Länge eines Passworts und die Einbeziehung von Buchstaben, Ziffern und Sonderzeichen gelten als Kriterien für die Sicherheit eines Passwortes. Verwendet man für das Generieren eines zufälligen Passworts Klein- und Großbuchstaben, die Ziffern 0 bis 9 und typische Sonderzeichen, stehen mindestens 80 verschiedene Zeichen zur Auswahl.

- c) Bestimmen Sie die Laufzeitkomplexität der Operation `knackePasswort` in Abhängigkeit des Parameters `laenge`, wenn sie zum Ausprobieren aller möglichen Passwörter bis zu einer Länge n verwendet wird. Gehen Sie davon aus, dass das Überprüfen der Bedingung, ob der Wert von `anfang` das gesuchte Passwort ist, in konstanter Laufzeit erfolgt und in der global definierten Reihung `zeichen` 80 verschiedene Zeichen enthalten sind.
- d) Diskutieren Sie, ab welcher Länge Sie ein zufälliges Passwort aus 80 möglichen Zeichen als sicher einstufen würden. Wie ändert sich Ihre Einschätzung, wenn das Passwort nur aus Großbuchstaben besteht?

Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz](#). Von der Lizenz ausgenommen ist das InfSII-Logo.

