Übungsaufgabe

Aufwandsabschätzung, Komplexitätstheorie und formale Korrektheitsnachweise von Algorithmen

Universität:	Technische Universität Berlin
Kurs/Modul:	Algorithmen und Datenstrukturen
Erstellungsdatum:	September 6, 2025

Aufgabe 1: Aufwandsabschätzung und Rekursionsanalyse

Betrachten Sie typische Muster der Aufwandsabschätzung in Algorithmen. Behalten Sie im Fokus, wie Schleifen, Rekursionen und Rekursionsglättung zur Komplexität beitragen.

a) Verschachtelte Schleifen:

```
for i := 1 to n do
  for j := 1 to i do
    x := x + 1
```

Bestimmen Sie die Laufzeitklasse dieses Algorithmus (ohne Lösung). Geben Sie eine kurze Begründung an.

b) Rekursiver Aufbau:

$T(n) = 2 \cdot T(n-1) + c, \quad T(1) = d$

Bestimmen Sie Theta(T(n)) bzw. die Theta-Notation der Lösung. Geben Sie die Art der Rekursion an (rein rekursiv, kein Tail-Recursion-Verhalten).

c) Divide-and-Conquer mit Master-Ansatz:

$T(n) = 2 \cdot T(n/2) + c \cdot n, \quad T(1) = d$

Bestimmen Sie Theta(T(n)) und nennen Sie die Master-Theorem-Fallunterscheidung, die zum Ergebnis führt.

Aufgabe 2: Formale Korrektheitsnachweise

Behandeln Sie formale Beweise für Eigenschaften von Algorithmen. Verwenden Sie Invarianten bzw. Beweisstrategien, um Korrektheit und Terminierung zu zeigen.

a) Endlicher Suchalgorithmus – Binary Search: Der folgende Algorithmus wird für eine sortierte Sequenz A[1..n] verwendet:

```
low := 1; high := n
while low  high do
   mid := ⌊(low + high)/2⌋
   if A[mid] = x then return true
   if A[mid] < x then low := mid + 1
   else high := mid - 1
return false
```

Formulieren Sie eine geeignete Schleifeninvariante und begründen Sie damit die Korrektheit des Algorithmus. Zeigen Sie außerdem, dass der Algorithmus terminiert.

b) Maximalauswahl – Invariante und Korrektheit: Betrachten Sie den folgenden Algorithmus zur Bestimmung des Maximums einer Liste L[1..n]:

```
max := L[1]
for i := 2 to n do
   if L[i] > max then max := L[i]
return max
```

Formulieren Sie eine geeignete Invariante für die Schleife und begründen Sie Korrektheit und Terminierung des Algorithmus.

c) Komplexität des In-Place-Maximalwert-Algorithmus: Bestimmen Sie die Worst-Case-Laufzeit dieses Algorithmus in Theta-Notation (Annahme: Vergleichs- und Zuweisungsoperationen kosten eine Konstante). Geben Sie eine kurze Begründung.

Aufgabe 3: Komplexitätstheorie – Modelle, Notationen und Beispiele

Untersuchen Sie exemplarisch gängige Modelle, Notationen und typische Beispiele hinsichtlich Laufzeit.

a) Master-Theorem-Anwendung: Betrachten Sie die Rekursion

$$T(n) = 2\,T(n/2) + \Theta(n).$$

Bestimmen Sie Theta(T(n)) und erläutern Sie kurz die Folgen der Master-Theorem-Fallschilderung.

b) Naive rekursive Fibonacci-Funktion: Betrachten Sie folgende naive Rekursion

fib(n) = 1, für n 1
fib(n) = fib(n-1) + fib(n-2), otherwise

Bestimmen Sie Theta(F(n)) der Laufzeit von fib(n). Geben Sie eine kurze Begründung anhand einer Rekursionsgleichung.

c) Doppelt verschachtelte Schleife mit wachsender Basis: Gegeben sei der Code:

```
for i := 1 to n do
  j := 1
  while j < n do
    j := j * 2
```

Bestimmen Sie die Gesamtlaufzeit in Theta-Notation. Begründen Sie kurz, warum diese Laufzeit entsteht.

Lösungen

Aufgabe 1: Aufwandsabschätzung und Rekursionsanalyse

a) Lösung: Die Schleifenstruktur zählt die Inkremente von x. Die äußere Schleife läuft n Mal; die innere Schleife läuft in Durchgängen i Mal, also insgesamt

$$\sum_{i=1}^{n} i \;=\; \frac{n(n+1)}{2} \;=\; \Theta(n^2).$$

Damit gehört der Algorithmus zur Laufzeitklasse $\Theta(n^2)$.

b) Lösung: Gegeben $T(n) = 2\,T(n-1) + c$ mit $T(1) = d$.

Durch wiederholtes Ausklappen erhalten wir

$$T(n) = 2^{n-1}T(1) + c\sum_{k=0}^{n-2} 2^k = 2^{n-1}d + c(2^{n-1}-1) = (d+c)\,2^{\,n-1} - c.$$

Daraus folgt

$$T(n) = \Theta(2^n),$$

also exponentielle Laufzeit. Die Rekursion ist rein rekursiv (kein Tail-Recursion-Verhalten), da nach dem rekursiven Aufruf weitere Rechenoperationen (hier die Multiplikation mit 2 und das Hinzufügen von c) folgen.

c) Lösung:

$$T(n) = 2\,T(n/2) + c\,n, \quad T(1) = d.$$

Hier sind $a = 2,\ b = 2,\ f(n) = \Theta(n)$ und $n^{\log_b a} = n^{\log_2 2} = n$. Da $f(n) = \Theta\big(n^{\log_b a}\big)$ ist, wendet man Fall 2 des Master-Theorems an und erhält

$$T(n) = \Theta(n \log n).$$

Aufgabe 2: Formale Korrektheitsnachweise

a) Lösung: Schleifeninvariante (Binary Search): Bei jedem Schleifendurchlauf gilt: - Alle Indizes $i < low$ erfüllen $A[i] < x$, - Alle Indizes $i > high$ erfüllen $A[i] > x$, - Falls x in der Sequenz enthalten ist, liegt x im Intervall $[low, high]$.

Initialisierung: $low := 1$, $high := n$ erfüllt die Invariante, da ansonsten der Bereich leer ist.

Aufrechterhaltung: - Falls $A[\text{mid}] = x$ entdeckt wird, wird die Suche beendet. - Falls $A[\text{mid}] < x$, bleibt x nur in $A[\text{mid} + 1..high]$; entsprechend wird $low := \text{mid} + 1$ gesetzt. - Falls $A[\text{mid}] > x$, bleibt x nur in $A[low..\text{mid} - 1]$; entsprechend wird $high := \text{mid} - 1$ gesetzt. In allen Fällen bleibt die Invariante erhalten.

Terminierung: Die Schlauchgröße $high - low + 1$ wird in jeder Iteration reduziert, da $\text{mid} = \lfloor(\text{low} + \text{high})/2\rfloor$ gewählt wird und entweder low oder high näher aneinander rückt. Damit endet der Algorithmus, wenn $low > high$ und damit die Bedingung verloren geht.

Begründung der Korrektheit: Wenn x in der Sequenz enthalten ist, bleibt gemäß der Invariante der Indexbereich, in dem sich x befinden muss, während der Suche erhalten; schließlich wird der Index von x gefunden, oder bei leerem Intervall wird x nicht gefunden.

b) Lösung: Schleifeninvariante: Nach jedem Durchlauf der Schleife gilt

$$\max\{\ L[1],\ L[2],\ \ldots,\ L[i]\ \} = \text{maximaler Wert der ersten } i \text{ Elemente.}$$

Initialisierung: Vor dem ersten Durchlauf gilt $i = 1$ und $\max = L[1]$, also die Invariante.

Aufrechterhaltung: Angenommen, nach Verarbeitung von $i-1$ gilt $\max = \max\{L[1], \ldots, L[i-1]\}$. Beim Schritt i wird max mit $L[i]$ verglichen; falls $L[i] > \max$, wird max entsprechend aktualisiert. Dann gilt

$$\max = \max\{L[1], \ldots, L[i]\}.$$

Terminierung: Nach Schluss der Schleife (bei $i = n$) enthält max den Maximalwert der gesamten Liste.

Korrektheit: Durch Induktionsbeweis der Invariante folgt, dass am Ende das Maximum der gesamten Liste in max steht.

c) Lösung: Worst-Case-Laufzeit des in-place-Maximalwert-Algorithmus. Es wird eine Schleife über alle n Elemente durchlaufen, wobei pro Iteration eine Konstante (Vergleich) durchgeführt wird und ggf. eine Zuweisung erfolgt. Daraus folgt:

$$\text{Laufzeit } = \Theta(n).$$

Aufgabe 3: Komplexitätstheorie – Modelle, Notationen und Beispiele

a) Lösung: Für die Rekursion

$$T(n) = 2\,T(n/2) + \Theta(n)$$

gilt nach dem Master-Theorem mit $a = 2,\ b = 2,\ f(n) = \Theta(n)$ und $n^{\log_b a} = n$. Damit ist Fall 2 erfüllt und

$$T(n) = \Theta(n \log n).$$

b) Lösung: Der naive Fibonacci-Algorithmus hat

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \quad (n > 1), \quad \text{fib}(0) = \text{fib}(1) = 1.$$

Die Rekursion führt zu einer Anzahl von Funktionsaufrufen, die exponentiell wächst. Genauer gilt

$$T(n) = T(n-1) + T(n-2) + O(1) = \Theta(\phi^n),$$

mit $\phi = \frac{1+\sqrt{5}}{2}$ (die dominierende Wurzel der charakteristischen Gleichung $r^n = r^{n-1} + r^{n-2}$). Somit ist die Laufzeit exponential, Theta(ϕ^n).

c) Lösung: Gegeben sei der Code

```
for i := 1 to n do
  j := 1
  while j < n do
    j := j * 2
```

Für jedes i durchläuft die innere Schleife circa $\lceil \log_2 n \rceil$ Mal, da j in jeder Iteration verdoppelt wird, bis $j \geq n$. Daher ist die Gesamtlaufzeit

$$\Theta\big(n \log n\big).$$