

# Lösungen: Algorithmen und Datenstrukturen - 09.02.2024

Ausgangsdatei: 41-Klausur vom 09.02.2024\_final.pdf

Diese Ausarbeitung enthält vollständige Lösungen mit Begründungen zu allen sechs Aufgaben. Es werden keine zusätzlichen Java-Bibliotheksklassen benötigt. Die Bonuspunkte auf dem Deckblatt gehören in der vorliegenden PDF zu keiner gesonderten Aufgabenstellung.

## Aufgabe 1 - Wissen und Multiple Choice

Antwortnummern beziehen sich auf die Reihenfolge in der Vorlage. Bei Aussagen über asymptotische Geschwindigkeit wird die Wachstumsklasse verglichen, nicht die tatsächliche Laufzeit auf einem konkreten Rechner.

### a) Begriff Algorithmus

Anzukreuzen sind 2 und 3.

| Nr. | Bewertung                      | Begründung                                                                                                                                      |
|-----|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | Falsch                         | Eine konkrete Programmiersprache ist nicht erforderlich; ein Algorithmus kann sprachunabhängig beschrieben werden.                              |
| 2   | Richtig                        | Der Algorithmus löst eine Problemklasse und nicht nur eine einzelne Eingabe.                                                                    |
| 3   | Richtig                        | In der hier verwendeten einführenden Definition terminiert er für jede zulässige Eingabe mit einem Ergebnis.                                    |
| 4   | Falsch                         | Die Beschreibung muss präzise und ausführbar sein, nicht vage.                                                                                  |
| 5   | Falsch                         | Ein korrekter Algorithmus muss nicht die bestmögliche Effizienz besitzen.                                                                       |
| 6   | Nicht die gesuchte Eigenschaft | Eine gesonderte Zerlegung unmittelbar vor der Ausführung ist kein Definitionsmerkmal; elementare ausführbare Schritte gehören zur Beschreibung. |

### b) Asymptotische Laufzeit von Sortierverfahren

Anzukreuzen sind 4 und 6.

| Nr. | Aussage in Kurzform                                    | Bewertung und Grund                                           |
|-----|--------------------------------------------------------|---------------------------------------------------------------|
| 1   | Bubblesort im Average Case schneller als Mergesort     | Falsch: $\Theta(n^2)$ wächst stärker als $\Theta(n \log n)$ . |
| 2   | Insertionsort im Average Case schneller als Bubblesort | Falsch als asymptotische Aussage: Beide haben $\Theta(n^2)$ . |

| Nr. | Aussage in Kurzform                                      | Bewertung und Grund                                                |
|-----|----------------------------------------------------------|--------------------------------------------------------------------|
| 3   | Quicksort im Average Case schneller als Heapsort         | Falsch als asymptotische Aussage: Beide haben $\Theta(n \log n)$ . |
| 4   | Bottom-Up-Heapsort im Worst Case schneller als Quicksort | Richtig: $\Theta(n \log n)$ gegenüber $\Theta(n^2)$ .              |
| 5   | Heapsort im Worst Case schneller als Mergesort           | Falsch als asymptotische Aussage: Beide haben $\Theta(n \log n)$ . |
| 6   | Quicksort und Bubblesort im Worst Case gleich schnell    | Richtig im Sinne der Wachstumsklasse: Beide haben $\Theta(n^2)$ .  |

### c) Suchverfahren

Die beabsichtigten zwei Antworten sind 2 und 6: Binäre Suche auf sortierten Schlüsseln benötigt im schlechtesten Fall  $\Theta(\log_2 n)$ ; lineare Suche benötigt auch bei sortierten Schlüsseln im schlechtesten Fall  $\Theta(n)$ .

| Nr. | Präzise Einordnung                                                                                                                              |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | Die enge mittlere Schranke der binären Suche ist unter der üblichen Verteilung der Suchfälle $\Theta(\log n)$ , nicht $\Theta(n^2)$ .           |
| 2   | Beabsichtigte richtige Antwort: Jeder Vergleich halbiert den Suchbereich.                                                                       |
| 3   | Interpolationssuche hat im Worst Case $\Theta(n)$ , nicht $\Theta(n^2)$ .                                                                       |
| 4   | Bei geeigneter Gleichverteilung gilt für Interpolationssuche im Average Case $\Theta(\log \log n)$ , nicht als enge Schranke $\Theta(\log n)$ . |
| 5   | Falsch: Eine lineare Suche kann im Worst Case $n$ Prüfungen benötigen; das ist nicht $O(\log n)$ .                                              |
| 6   | Beabsichtigte richtige Antwort: Der gesuchte Schlüssel kann am Ende stehen oder erst nach dem letzten Element ausgeschlossen werden.            |

Die Aufgabe ist bei wörtlicher mathematischer Lesart mehrdeutig:  $O$  ist nur eine obere Schranke. Damit sind zusätzlich 1 und 3 als  $O(n^2)$ -Obergrenzen sowie 4 als  $O(\log n)$ -Obergrenze ebenfalls wahr. „Genau zwei Antworten“ passt daher nur zur offenbar gemeinten Auswahl enger asymptotischer Klassen. Für die Klausurkonvention sind 2 und 6 die gesuchten Kreuze; die anderen genannten Obergrenzen sind mathematisch nicht falsch, sondern ungenau.

## d) Rekursive und iterative Implementierung

| Nr. | Aussage in Kurzform                                          | Richtig/falsch | Begründung                                                                                  |
|-----|--------------------------------------------------------------|----------------|---------------------------------------------------------------------------------------------|
| 1   | Fakultät nur iterativ, nicht rekursiv                        | Falsch         | Rekursiv: $\text{fak}(0) = 1$ , $\text{fak}(n) = n \cdot \text{fak}(n - 1)$ .               |
| 2   | Heapsort iterativ und rekursiv möglich                       | Richtig        | Insbesondere das Wiederherstellen der Heapordnung kann als Schleife oder rekursiv erfolgen. |
| 3   | Lineare Suche iterativ und rekursiv möglich                  | Richtig        | Entweder eine Schleife oder ein rekursiver Aufruf für die restliche Folge.                  |
| 4   | Mergesort rekursiv, aber nicht iterativ möglich              | Falsch         | Bottom-Up-Mergesort vereinigt iterativ Läufe der Länge 1, 2, 4, ...                         |
| 5   | Postorder-Traversierung rekursiv und iterativ möglich        | Richtig        | Die iterative Form verwaltet ausstehende Besuche mit einem expliziten Stapel.               |
| 6   | Preorder-Traversierung rekursiv, aber nicht iterativ möglich | Falsch         | Ein expliziter Stapel ersetzt den Rekursionsstapel auch bei Preorder.                       |

## Aufgabe 2 - Komplexität und Rekursion

### a) Aufrufzahlen und Laufzeiten

Für `proz1` sei  $n \geq 1$ . Für `proz2` sei  $n$  eine positive Zweierpotenz.

| Methode            | Aufrufe bei $n = 4$ | Allgemeine Anzahl der <code>tuwas</code> -Aufrufe | Zeitkomplexität |
|--------------------|---------------------|---------------------------------------------------|-----------------|
| <code>proz1</code> | 21                  | $1 + n + n^2$                                     | $O(n^2)$        |
| <code>proz2</code> | 2                   | $\log_2 n$                                        | $O(\log n)$     |

Bei `proz1` gibt es zunächst einen Aufruf, dann  $n$  weitere Aufrufe direkt in der äußeren Schleife und  $n \cdot n$  Aufrufe in der inneren Schleife. Für  $n = 4$  ergibt sich  $1 + 4 + 16 = 21$ .

Bei `proz2` gibt es pro Halbierung genau einen Aufruf von `tuwas`. Für  $n = 4$  geschieht dies bei  $n = 4$  und  $n = 2$ ; bei  $n = 1$  ist der Basisfall erreicht. Allgemein gilt  $A(1) = 0$  und  $A(n) = 1 + A(n/2)$ , also  $A(n) = \log_2 n$ . Da `tuwas`  $O(1)$  kostet, entspricht die Laufzeit jeweils der Größenordnung der Schleifen beziehungsweise Aufrufe.

### b) Maximale Rekursionstiefe für $n = 4$

Die Aufrufkette lautet `proz2(4) → proz2(2) → proz2(1)`. Das sind 3 gleichzeitig aktive Methodenaufrufe oder, wenn der Erstaufruf nicht mitgezählt wird, 2 rekursive Abstiege.

### c) Rekursionstiefe allgemein

Für  $n = 2^k$  wird nach  $k$  Halbierungen der Wert 1 erreicht. Die maximale Zahl gleichzeitig aktiver Aufrufe ist  $\log_2 n + 1$ ; die Zahl der Abstiege ist  $\log_2 n$ . Der Zusatzspeicher ist  $O(\log n)$ .

## Aufgabe 3 - Auftragsstapel

### a) push mit Entfernen eines bereits enthaltenen Auftrags

```
public void push(Auftrag a) {
    assert(a != null);
    if (oben != null && oben.daten.equals(a)) {
        oben = oben.naechster;
    } else if (oben != null) {
        Link<Auftrag> vorgaenger = oben;
        while (vorgaenger.naechster != null
            && !vorgaenger.naechster.daten.equals(a)) {
            vorgaenger = vorgaenger.naechster;
        }
        if (vorgaenger.naechster != null) {
            vorgaenger.naechster =
                vorgaenger.naechster.naechster;
        }
    }
    oben = new Link<Auftrag>(a, oben);
}
```

Zunächst wird ein inhaltlich gleicher Auftrag an seiner bisherigen Stelle entfernt. Liegt er am Kopf, wird oben weitergesetzt; sonst wird der entsprechende Knoten über seinen Vorgänger ausgekettet. Anschließend wird der übergebene Auftrag immer oben abgelegt.

Verglichen wird mit der vorgegebenen equals(Auftrag)-Methode, die sowohl id als auch bezeichnung prüft. Ein bloßer Referenzvergleich würde zwei verschiedene Objekte mit denselben Auftragsdaten übersehen. Da der Stapel leer beginnt und nur diese push-Methode Elemente hinzufügt, kann von jedem Auftrag höchstens ein gleiches Element enthalten sein; das Entfernen des ersten Treffers reicht daher aus. Laufzeit  $O(n)$ .

### b) pop

```
public Auftrag pop() {
    if (oben == null) {
        return null;
    }
    Auftrag ergebnis = oben.daten;
    oben = oben.naechster;
    return ergebnis;
}
```

Der oberste Auftrag wird gesichert, der Kopf auf den Nachfolger verschoben und der Auftrag zurückgegeben. Damit gilt das LIFO-Prinzip: Der zuletzt nach oben gelegte Auftrag wird zuerst entfernt. Bei leerem Stapel wird null geliefert. Aufwand:  $O(1)$ .

## Aufgabe 4 - Binäre Suchbäume

### a) Iterativ prüfen, ob der Baum eine lineare Liste ist

```
public boolean istLineareListe() {
    Knoten aktuell = wurzel;
    while (aktuell != null) {
        if (aktuell.linkesKind != null
            && aktuell.rechtesKind != null) {
            return false;
        }
        if (aktuell.linkesKind != null) {
            aktuell = aktuell.linkesKind;
        } else {
            aktuell = aktuell.rechtesKind;
        }
    }
    return true;
}
```

Ein zu einer Liste entarteter Baum besitzt an jedem Knoten höchstens ein Kind. Dieses Kind darf links oder rechts liegen; ein Wechsel der Richtung ist zulässig. Sobald ein Knoten zwei Kinder besitzt, ist die Listenform widerlegt. Bei einem leeren Baum wird die Schleife nicht betreten und true zurückgegeben. Laufzeit im schlechtesten Fall  $O(n)$ , zusätzlicher Speicher  $O(1)$ .

### b) Schlüssel absteigend als Zeichenkette

```
private String absteigend(Knoten k) {
    if (k == null) {
        return "";
    }
    String ergebnis = "";
    if (k.rechtesKind != null) {
        ergebnis = absteigend(k.rechtesKind) + " ";
    }
    ergebnis += k.schluessel;
    if (k.linkesKind != null) {
        ergebnis += " " + absteigend(k.linkesKind);
    }
    return ergebnis;
}
```

Im Suchbaum stehen rechts die größeren, links die kleineren Schlüssel. Die umgekehrte Inorder-Reihenfolge „rechter Teilbaum, Wurzel, linker Teilbaum“ liefert daher die Schlüssel absteigend. Leerzeichen werden nur zwischen tatsächlich vorhandenen Teilfolgen gesetzt; es gibt weder ein führendes noch ein abschließendes Leerzeichen. Der leere Baum liefert den leeren String.

Beispiel: Eine Wurzel 5 mit linkem Kind 2 und rechtem Kind 8 ergibt 8 5 2. Der Baum wird einmal durchlaufen; wiederholtes Kopieren unveränderlicher Strings kann aber insbesondere bei einer langen Kette insgesamt quadratischen Zeichenkopieraufwand verursachen. Die geforderte Lösung benötigt keine weitere Bibliotheksklasse.

## Aufgabe 5 - Gnome-Sort

### a) Vollständige Ablaufspur

Jede Zeile nach „Start“ zeigt den Zustand am Ende genau einer Iteration der while-Schleife. Auch Durchläufe ohne Vertauschung sind enthalten.

| Iteration | i danach | array danach   | Erklärung                       |
|-----------|----------|----------------|---------------------------------|
| Start     | 0        | 65, 34, 23, 43 | Initialisierung                 |
| 1         | 1        | 65, 34, 23, 43 | $i = 0$ , daher vorwärts        |
| 2         | 0        | 34, 65, 23, 43 | $65 > 34$ : tauschen und zurück |
| 3         | 1        | 34, 65, 23, 43 | $i = 0$ , daher vorwärts        |
| 4         | 2        | 34, 65, 23, 43 | $34 \leq 65$ , daher vorwärts   |
| 5         | 1        | 34, 23, 65, 43 | $65 > 23$ : tauschen und zurück |
| 6         | 0        | 23, 34, 65, 43 | $34 > 23$ : tauschen und zurück |
| 7         | 1        | 23, 34, 65, 43 | $i = 0$ , daher vorwärts        |
| 8         | 2        | 23, 34, 65, 43 | $23 \leq 34$ , daher vorwärts   |
| 9         | 3        | 23, 34, 65, 43 | $34 \leq 65$ , daher vorwärts   |
| 10        | 2        | 23, 34, 43, 65 | $65 > 43$ : tauschen und zurück |
| 11        | 3        | 23, 34, 43, 65 | $34 \leq 43$ , daher vorwärts   |
| 12        | 4        | 23, 34, 43, 65 | $43 \leq 65$ , daher vorwärts   |

Nun ist  $i = \text{array.length} = 4$ ; die while-Schleife endet. Wegen der Kurzschlussauswertung von  $||$  wird bei  $i = 0$  niemals  $\text{array}[-1]$  gelesen.

### b) Asymptotische Anzahl der Vergleiche

Bester Fall:  $O(n)$ . Ist das Feld bereits aufsteigend sortiert, erhöht sich  $i$  in jedem Durchlauf. Es gibt  $n - 1$  Schlüsselvergleiche und keine Vertauschung.

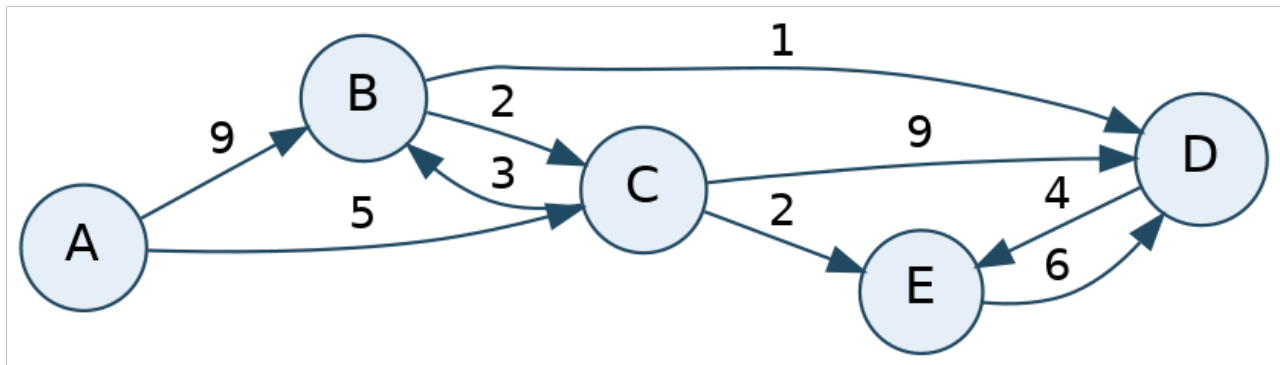
Schlechtester Fall:  $O(n^2)$ . Ein rückwärts sortiertes Feld mit verschiedenen Schlüsseln besitzt  $n(n - 1)/2$  Inversionen. Jeder Nachbartausch beseitigt genau eine solche Fehlordnung und verursacht Rückwärtsbewegungen mit erneuten Prüfungen. Damit sind quadratisch viele Vergleiche nötig.

## Aufgabe 6 - Graphen und kürzeste Wege

### a) Gerichtet oder ungerichtet

Der Graph ist gerichtet. Die Matrix ist nicht symmetrisch:  $A \rightarrow B$  mit Gewicht 9 existiert, aber  $B \rightarrow A$  fehlt. Außerdem haben  $B \rightarrow C$  und  $C \rightarrow B$  unterschiedliche Gewichte 2 beziehungsweise 3. Die Zahlen sind Kantengewichte.

### b) Graph mit allen Gewichten



Alle neun Kanten tragen ihre Gewichte. Gegenläufige Pfeile bleiben getrennt, weil sie verschiedene Gewichte besitzen.

### c) Dijkstra mit Startknoten A

Zellen: Entfernung/Vorgänger;  $\infty$  = noch unerreichbar, - = kein Vorgänger. Jede Zeile gilt nach dem Markieren und Entspannen der ausgehenden Kanten.

| Neu markierter Knoten | A   | B           | C           | D           | E           |
|-----------------------|-----|-------------|-------------|-------------|-------------|
| Initialisierung       | 0/- | $\infty$ /- | $\infty$ /- | $\infty$ /- | $\infty$ /- |
| A                     | 0/- | 9/A         | 5/A         | $\infty$ /- | $\infty$ /- |
| C                     | 0/- | 8/C         | 5/A         | 14/C        | 7/C         |
| E                     | 0/- | 8/C         | 5/A         | 13/E        | 7/C         |
| B                     | 0/- | 8/C         | 5/A         | 9/B         | 7/C         |
| D                     | 0/- | 8/C         | 5/A         | 9/B         | 7/C         |

Die Markierungsreihenfolge ist A, C, E, B, D. Über C verbessern sich B auf  $5 + 3 = 8$  und E auf  $5 + 2 = 7$ ; D erhält zunächst  $5 + 9 = 14$ . Über E sinkt die Entfernung nach D auf  $7 + 6 = 13$  und über B schließlich auf  $8 + 1 = 9$ . Alle Kantengewichte sind positiv, daher sind die Voraussetzungen von Dijkstra erfüllt.

### d) Kürzester Weg von A nach D

Die letzte Tabellenzeile liefert rückwärts  $D \leftarrow B \leftarrow C \leftarrow A$ . Umgekehrt ergibt dies  $A \rightarrow C \rightarrow B \rightarrow D$  mit Länge  $5 + 3 + 1 = 9$ , passend zum endgültigen Distanzwert für D.