

Lösungsvorschläge zur Probeklausur zu Algorithmen und Datenstrukturen

**Aufgabe 1 Rechnen mit Landau-Symbolen**

(4 Punkte)

Untersuchen Sie, ob folgende Behauptung wahr ist und zeigen Sie Ihre Aussage.

$$\frac{1}{2}n^2 - 3n + 4 = O(n^3)$$

*Einsetzen in die Ungleichung aus der Definition von  $O(n^3)$ :*

$$\frac{1}{2}n^2 - 3n + 4 \leq c \cdot n^3$$

*Teilung durch  $n^3 > 0$  (für  $n > 0$ ):*

$$\frac{1/2}{n} - \frac{3}{n^2} + \frac{4}{n^3} \leq c$$

*Die Folgen  $1/n^i$  konvergieren für alle  $n > n_0 = 1$  gegen 0. Wähle zusätzlich  $c$  hinreichend groß, etwa  $c \geq 1.5$  ( $= \frac{1}{2} - 3 + 4$ ), so gilt die Ungleichung, und die Behauptung ist gezeigt.  $\square$*

**Bewertung:** 1 Punkt für Ungleichung von  $O(n^3)$ , 1 Punkt für Umformen der Ungleichung, 1 Punkt für eine korrekte Konstante, 1 Punkt für korrekte Schlussfolgerung.

**Aufgabe 2 Boolesche Logik**

(4 Punkte)

Zeigen Sie, dass der logische Ausdruck  $a \Rightarrow b$  identisch ist mit  $\neg a \vee b$ .

*Der Beweis kann etwa über eine Wahrheitstabelle erledigt werden:*

$a$	$b$	$\neg a$	$\neg a \vee b$	$a \Rightarrow b$
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0
1	1	0	1	1

*Die den beiden Seiten der Gleichung entsprechenden Spalten sind gleich, damit ist die Behauptung gezeigt.  $\square$*

**Bewertung:** 1 Punkt für Aufstellung der Wahrheitstabelle mit  $a, b$ , 1 Punkt für Spalte  $\neg a$ , 1 Punkt für Spalte  $\neg a \vee b$ , 1 Punkt für Spalte  $a \Rightarrow b$ .

**Aufgabe 3 Zahldarstellung**

(4 Punkte)

Konvertieren Sie die Zahl  $7_{10}$  in eine 4-bit Binärzahl. Berechnen Sie dann das 2-Komplement dieser 4-bit Binärzahl. Warum lässt sich das entstandene Bitmuster in der 2-Komplement-Darstellung als negative Zahl interpretieren?

*Konversion in Binärzahl:*

$$7_{10} = 4_{10} + 2_{10} + 1_{10} = 0111_2$$

Zweierkomplement ist bitweise Inversion und Addition von 1:

$$\overline{0111}_2 + 1_2 = 1000_2 + 1_2 = 1001_2$$

Das erste Bit stellt das Vorzeichen dar:  $1001_2$  kann interpretiert werden als

- $0111_2$  ist positiv (wegen 0): also  $7_{10}$
- $1001_2$  ist negativ (wegen 1), also  $-7_{10}$

**Bewertung:** 1 Punkt für korrekte 4-bit Binärzahl, 1 Punkt für korrektes 2-Komplement, jeweils 1 Punkt für korrekte Interpretation des Bitmusters.

#### Aufgabe 4 Sequentielle Liste

(2 Punkte)

Gegeben sei ein Feld  $A$  implementiert als sequentielle Liste (Array). Was ist die Komplexität der Einfüge-Operation `insert` und der Zugriffs-Operation `elementAt`?

*insert* : Man muss Elemente hinter der Einfügeposition nach hinten verschieben, diese Operation hat die Komplexität  $O(n)$ .

*elementAt* : In einer sequentiellen Liste hat man direkten Zugriff auf die einzelnen Elemente, daher ist die Komplexität  $O(1)$ .

**Bewertung:** Jeweils 1 Punkt für korrekte Komplexität.

#### Aufgabe 5 Stack

(3 Punkte)

Nennen Sie kurz *eine* Methode, um den abstrakten Datentyp Stack mittels elementarer Datenstrukturen zu implementieren. Was sind dann jeweils die Komplexitäten der Operation `push` und `pop`?

**Möglichkeit 1:** Einfach verkettete Liste. Der Zeiger *start* zeigt hierbei auf das oberste Element des Stacks.

**Möglichkeit 2:** Sequentielle Liste mit Index-Variable.

Komplexität von `push` und `pop` bei beiden Möglichkeiten jeweils  $O(1)$ , da die Liste nie durchlaufen werden muss.

**Bewertung:** 1 Punkt für korrekte Implementierung von Stack mittels elementarer Datenstruktur, jeweils 1 Punkt für korrekte Komplexitäten.

#### Aufgabe 6 Fibonacci-Folge, Verifikation

(6 Punkte)

Die Fibonacci-Folge  $(f_n)_{n \in \mathbb{N}}$  ist rekursiv definiert als:

$$\begin{aligned} f_1 &= f_2 = 1 \\ f_n &= f_{n-1} + f_{n-2} \end{aligned}$$

Nachfolgend ist die aus der Vorlesung bekannte iterative Implementierung nach dem Prinzip der dynamischen Programmierung angegeben als Funktion **fib**( $n$ ).

**fib**( $n$ ) :

*fib* = leeres Feld;

*fib*[1] = 1; *fib*[2] = 1;  $k = 3$ ;

$\{C_1 := P \wedge (k = 3)\}$

**while** ( $k \leq n$ ) {

$\{C_2 := P \wedge (3 \leq k \leq n)\}$

```
    fib[k] = fib[k-1] + fib[k-2];
    k = k + 1;
{C3 := P}
}
{C4 := P ∧ (k > n) = NACH}
return fib[n];
```

a) Geben Sie jeweils eine geeignete Vor- und Nachbedingung der Funktion **fib**(*n*) an!

$$\begin{aligned} \text{VOR} &= (n > 0) \\ \text{NACH} &= (f_n = \text{fib}[n]) \end{aligned}$$

**Bewertung:** 1 Punkt für korrekte Vorbedingung, 1 Punkt für Nachbedingung.

b) Geben Sie eine geeignete Schleifeninvariante *P* für die **while** (*k* ≤ *n*) { ... } Schleife an!

$$P = (f_{k-1} = \text{fib}[k-1])$$

**Ausführlicher:** Die Idee hinter dieser Invariante ist, dass bis zum Index *k* (exklusiv) die korrekten Werte in *fib* stehen müssen (vgl. Entwurfsprinzip!), so dass darauf aufbauend dann die nächsten Werte (ab *k*) iterativ berechnet werden können. Wichtig ist, dass die „mathematische Wahrheit“  $f_{k-1}$  in Bezug gesetzt wird zur Berechnung  $\text{fib}[k-1]$ .

**Bewertung:** 1 Punkt für korrekte Invariante.

c) Welche drei Bedingungen müssen überprüft werden, damit die Korrektheit der **while** Schleife mittels der Invariante *P* gezeigt werden kann? Zeigen Sie, dass diese drei Bedingungen in diesem Fall erfüllt sind!

Zu zeigen sind:

- **Schleifeneintritt:**  $C_1 \Rightarrow P$

Trivial, Definition von  $f_1$  und  $f_2$ .

**Ausführlicher:** Manuell wird gesetzt:  $\text{fib}[1] = 1$  und  $\text{fib}[2] = 1$ . Das entspricht genau der Definition von  $f_n$ , also  $\text{fib}[1] = f_1$  und  $\text{fib}[2] = f_2$ . Außerdem setzt man  $k = 3$ , und die Schleifeninvariante *P* (siehe oben) gilt damit für  $k-1$ .

- **Schleifenkörper:**  $\{C_2 := P \wedge (k \leq n)\} \dots \{P\}$

Update wie Rekursionsvorschrift, *P* garantiert richtige Werte im Array.

**Ausführlicher:** Wie beim Induktionsbeweis wird angenommen, dass *P* am Anfang der Schleife gilt. Explizit also gilt  $\text{fib}[l] = f_l$  für alle  $l < k$ . Die erste Zeile des Schleifenkörpers führt also zu:

$$\text{fib}[k] = \text{fib}[k-1] + \text{fib}[k-2] = f_{k-1} + f_{k-2} = f_k$$

Die letzte Gleichheit folgt aus der mathematischen Formel für die Folge. Durch den Schritt  $k \rightarrow k+1$  folgt wieder *P* (das ja mit  $k-1$  geschrieben wurde).

- **Schleifenende:**  $(C_4 := P \wedge \neg(k \leq n)) \Rightarrow \text{NACH}$

*P* garantiert richtige Werte im Array,  $k > n$ , also  $\text{fib}[n] = f_n$ , also *NACH*.

**Ausführlicher:** Durch die verletzte Schleifenbedingung folgt beim Abbruch  $k = n+1$ , eingefügt in *P* (mit  $k-1$ ) folgt *NACH* (mit *n*).

**Bewertung:** 1 Punkt jeweils für Bedingung und Anwendung.

- a) Geben Sie einen Sortier-Algorithmus an, der im Mittel mit Komplexität  $O(n \log n)$  sortiert! Nach welchem Algorithmen-Muster wurde dieser Algorithmus entworfen?

*Divide and Conquer: Merge oder Quick Sort*

*Bewertung: 1 Punkt für Merge oder Quick Sort, 1 Punkt für Divide and Conquer.*

- b) Gegeben sei eine bereits sortierte Liste von natürlichen Zahlen der Länge  $n$ . In diese Liste soll nun ein weiteres Element einsortiert werden. Schlagen Sie einen möglichst effizienten Algorithmus für diese Aufgabe vor und geben Sie dessen Komplexität an.

*Insertion Sort ohne die äußere Schleife, damit dann  $O(n)$ .*

*Bewertung: 1 Punkt für Insertion Sort, 1 Punkt für Komplexität.*

- c) Skizzieren Sie (durch Angabe des Zustandes nach jedem Durchlauf der Hauptschleife), wie der *Selection Sort* die folgende Zahlenreihe sortieren würde!

5 , 1 , 9 , 3 , 8 , 6 , 4 , 7

1 | 5 , 9 , 3 , 8 , 6 , 4 , 7

1 , 3 | 9 , 5 , 8 , 6 , 4 , 7

1 , 3 , 4 | 5 , 8 , 6 , 9 , 7

1 , 3 , 4 , 5 | 8 , 6 , 9 , 7

1 , 3 , 4 , 5 , 6 | 8 , 9 , 7

1 , 3 , 4 , 5 , 6 , 7 | 9 , 8

1 , 3 , 4 , 5 , 6 , 7 , 8 | 9

1 , 3 , 4 , 5 , 6 , 7 , 8 , 9

*Bewertung: 0,5 Punkte pro Schritt, letzter Schritt kann entfallen (da trivial).*

*In einer echten Klausur sollten Sie von den erreichbaren Punkten mindestens die Hälfte erzielen, in diesem Beispiel also 17 Punkte.*