Beispiel für einen DAP2 Übungstest

Der Test besteht aus **vier** Aufgaben mit insgesamt **28** Punkten. Zum Bestehen des Tests sind daher **14 Punkte** notwendig.

Als Hilfsmittel ist **ein doppelseitig handgeschriebener Zettel** zulässig. Weitere Hilfsmittel dürfen nicht verwendet werden.

Schreiben Sie unbedingt **leserlich**, da unleserliche Antworten nicht gewertet werden können. Mit dem Beginn des Ausfüllens dieses Tests gilt die Prüfungsfähigkeit als bestätigt. Täuschungsversuche führen zu einer Bewertung mit 0 Punkten.

Aufgabe 1 (4+4 Punkte): (O–Notation und Rekursionsgleichung)

1. Kreuzen Sie jede Beziehung an, die gültig ist. Es gibt pro vollständig richtig ausgefüllter Zeile genau einen Punkt.

f	g	$f = O(g)$	$f = \Theta(g)$	$f = \Omega(g)$
$17n^2 + 31$	$3n^3$			
n	$\sqrt{n} + 8$			
$\frac{n^2-81}{n-9}$	$14n - 7$			
$5\sqrt{n} + \frac{1}{3}$	$\sqrt{n} + 3n \log n$			

2. Gegeben ist die folgende Rekursionsgleichung:

$$T(n) = \begin{cases} 2 \cdot T(n/2) + n^3 & \text{wenn } n > 1 \\ 1 & \text{sonst} \end{cases}$$

Bestimmen Sie eine asymptotisch gute obere Schranke für $T(n)$ und zeigen Sie ihre Gültigkeit mittels vollständige Induktion.

Aufgabe 2 (6 Punkte): (Schleifeninvariante)

Betrachten Sie den folgenden Algorithmus:

Verdreifache(**int** n):
$m \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
  $m \leftarrow m + 3$
return m

1. Stellen Sie eine Schleifeninvariante für m auf, die *vor* jeder Iteration gelten soll.
2. Beweisen Sie die Richtigkeit ihrer Schleifeninvariante durch vollständige Induktion.

Aufgabe 3 (6 Punkte): (Teile und Herrsche)

Es seien $a, n \in \mathbb{N}$ natürliche Zahlen. Gesucht ist ein **Teile&Herrsche**-Algorithmus für die schnelle Berechnung des Wertes a^n. Der Algorithmus sollte eine Worst-Case Laufzeit von $O(\log n)$ haben.

1. Entwerfen Sie einen solchen Teile-und-Herrsche Algorithmus und beschreiben Sie ihn mit eigenen Worten.
2. Geben Sie eine Implementierung Ihres Algorithmus in Pseudocode an.
3. Beweisen Sie die Korrektheit Ihres Algorithmus.

Aufgabe 4 (8 Punkte): (Dynamische Programmierung)

Das *Maxsummenproblem* sei folgendermaßen definiert. Für eine Folge

$$a_1, \ldots, a_n \in \mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$$

ganzer Zahlen, die in einem Array A gegeben sind, soll die Zielfunktion

$$f(i, j) = a_i + \cdots + a_j$$

über alle $1 \leq i \leq j \leq n$ maximiert werden, d.h. gesucht ist $\max\limits_{1 \leq i \leq j \leq n} f(i, j)$.

Mit $S(k)$ bezeichnen wir das Maximum über alle Summen $f(i, j)$, die mit dem Element a_k als letzten Summanden enden, für die also $1 \leq i \leq j = k$ gilt. Weiterhin bezeichne $M(k)$ das Maximum über alle Summen $f(i, j)$ im Intervall $[1, k]$, für die also $1 \leq i \leq j \leq k$ gilt.

1. Geben Sie eine rekursive Formulierung zur Berechnung der $S(k)$ und der $M(k)$ an.
2. Geben Sie einen Algorithmus in Pseudocode als dynamisches Programm an, welcher die rekursive Formulierung aus Aufgabenteil 1 umsetzt und das Maximum $\max\limits_{1 \leq i \leq j \leq n} f(i, j)$ ausgibt. Das maximierende Intervall $[i, j]$ muss dabei *nicht* berechnet werden, sondern nur das Maximum selbst.
3. Beweisen Sie, dass Ihr Algorithmus eine optimale Lösung berechnet.
4. Geben Sie eine möglichst kleine obere Schranke für die Worst-Case Laufzeit Ihres Algorithmus in O-Notation an und *begründen* Sie diese.