Algorithmen & Datenstrukturen
Midterm Test 1

Martin Avanzini <martin.avanzini@uibk.ac.at>
Thomas Bauereiß <thomas.bauereiss@uibk.ac.at>
Herbert Jordan <herbert@dps.uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>

3. Mai

Name	
Matrikelnummer	

Aufgabe	**mögliche Punkte**	**erreichte Punkte**
1.1	3	
1.2	6	
1.3	6	
2.1	4	
2.2	11	
3.1	3	
3.2	5	
3.3	5	
3.4	2	
Gesamt	**45**	

Aufgabe 1) Analyse von Algorithmen (15 Punkte)

1. Geben Sie die formalen Definitionen von $\mathcal{O}(g(n))$ und $\Theta(g(n))$ an.
2. Zeigen Sie anhand der Definition von $\mathcal{O}(n^3)$, dass $3n^2 + 12 = \mathcal{O}(n^3)$.
3. Finden Sie mit Hilfe des Master Theorems eine geschlossene Form $T(n) = \Theta(\dots)$ für die Rekursionsgleichung:

$$T(n) = 3 \cdot T\left(\frac{n}{3}\right) + n^3$$

Aufgabe 1) Lösung

1.

$$\mathcal{O}(g(n)) = \{f(n) \mid \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c\, g(n)\}$$
$$\Theta(g(n)) = \{f(n) \mid \exists n_0, c_1, c_2 > 0 : \forall n \geq n_0 : c_1\, g(n) \leq f(n) \leq c_2\, g(n)\}$$

2. Gemäß der Definition von $\mathcal{O}(n^3)$ müssen Konstanten c und n_0 gefunden werden, so dass gilt:

$$\begin{aligned} & 3\,n^2 + 12 \leq c\,n^3 \\ \Leftrightarrow & c\,n^3 - 3\,n^2 \geq 12 \\ (\text{sei } c = 3) \Leftrightarrow & 3\,n^3 - 3\,n^2 \geq 12 \\ \Leftrightarrow & 3\,n^2(n-1) \geq 12 \\ \Leftrightarrow & n^2(n-1) \geq 4 \end{aligned}$$

   Mit $n_0 = 2$ ist dies für alle $n \geq n_0$ erfüllt.

3. - $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ mit $a = 3$, $b = 3$, $f(n) = n^3$.
   - Es liegt möglicherweise Fall 3 des Master Theorems vor, da $f(n) = n^3 = \Omega(n^{\log_3(3)+\varepsilon}) = \Omega(n^{1+\varepsilon})$ mit z.B. $\varepsilon = 0.1$.
   - Zu prüfen ist die Nebenbedingung $\exists c < 1 : \exists n_0 : \forall n \geq n_0 : 3 \cdot f(\frac{n}{3}) \leq c \cdot f(n)$:

$$\begin{aligned} & 3 \cdot \left(\frac{n}{3}\right)^3 \leq c \cdot n^3 \\ \Leftrightarrow & 3 \cdot \left(\frac{n^3}{27}\right) \leq c \cdot n^3 \\ \Leftrightarrow & \frac{1}{9} \cdot n^3 \leq c \cdot n^3 \\ (\text{für } n \neq 0) \Leftrightarrow & c \geq \frac{1}{9} \end{aligned}$$

     Die Nebenbedingung ist also z.B. für $c = \frac{1}{9}$ und $n_0 = 1$ erfüllt.
   - Die Lösung ist demzufolge $T(n) = \Theta(f(n)) = \Theta(n^3)$.

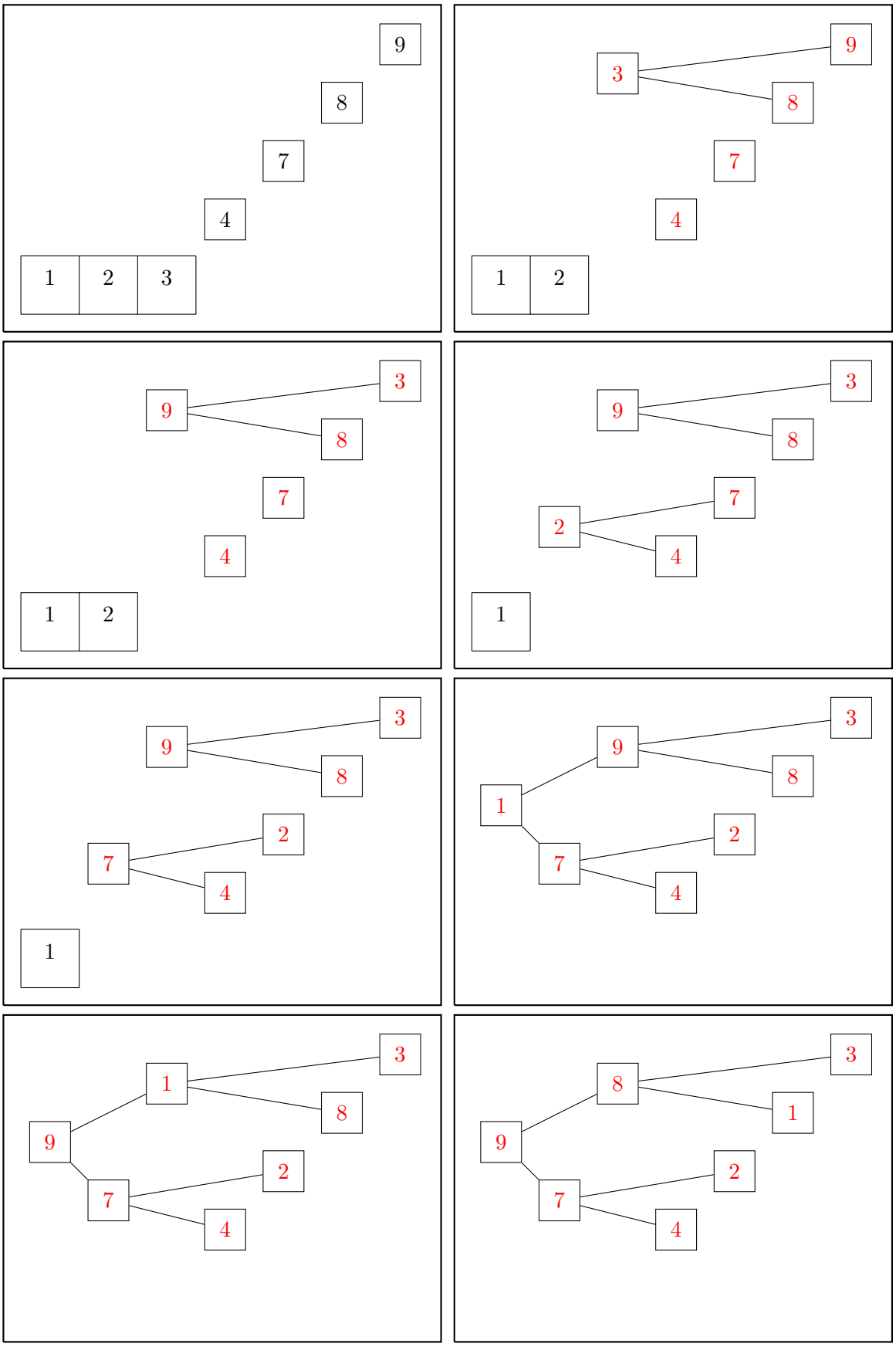
Aufgabe 2) Heap-Sort (15 Punkte)

1. Entscheiden Sie die Gültigkeit nachfolgender Aussagen:

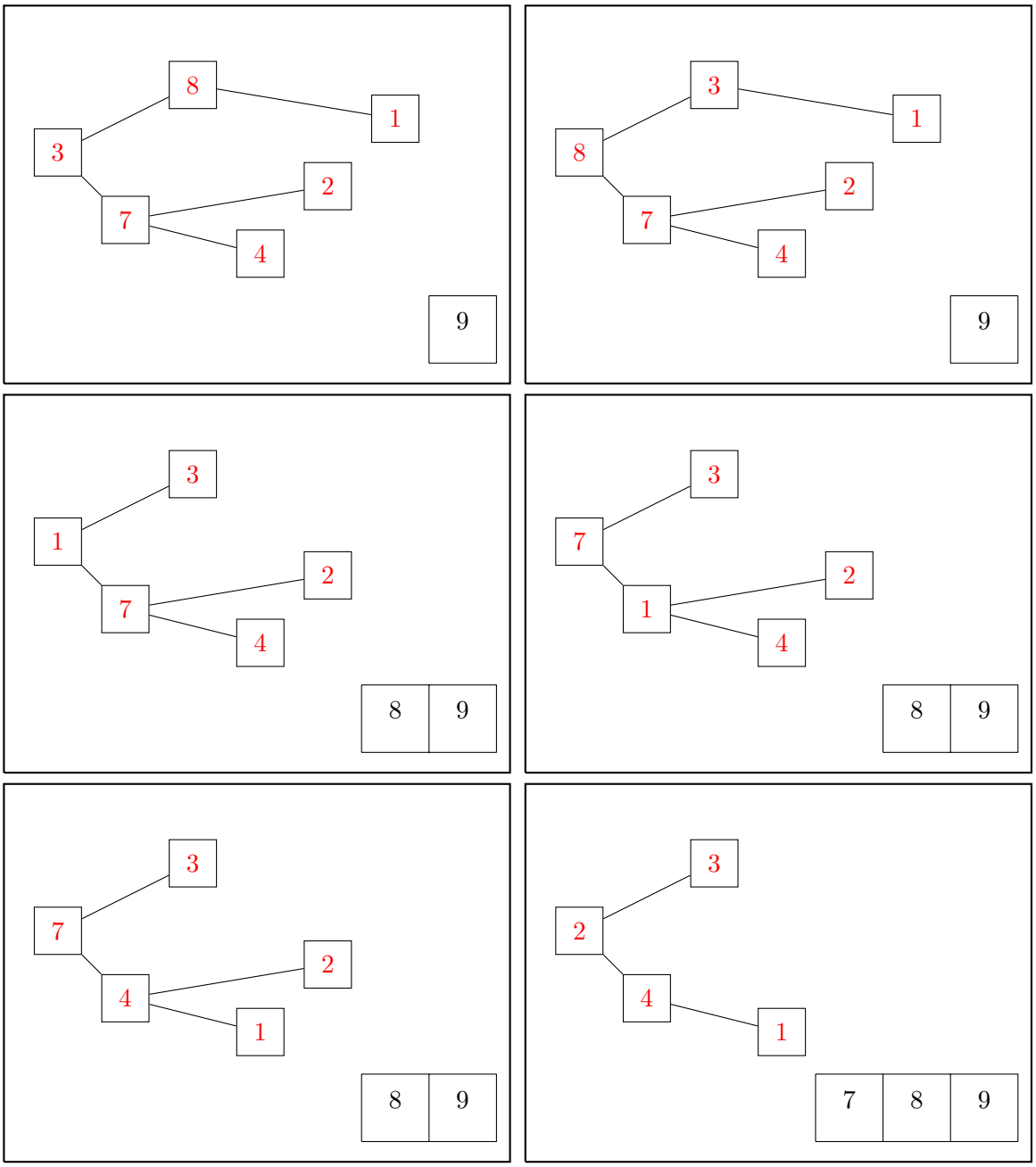
Aussage	wahr	falsch
Heap-Sort ist stabil.	☐	☒
Heap-Sort läuft insitu.	☒	☐
Heap-Sort operiert sequentiell.	☐	☒
Heap-Sort operiert in Zeit $\Theta(n^2)$ auf entarteten Eingaben worst-case.	☐	☒
Ein Array $a[0] \dots a[n-1]$ ist ein Heap genau dann wenn gilt: $\forall 0 \leqslant i < n : a[i] \geqslant a[2i+1] \wedge a[2i+2] \geqslant a[i]$ (falls $2i+1$ und $2i+2$ größer als n wird nichts verlangt).	☐	☒
Die Höhe eines Heaps mit n Elementen ist begrenzt durch $\lceil \log n \rceil$.	☒	☐

2. In dieser Aufgabe sollen Sie das Array $\{1, 2, 3, 4, 7, 8, 9\}$ mittels Heap-Sort sortiert werden. Vervollständigen Sie dazu die auf der nächsten Seite dargestellte Abbildung. Jedes Bild entspricht genau einer swap-Operation. Die Sortierung muss nicht vollständig durchgeführt werden, sind drei Elemente sortiert kann abgebrochen werden (siehe Abbildung).

(a) Herstellen des Heaps



(b) Sortierung



Aufgabe 3) Suchbäume (15 Punkte) Gegeben: ein binärer Suchbaum bestehend aus Knoten der Form:

Knoten eines binären Suchbaumes

```
class Node {
    int value;
    Node left;
    Node right;
}
```

Sollten keine linken/rechten Teilbäume vorhanden sein, sinde die jeweiligen Felder mit `null` belegt. Implementieren Sie folgende Operationen in Java-ähnlicher Syntax:

1. Finden des kleinsten Elements in einem Baum mit der Wurzel `root`. Dabei darf angenommen werden, dass `root != null`.

```
int smallest(Node root) {
    // TODO: implement
}
```

2. Zählen der Elemente in einem Baum mit der Wurzel `root`

```
int countElements(Node root) {
    // TODO: implement
}
```

3. Ausgabe der Element in preorder beginnend mit der Wurzel `root`

```
void printInPreorder(Node root) {
    // TODO: implement
}
```

4. Welche best-case / worst-case Laufzeitkomplexität haben Ihre Lösungen der Teilaufgaben 1 und 2 in einem Baum mit n Knoten? Begründen Sie ihre Antwort!

Aufgabe 3) Lösung

1. Um das kleinste zu finden muss man einfach immer nach links gehen:

```
int smallest(Node root) {
    if (root.left == null) {
        return root.value;
    }
    return smallest(root.left);
}
```

2. Um die Anzahl der Element zu ermitteln addiert man die Elemente im linken und rechten Teilbaum plus 1. Ein leerer Baum hat keine Elemente.

```
int countElements(Node root) {
    if (root == null) return 0;
    return 1 + countElements(root.left) + countElements(root.right);
}
```

3. Für die Preorder Ausgabe kann man rekursive durch den Baum wandern und dabei immer den aktuellen Knoten vor dem linken und rechten Teilbaum ausgeben.

```
void printInPreorder(Node root) {
    if (root == null) return;
    System.out.println(root.value);
    printInPreorder(root.left);
    printInPreorder(root.right);
}
```

4. (a) Suchen des kleinsten Elements — best: $\mathcal{O}(1)$ (entarteter Baum nach rechts), worst: $\mathcal{O}(n)$ (entarteter Baum nach links)

   (b) Zählen der Elemente: best = worst: $\mathcal{O}(n)$ da alle Knoten immer exakt 1x besucht werden müssen.