Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Friedrich Gretz, Benjamin Kaminski, Thomas Ströder

Präsenzübung Datenstrukturen und Algorithmen SS 2015

Vorname: _____

Nachname: _____

Matrikelnummer: _____

Studiengang (bitte **genau** einen markieren):

- ○ Informatik Bachelor
- ○ Informatik Lehramt (Bachelor)
- ○ Sonstiges: _____
- ○ Mathematik Bachelor
- ○ CES Bachelor

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	**18**	
Aufgabe 2	**18**	
Aufgabe 3	**18**	
Aufgabe 4	**18**	
Aufgabe 5	**18**	
Summe	**90**	

Allgemeine Hinweise:

- **Auf alle Blätter** (inklusive zusätzliche Blätter) müssen Sie **Ihren Vornamen, Ihren Nachnamen und Ihre Matrikelnummer** schreiben.
- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- Schreiben Sie mit **dokumentenechten** Stiften, nicht mit roten oder grünen Stiften und nicht mit Bleistiften.
- Bitte beantworten Sie die Aufgaben auf den Aufgabenblättern.
- Geben Sie für jede Aufgabe **maximal eine** Lösung an. Streichen Sie alles andere durch. Andernfalls werden alle Lösungen der Aufgabe mit **0 Punkten** bewertet.
- Werden **Täuschungsversuche** beobachtet, so wird die Übung mit **0 Punkten** bewertet.
- Geben Sie am Ende der Übung **alle Blätter zusammen mit den Aufgabenblättern ab**.

Aufgabe 1 ($\mathcal{O}$-Notation): (9 + 4 + 5 = 18 Punkte)

a) Ordnen Sie die folgenden Klassen von Funktionen in aufsteigender Größe an und setzen Sie dabei benachbarte Funktionenmengen mit "$\subset$" (echte Teilmenge) und "=" in Beziehung.

$$\mathcal{O}(n^2), \quad \mathcal{O}(3^n), \quad \mathcal{O}(n!), \quad \mathcal{O}(n + \log_2(n)), \quad \mathcal{O}\left(\sum_{i=0}^{\lceil\sqrt{n}\rceil} i\right),$$

$$\mathcal{O}\left(\frac{1}{n}\right), \quad \mathcal{O}(\log_2(n)), \quad \mathcal{O}(n^n), \quad \mathcal{O}\left(\sum_{i=0}^{n-1} i\right), \quad \mathcal{O}(n)$$

Schreiben Sie beispielsweise

$$\mathcal{O}(f(n)) \subset \mathcal{O}(g(n)) = \mathcal{O}(h(n))$$

wenn $\mathcal{O}(f(n))$ eine **echte Teilmenge** von $\mathcal{O}(g(n))$ ist und $\mathcal{O}(g(n))$ und $\mathcal{O}(h(n))$ **identisch** sind.

b) Beweisen oder widerlegen Sie folgende Aussage: Ist $f(n)$ eine Funktion mit $\mathbf{f(n) > 0}$ **für alle** $\mathbf{n} \in \mathbb{N}$, so gilt $\mathcal{O}(f(n)) = \mathcal{O}(f(n) + c)$ für alle $c \in \mathbb{R}^{>0}$.

c) Beweisen oder widerlegen Sie folgende Aussage: Ist $f(n)$ eine Funktion mit $\mathbf{f(n) \in \Omega(1)}$, so gilt $\mathcal{O}(f(n)) = \mathcal{O}(f(n) + c)$ für alle $c \in \mathbb{R}^{>0}$.

Aufgabe 2 (Sortieren): (3 + 5 + 7 + 3 = 18 Punkte)

a) Sortieren Sie das folgende Array mithilfe von Bubblesort. Geben Sie dazu das Array nach jeder Swap-Operation an. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

7	5	0	9	2

b) Sortieren Sie das folgende Array mithilfe von Quicksort. Geben Sie dazu das Array nach jeder Partition-Operation an und markieren Sie das jeweils verwendete Pivot-Element. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

5	9	6	4	1	8	7	2	3

c) Sortieren Sie das folgende Array mithilfe von Heapsort. Geben Sie dazu das Array nach jeder Swap-Operation an. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

6	7	3	1	8	4

d) Zeigen oder widerlegen Sie: Heapsort ist stabil.

Aufgabe 3 (Rekursionsgleichungen): (7 + 6 + 5 = 18 Punkte)

a) Geben Sie für das Programm

```
int berechne(int n){
    if(n <= 1)
        return 3;

    if(n = 2)
        return berechne(n/2);

    int value = log(n);

    value += 3 * berechne(n/2) + 4 * berechne(value) + 5
    return value;
}

int log(n){
    int res = 0;

    while(n > 1){
        n = n / 2;
        res = res + 1
    }

    return res;
}
```

eine Rekursionsgleichung für die asymptotische Laufzeit des Aufrufes `berechne(n)` an. Die elementaren, also die für die asymptotische Laufzeit relevanten, Operationen sind alle arithmetischen Operationen sowie Vergleiche. Sie brauchen die Basisfälle der Rekursionsgleichung *nicht* anzugeben.

b) Bestimmen Sie für die Rekursionsgleichung

$$T(n) = 125 \cdot T\left(\frac{n}{25}\right) + n^{\sqrt{2}} + \log_2(n) + 4 \cdot n + 3$$

die Komplexitätsklasse Θ mit Hilfe des Master-Theorems. Begründen Sie Ihre Antwort.

Hinweis: $\sqrt{2} \approx 1.414214$

c) Finden Sie mit Hilfe eines Rekursionsbaumes eine exakte Lösung der Rekursionsgleichung

$$T(n) = 3 \cdot T(n-1) + 1, \qquad T(1) = 1 \ .$$

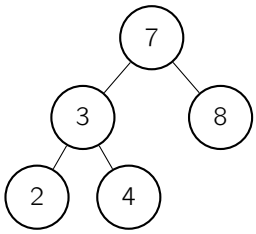
Dabei müssen Sie gegebenenfalls auftretende Summen *nicht* auflösen.

Aufgabe 4 (Bäume): (2 + 3 + 2 + 4 + 3 + 4 = 18 Punkte)

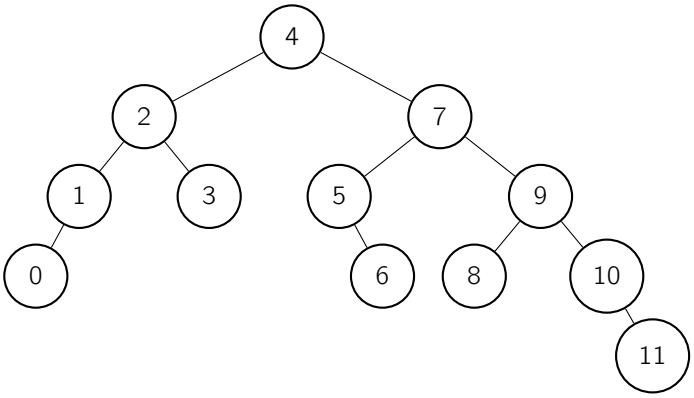
a) Geben Sie einen höhenbalancierten Binärbaum (also die Höhe der Teilbäume unterscheidet sich höchstens um 1 an jedem Knoten) mit den Schlüsselwerten von 0 bis 6 an, der bei *Inorder-Traversierung* die Schlüssel in folgender Reihenfolge ausgibt:

3, 6, 4, 0, 2, 1, 5

b) Fügen Sie den Wert 6 in den folgenden *AVL-Baum* ein und geben Sie die entstehenden Bäume nach jeder *Einfügeoperation* sowie jeder *Rotation* an. Markieren Sie außerdem zu jeder Rotation, welcher Knoten in welche Richtung rotiert wird:



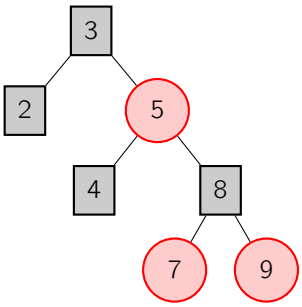
c) Löschen Sie den Wert 4 aus dem folgenden *AVL-Baum* und geben Sie die entstehenden Bäume nach jeder *Löschoperation* sowie jeder *Rotation* an. Markieren Sie außerdem zu jeder Rotation, welcher Knoten in welche Richtung rotiert wird:



d) Fügen Sie den Wert 6 in den folgenden *Rot-Schwarz-Baum* ein und geben Sie die entstehenden Bäume nach

- jeder *Einfügeoperation*,
- jeder *Rotation* sowie
- jeder *Umfärbung* an.

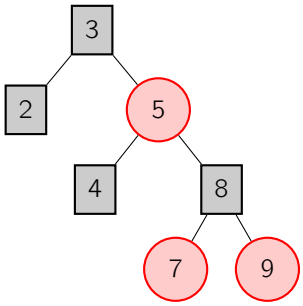
Markieren Sie außerdem zu jeder Rotation, welcher Knoten in welche Richtung rotiert wird. Mehrere Umfärbungen können Sie in einem Schritt zusammenfassen. Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.



e) Löschen Sie den Wert 4 aus dem folgenden *Rot-Schwarz-Baum* und geben Sie die entstehenden Bäume nach

- jeder *Löschoperation*,
- jeder *Rotation* sowie
- jeder *Umfärbung* an.

Markieren Sie außerdem zu jeder Rotation, welcher Knoten in welche Richtung rotiert wird. Mehrere Umfärbungen können Sie in einem Schritt zusammenfassen. Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.



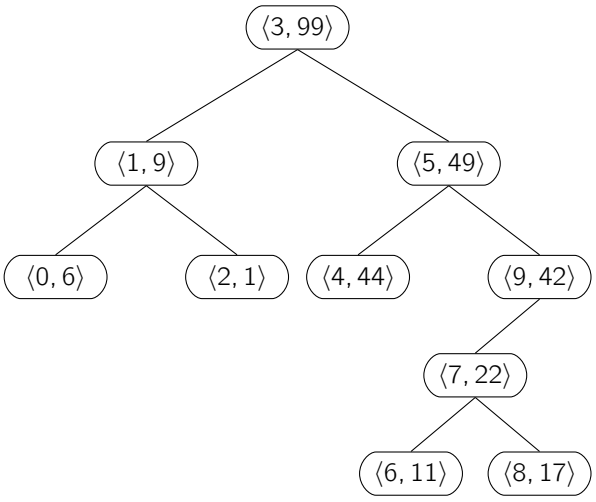
f) Beschreiben Sie stichpunktartig ein Verfahren, mit dem man aus einem aufsteigend sortierten Array einen AVL-Baum mit den gleichen Elementen in gleicher Anzahl konstruieren kann und dessen Worst-Case Laufzeit in $\mathcal{O}(n)$ ist, wobei n die Länge des Eingabearrays ist und die Anzahl an Vergleichen und Rechenoperationen sowie Lese- und Schreibzugriffen als elementare Operationen berücksichtigt werden sollen. Begründen Sie, warum Ihr Verfahren diese Laufzeitschranke einhält.

Aufgabe 5 (Datenstruktur): (9 + 9 = 18 Punkte)

Ein **Treap** (auch Balde oder Baufen) t ist ein binärer Baum. Ein Knoten besitzt

- einen Schlüssel **key**,
- eine Priorität **prio**,
- einen Zeiger **parent**, der auf den Elternknoten zeigt,
- einen Zeiger **left**, der auf den linken Teilbaum zeigt,
- sowie einen Zeiger **right**, der auf den rechten Teilbaum zeigt,

Bezüglich der Schlüssel erfüllt t die Suchbaum-Eigenschaft und bezüglich der Prioritäten erfüllt t einen Teil der Max-Heap-Eigenschaft: die Priorität jedes Knotens ist größer oder gleich der Prioritäten seiner Kinder. Ein Beispiel-Treap mit Knoten $\langle$**key**, **prio**$\rangle$ könnte also wie folgt aussehen:



Die Operation `insert(Treap t, int key, int prio)`, die ein neues Element mit dem Schlüssel `key` und der Priorität `prio` in den Treap `t` einfügt, kann wie folgt realisiert werden

```
insert(Treap t, int key, int prio) {
    Node n = new Node(key, prio);
    bstIns(t, n);
    bubble(n);
}
```

wobei

(i) `bstIns(Treap t, Node n)` ein neues Element `n` entsprechend der Suchbaum-Eigenschaft in den Treap `t` einfügt,

(ii) `bubble(Element n)` ein Element `n` solange durch Rotationen bewegt, bis die Max-Heap-Eigenschaft nicht mehr verletzt ist,

(iii) der Konstruktor `Node(int key, int prio)` einen Knoten kreiert, dessen Schlüssel und Priorität mit den gegebenen Werten initialisiert und dessen Elternknoten **parent** und Nachfolger **left**, **right** auf den Wert **null** setzt.

a) Implementieren Sie die Operation `bubble(Node n)`. Beachten Sie hierbei, dass Ihre Implementierung gewährleisten muss, dass `insert(Treap t, int key, int prio)` die Treap-Eigenschaften herstellt.

Hinweis: Sie dürfen die bekannten Operationen `leftRotate` und `rightRotate` aus der Vorlesung benutzen.

b) Gegeben seien n Elemente $\langle$**key**, **prio**$\rangle$ bestehend aus Schlüsseln und ihren Prioritäten, wobei jeder Schlüssel **maximal einmal** und jede Priorität **maximal einmal** vorkommt. Argumentieren Sie (informell), wieso der Treap t, der diese Elemente speichert, **eindeutig** ist. Das bedeutet, es gibt keinen anderen Treap t', der genau diese Elemente speichert.