Hochschule Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Prof. Dr. Peter Becker

Klausur — Sommersemester 2013
Datenstrukturen und Algorithmen
18. September 2013

Bevor Sie mit der Bearbeitung dieser Klausur beginnen, lesen Sie bitte folgende Hinweise. Diese Hinweise sind bei der Bearbeitung zu beachten.

1. Prüfen Sie die Vollständigkeit Ihres Exemplars. Es sollte

   - dieses Hinweisblatt und
   - acht Aufgaben auf acht Blättern

   umfassen.

2. Tragen Sie auf jedem Lösungsblatt oben an den vorgesehenen Stellen Ihren Namen und Ihre Matrikelnummer ein. Blätter ohne diese Angaben werden nicht bewertet.

   Hinter den Aufgaben ist jeweils hinreichend Platz für die Lösungen freigelassen. Reicht der Platz nicht aus, benutzen Sie die Rückseiten, wobei die Zuordnungen von Lösungen zu Aufgaben deutlich erkennbar sein müssen.

3. Geben Sie dieses Deckblatt zusammen mit den Aufgabenstellungen und den Lösungen sowie alles weitere beschriebene Papier ab.

4. Zugelassene Hilfsmittel: keine

5. Mit $\geq$ 50 Punkten haben Sie die Klausur bestanden.

6. Ergebnis (bitte nichts eintragen):

1 (12)	2 (12)	3 (10)	4 (20)	5 (15)	6 (10)	7 (15)	8 (10)	$\sum_{Punkte}$ (104)

Viel Erfolg!

Name: **Matrikel:**

Aufgabe 1 (12 Punkte)

Wir wollen in Java Poststücke mit den folgenden Eigenschaften modellieren:

- Jedes Poststück hat Empfänger, Absender, Gewicht, Porto und eine eindeutige Kennung zur Sendungsverfolgung. Diese eindeutige Kennung soll automatisch vergeben werden.
- Jedes Poststück ist entweder ein Päckchen oder ein Paket.
- Zur Berechnung des Porto soll eine Methode `berechnePorto()` zur Verfügung stehen.
- Bei einem Päckchen fallen pro kg Gewicht zwei Euro Porto an.
- Ein Paket kann zusätzlich gegen Verlust versischert werden. Das Porto eines Pakets beträgt 3 Euro plus 1 Euro pro kg an Gewicht. Wenn das Paket versichert ist, erhöht sich das Porto um weitere 2 Euro.
- Eine Methode `drucke()` soll die Eigenschaften eines Poststücks ausgeben.

Erstellen Sie in Java eine entsprechende Klassenhierarchie. Verwenden Sie dabei in angemessener Weise Vererbung und abstrakte Klassen.

Name: **Matrikel:**

```
public abstract class Poststueck {

    private static int nextId = 0;

    private int id;
    private String empfaenger;
    private String absender;
    private int gewicht;

    public Poststueck(String empfaenger, String absender, int gewicht) {
        this.id = nextId;
        nextId++;

        this.empfaenger = empfaenger;
        this.absender = absender;
        this.gewicht = gewicht;
    }

    public int gibGewicht() {
       return this.gewicht;
    }

    public abstract int berechnePorto();

    public void drucke() {
        System.out.println("Id: " + this.id);
        System.out.println("Empfaenger: " + this.empfaenger);
        System.out.println("Absender: " + this.absender);
        System.out.println("Gewicht: " + this.gewicht);
        System.out.println("Porto: " + this.berechnePorto());
    }
}

public class Paeckchen extends Poststueck {

    public Paeckchen(String empfaenger, String absender, int gewicht) {
        super(empfaenger, absender, gewicht);
    }

    public int berechnePorto() {
        return 2*this.gibGewicht();
    }
}

public class Paket extends Poststueck {

    private boolean versichert;
```

Name: **Matrikel:**

```
    public Paket(String empfaenger, String absender, int gewicht, boolean vers
        super(empfaenger, absender, gewicht);

        this.versichert = versichert;
    }

    public int berechnePorto() {
        return 3 + this.gibGewicht() + (this.versichert?2:0);
    }

    public void drucke() {
       super.drucke();
       System.out.println("versichert? " + this.versichert);
    }
}
```

Name: **Matrikel:**

Aufgabe 2 (4+4+4=12 Punkte)

(a) Die Schnittstelle `Foo` sei wie folgt definiert:

```
interface Foo {
    int bar(double x, String s);
    void baz() throws Exception;
}
```

Definieren Sie eine Klasse, die `Foo` implementiert.

(b) Auf Mengen sollen folgende Operationen möglich sein:

- Einfügen eines Elements
- Löschen eines Elements
- Test, ob sich ein Element in einer Menge befindet.

Definieren Sie eine generische Schnittstelle für Mengen. Berücksichtigen Sie auch notwendige Typeinschränkungen.

(c) Gegeben sei der folgende Quelltext:

```
interface F { }
class B { }
class A extends B implements F {
   public static void main(String[] args) {
      A a = new A();
      if (a instanceof A) {
         System.out.print("1");
      }
      if (a instanceof B) {
         System.out.print("2");
      }
      if (a instanceof F) {
         System.out.print("3");
      }
   }
}
```

Was wird ausgegeben und warum?

Name: **Matrikel:**

(a)
```
public class FooImpl implements Foo {
    public int bar(double x, String s) { return 42; }
    public void baz() { }
}
```

(b)
```
interface Menge<T> {
    void einfuegen(T t);
    void loeschen(T t);
    boolean istElementvon(T t);
}
```

bzw. mit Typeinschränkung

```
interface Menge<T extends Comparable <T>> {
    void einfuegen(T t);
    void loeschen(T t);
    boolean istElementvon(T t);
}
```

(c) Ausgegeben wird: `123`

- `a` wird als Instanz von `A` erzeugt.
- Weil die Klasse `A` Unterklasse von `B` ist, ist `a` auch eine Instanz von `B`.
- Weil die Klasse `A` die Schnittstelle `F` implementiert, ist `a` auch eine Instanz von `F`.

Name: **Matrikel:**

Aufgabe 3 (2+2+2+2+2=10 Punkte)

Es sei der folgende Quelltext gegeben:

```
import java.io.*;
public class Oppeln extends Schlesien {
  public static void main(String[] args) throws Exception {
    new Oppeln().visit();
  }
  void visit() throws Exception { }
}
class Schlesien {
  // insert code here
}
```

Für welche der folgenden Methodendefinitionen, jeweils eingefügt für die Zeile mit dem Kommentar "`insert code here`", lässt sich der Quelltext kompilieren?

(a) `void visit(int x) { }`

(b) `void visit() throws Exception { throw new IOException(); }`

(c) `void visit() throws IOException { }`

(d) `void visit() { throw new Exception(); }`

(e) `void visit() throws Exception { }`

Erläutern Sie jeweils in einem Satz, warum eine Kompilierung möglich bzw. nicht möglich ist.

(a) `void visit(int x) { }`

Kompiliert, weil unterschiedliche Signaturen in Ober- und Unterklasse vorliegen. Daher wird die Methode `visit()` hier nicht überschrieben.

(b) `void visit() throws Exception { throw new IOException(); }`

Kompoliert. Sinaturen sind identisch. In der Methode darf eine Unterklasse der Exception ausgelöst werden, die im Methodenkopf deklariert ist.

(c) `void visit() throws IOException { }`

Kompiliert nicht. Überschreibende Methoden dürfen keine allgemeineren als die in der Oberklasse definierten Exceptions auslösen!

(d) `void visit() { throw new Exception(); }`

Kompiliert nicht: Es liegen mehrere Fehler vor:

- In der Klasse `Schlesien` wird in `visit()` eine Exception ausgelöst, die im Kopf nicht deklariert ist.

- Die überschriebende Methode löst eine Exception aus, in der Oberklasse ist aber (im Methodenkopf) keine deklariert.

(e) `void visit() throws Exception { }`

Kompiliert, identische Signaturen in Ober- und Unterklasse.

Name: **Matrikel:**

Aufgabe 4 (20 Punkte)

Implementieren Sie mit Hilfe einfacher Verkettung eine generische Liste, die folgende Operationen unterstützt:

- Einfügen eines Elements am Ende der Liste
- Löschen des ersten Elements
- Alle Listenelemente ausgeben
- Die Reihenfolge der Elemente invertieren

  Beispiel: Aus der Liste $(3, 7, 2, 5)$ wird die Liste $(5, 2, 7, 3)$.

```
public class Liste<T> {

    private class Item {
        T    value;
        Item next;
    }

    private Item first;
    private Item last;

    public void insert(T t) {
        Item item = new Item();

        item.value = t;
        item.next = null;
        if (last==null) {
            first = last = item;
        }
        else {
            last.next = item;
            last = item;
        }
    }

    public void deleteFirst() {
        if (first!=null) {
            first = first.next;
        }
    }

    public void printAll() {
        Item item = first;
```

Name: **Matrikel:**

```
        while (item!=null) {
            System.out.print(item.value);
            item = item.next;
        }
        System.out.println();
    }

    public void invert() {
        Item item = first;
        Item firstInvert = null;
        Item lastInvert = null;
        Item itemInvert;

        while (item!=null) {
            itemInvert = new Item();
            itemInvert.value = item.value;
            itemInvert.next = firstInvert;
            if (lastInvert==null) {
               lastInvert = item;
            }
            firstInvert = itemInvert;
            item = item.next;
        }
        first = firstInvert;
        last = lastInvert;
    }
}
```

Name: **Matrikel:**

Aufgabe 5 (5+5+5=15 Punkte)

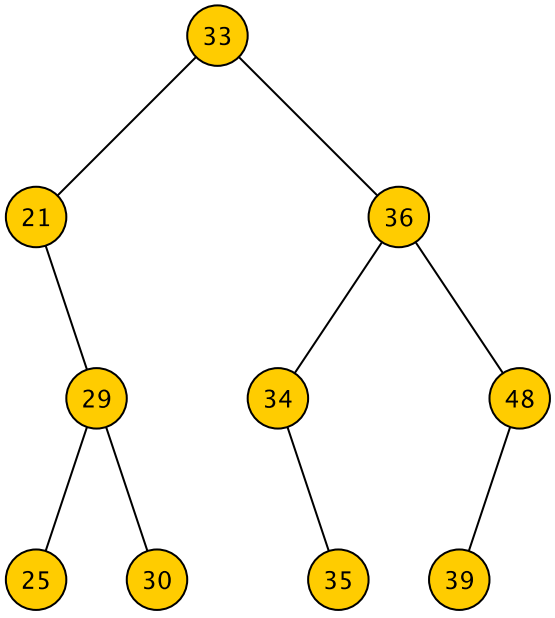
Gegeben sei der folgende Klassenrahmen zur Implementierung generischer Suchbäume:

```
class SearchTree<T extends Comparable<T>> {

    class Node {
        T    value;
        Node left;
        Node right;
    }

    private Node root;
    ...
}
```

(a) Implementieren Sie eine Methode zur Ausgabe aller Elemente des Suchbaums in aufsteigend sortierter Reihenfolge.

(b) Geben Sie für die Schlüssel $17, 42, 69$ alle möglichen Suchbäume an.

(c) Gegeben sei der folgende Suchbaum:



Wir löschen die 21 aus dem Baum von (c). Wie sieht der Baum jetzt aus? Was passiert, wenn wir als nächstes die 33 löschen?

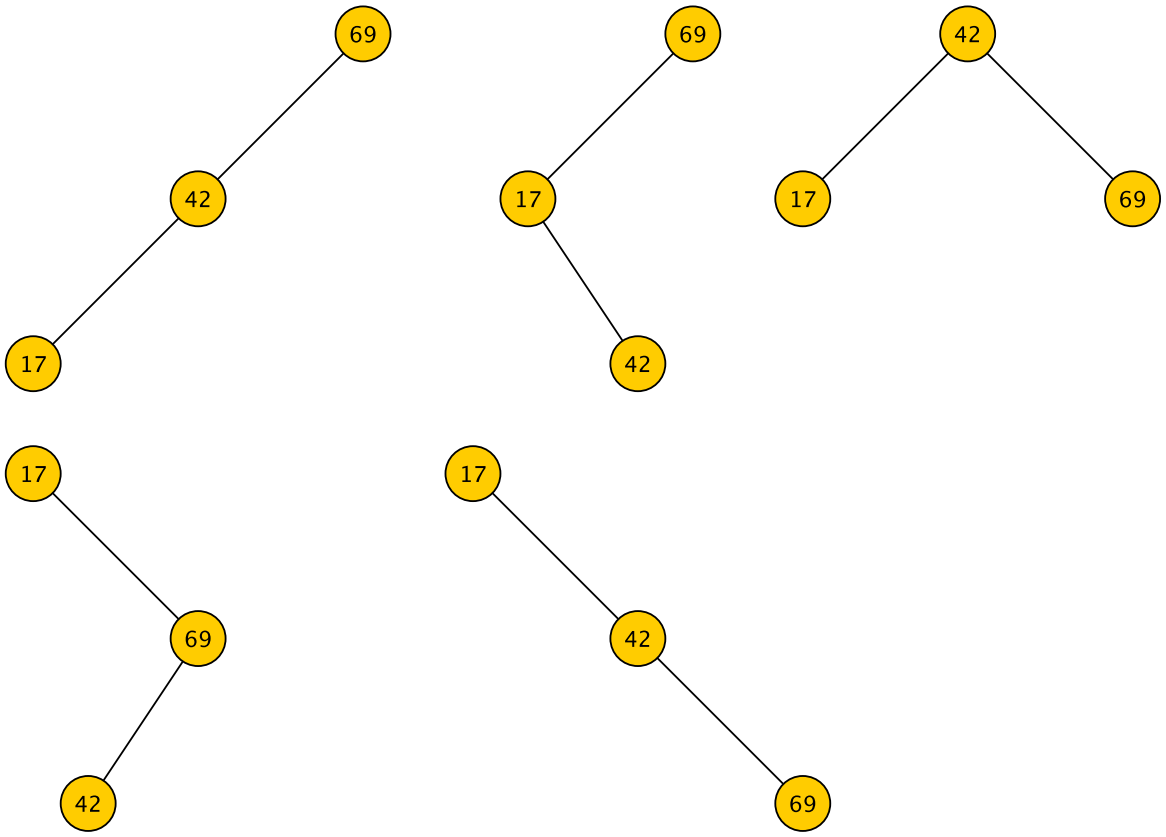
(a)

```
public void printSorted() {
    printSorted(root);
}
```

Name: **Matrikel:**

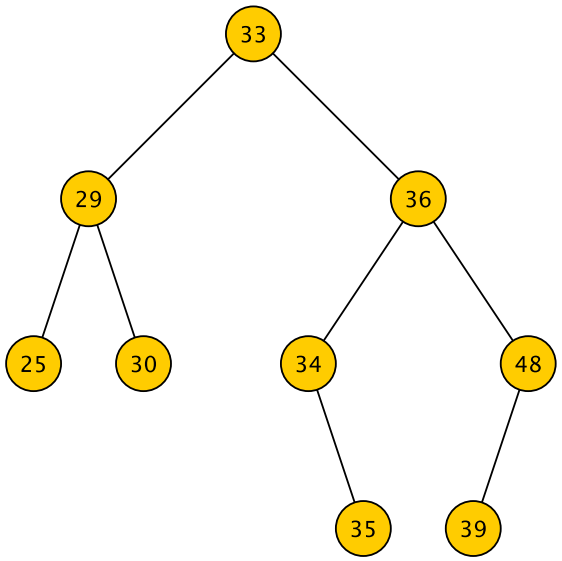
```
public void printSorted(Node node) {
    if (node==null) {
        return;
    }

    printSorted(node.left);
    System.out.println(node.value);
    printSorted(node.right);
}
```

(b) Alle Suchbäume für $17, 42, 69$:

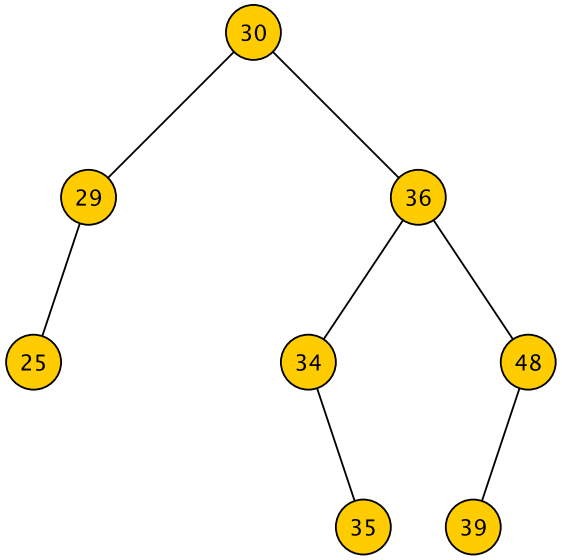


(c) Nach dem Löschen der 21:

Name: **Matrikel:**

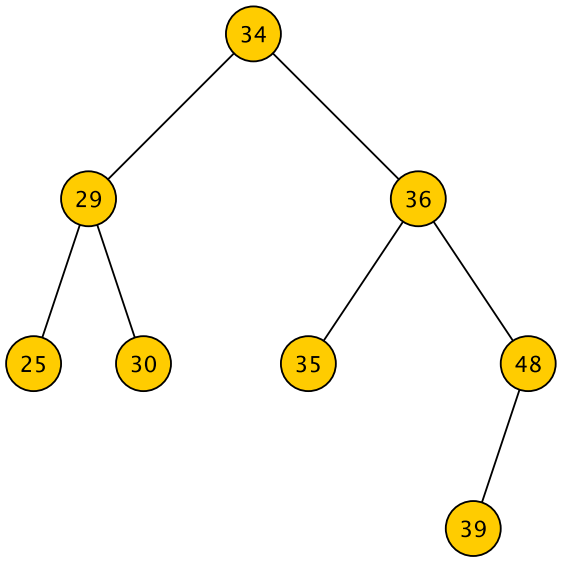


Nach dem Löschen der 33:



oder

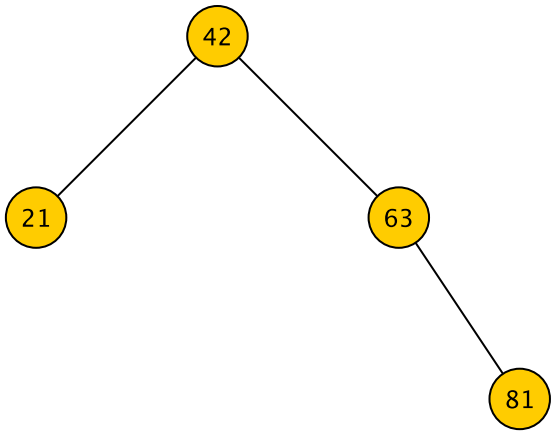
Name: **Matrikel:**



Name: **Matrikel:**

Aufgabe 6 (7+3=10 Punkte)

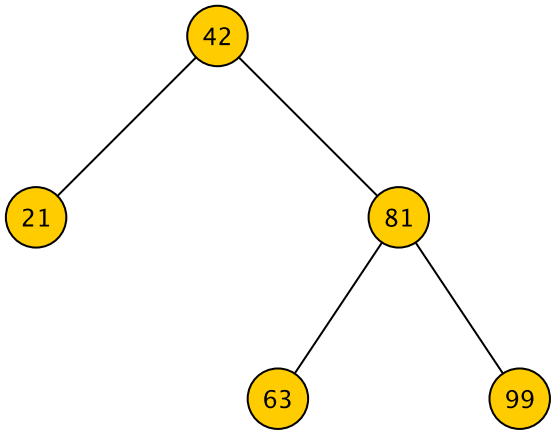
(a) Gegeben sei der folgende AVL-Baum:



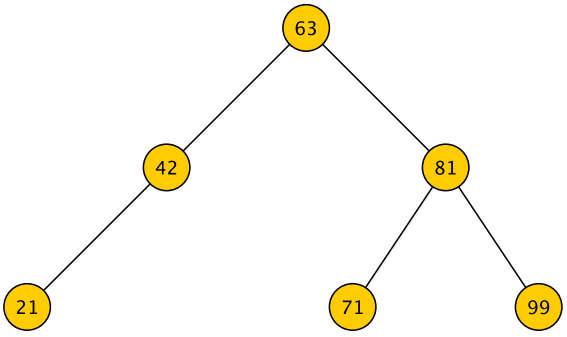
Nun werden nacheinander die folgenden Schlüssel eingefügt:
99, 71, 15, 69, 70

Geben Sie für jeden Schlüssel an, wie der AVL-Baum nach dem Einfügen des Schlüssels (inklusive eventuell notwendiger Ausgleichsoperationen) aussieht.

99 eingefügt:

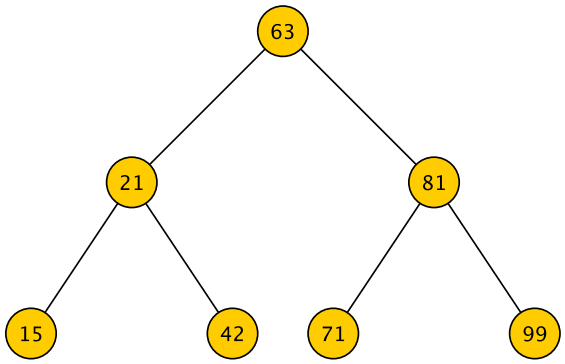


71 eingefügt:

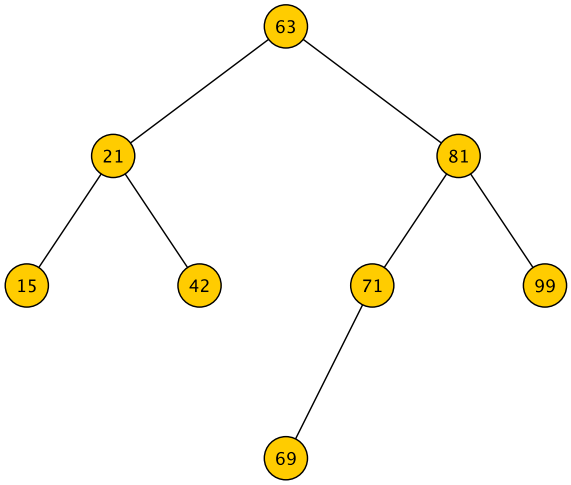


15 eingefügt:

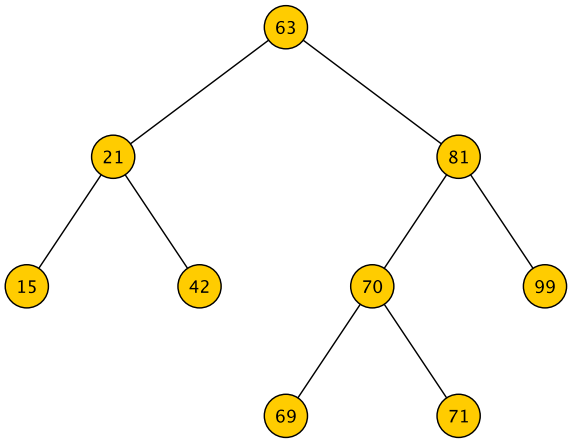
Name: **Matrikel:**



69 eingefügt:

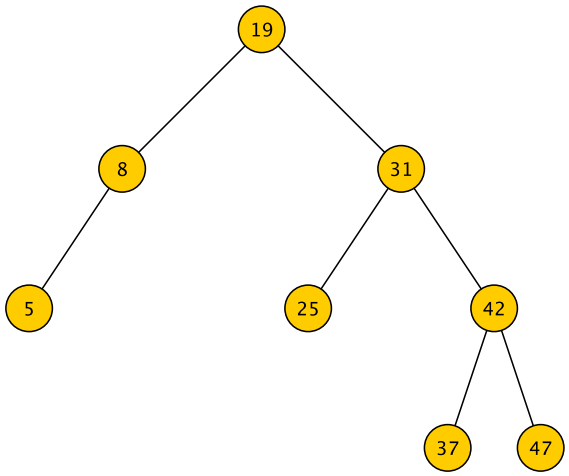


70 eingefügt:



(b) Gegeben sei der folgende AVL-Baum:

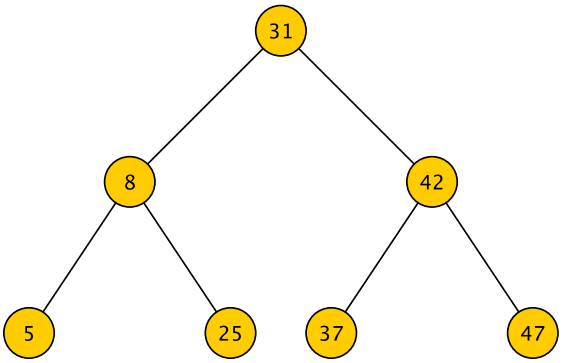
Name: **Matrikel:**



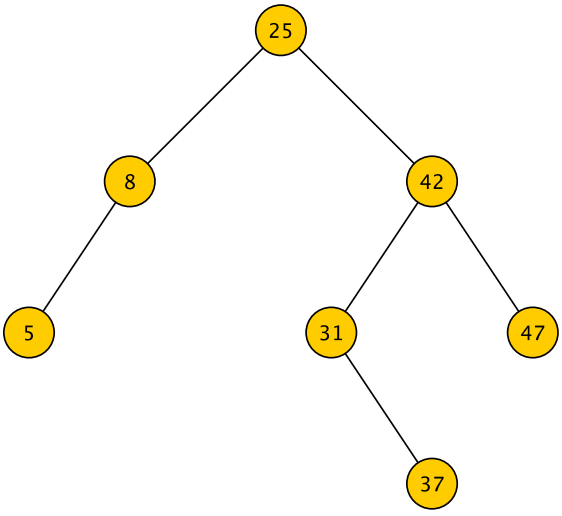
Wie sieht der Baum nach dem Löschen der 19 aus?

Es gibt zwei verschiedene Möglichkeiten.

Nach dem Löschen der 19 wird die 8 hochgezogen. Dann findet an der Wurzel eine RR-Rotation statt.



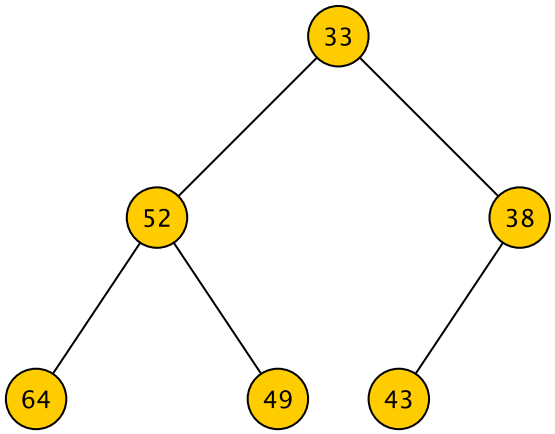
Oder nach dem Löschen der 19 wird die 25 hochgezogen. Dann findet bei der 31 eine RR-Rotation statt.



Name: **Matrikel:**

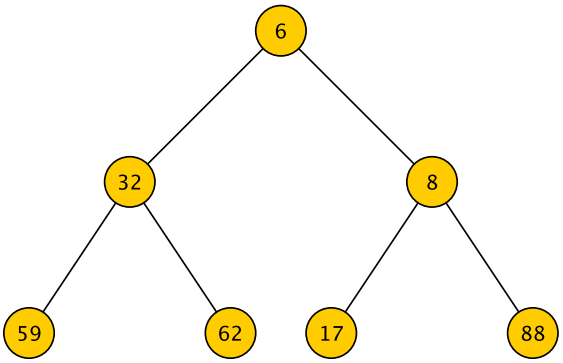
Aufgabe 7 (3+3+6+2=15 Punkte)

(a) Ist der folgende Baum ein Heap? Begründen Sie Ihre Antwort (ein Satz).

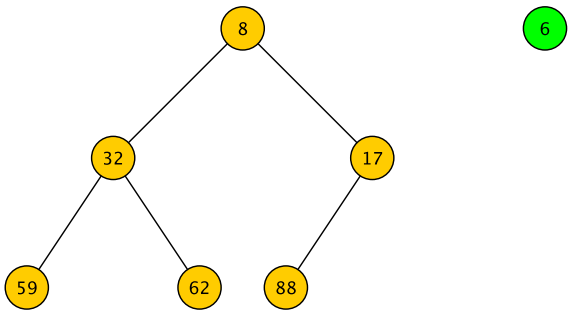


Der Baum ist kein (Min-)Heap, weil an den Knoten 52 und 49 die Heap-Bedingung verletzt ist.

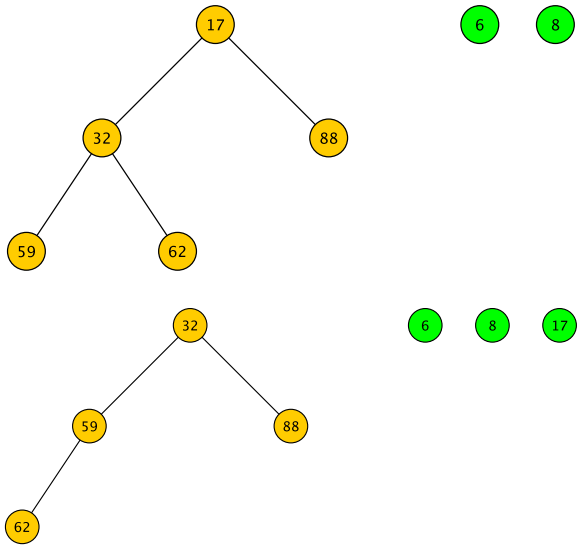
(b) Geben Sie für die Zahlenfolge $62, 59, 8, 17, 32, 6, 88$ einen Heap an.



(c) Führen Sie ausgehend von dem Heap aus (b) drei Iterationen der Sortierphase von Heapsort durch. Geben Sie dabei jeweils an, welche Zahl aus dem Heap entfernt wird und wie der Heap für die nächste Iteration aussieht.



Name: **Matrikel:**



(d) Nennen Sie ein stabiles Sortierverfahren. Welchen Zeitaufwand hat dieses Verfahren?
Mergesort, hat Aufwand $O(n \log n)$.

Name: **Matrikel:**

Aufgabe 8 (6+4=10 Punkte)

(a) Gegeben Sei eine (zunächst leere) Hash-Tabelle der Größe 7, sowie die Hashfunktion

$$h(x) \quad = \quad x \bmod 7$$

Fügen Sie die folgenden Zahlen in die Hash-Tabelle ein, wobei Kollisionen durch Verkettung behandelt werden:

$$24, 47, 21, 40, 8, 19$$

	24	47	21	40	8	19
0			21	21	21	21
1					8	8
2						
3	24	24	24	24	24	24
4						
5		47	47	$40 \rightarrow 47$	$40 \rightarrow 47$	$19 \rightarrow 40 \rightarrow 47$
6						

(b) Wir wollen ganzzahlige Schlüssel x in einer Hashtabelle der Größe N verwalten. Erläutern Sie jeweils mit einem Satz, warum die folgenden Hashfunktionen $h(x)$ nicht sinnvoll sind.

- $h(x) = 4711 \bmod N$
  Keine Streuung, jeder Schlüssel wird auf die gleiche Position der Hashtabelle abgebildet.
- $h(x) = (x/1024) \bmod N$
  Die letzten 10 Bits des Schlüssels bleiben unberücksichtigt, so dass bspw. alle Schlüssel zwischen 0 und 1023 auf die gleiche Position der Hashtabelle abgebildet werden.
- $h(x) = (\texttt{random()} \cdot x) \bmod N$
  wobei die Funktion `random()` eine ganzzahlige Zufallszahl liefert.

  Mit einer Zufallszahl ist ein Wiederfinden des Schlüssels nicht möglich.