Aufgabe 1 (12 Punkte)

Diese Aufgabe umfasst 3 Multiple-Choice-Unteraufgaben. Innerhalb jeder Unteraufgabe gilt: wenn alle 4 Kreuze an der richtigen Stelle stehen, gibt es 4 Punkte für die Unteraufgabe. Ein falsches Kreuz gibt einen Punkt Abzug. Wer 2 richtige und 2 falsche Kreuzchen in einer Unteraufgabe macht, erhält 1+1-1-1=0 Punkte. Es gibt keine negativen Gesamtpunktzahlen pro Unteraufgabe, jede Unteraufgabe bringt 0 bis 4 Punkte.

Aufgabe 1a (4 Punkte):

		Ja	Nein	Frage
a)				**Interface**
	1.	☒	☐	`class A extends B implements C,D {…}` ist korrekter Java-Code
	2.	☐	☒	Eine Klasse, die keine Methoden besitzt, ist ein Interface.
	3.	☒	☐	Die Klasse String besitzt eine Methode compareTo.
	4.	☐	☒	Eine Klasse darf keine 2 Methoden mit gleicher Signatur enthalten.

Aufgabe 1b (4 Punkte):

				Sortieren
	1.	☐	☒	Quicksort ist ein greedy-Algorithmus.
	2.	☒	☐	Selection Sort hat – bei gleicher Anzahl Elemente- immer den gleichen Aufwand, unabhängig von der konkreten Zahlenfolge.
	3.	☒	☐	Mit n Vergleichen kann man prüfen, ob n+1 Zahlen geordnet sind.
	4.	☒	☐	Mergesort funktioniert nach den Prinzip „easy split – hard join“,

Aufgabe 1c (4 Punkte):

c)				**Suchen**
	1.	☒	☐	Greedy-Verfahren sind bei kleinen Suchmengen empfehlenswert.
	2.	☒	☐	Die Anzahl der Vergleichsoperationen beim Hashing ist unabhängig von der Anzahl der gespeicherten Elemente, wenn es keine Kollisionen gibt.
	3.	☒	☐	Binäres Suchen ist mit Feldern effizienter zu implementieren als mit verzeigerten Listen.
	4.	☒	☐	Binärsuche ist ein Divide & Conquer-Verfahren.

Aufgabe 2 (12 Punkte)

Die folgende Klasse implementiert generische Listen. Ergänzen Sie die Klasse um eine Methode `public ObjectNode getLast()`, die die Knoteninformation des letzten Listenelementes der Liste liefert. Ist die Liste leer, wird null zurückgegeben..

```
public class List {
      private ObjectNode head;

      public List() {
            head = null;
      }

      public List(ObjectNode head) {
            this.head = head;
      }

      public void insertFirst(Object data) {
            head = new ObjectNode(data, head);
      }

      public Object getHead() {
            return head.data;
      }

      public boolean empty() {
            return head == null;
      }

      public List getTail() {
            if (head == null)
                  return new List(null);
            else if (head.getNext() == null)
                  return new List(null);
            else
                  return new List(head.getNext());
      }
```

```
public class ObjectNode {
  Object data;        Ob-
jectNode next;

  Object getData() {
    return data;
  }

  void setData(Object data)
{
    this.data = data;
  }

  ObjectNode getNext() {
    return next;
  }

  void setNext(ObjectNode
next) {
    this.next = next;
  }

  public ObjectNode(Object
a) {
    this(a, null);
  }

  public ObjectNode(Object
a, ObjectNode n) {
    data = a;
    next = n;
```

```
/**
*
* Bitte fügen Sie den Code in den Methodenrumpf ein:
**/

      public ObjectNode getLast() {

            ObjectNode l,n;
            if (head==null) return null; else
            { l= head; n = l.getNext();
              while (n!=null) {
                 l=n; n = n.getNext();
              }
              return l;
            }









      }
}
```

Aufgabe 3 (12 Punkte)

Gegeben sei folgendes Code-Fragment:

```
          int i=n,j; int[][] a;
          …
(1)  while (i>0) {
(2)       j = i;
(3)       while (j>0) {
(4)          a[i,j] = 0;
(5)          j = j-1;
          }
(6)       i = i-1;
     }
```

a) Berechnen Sie den exakten Aufwand in Abhängigkeit von n unter der Maßgabe, dass jede Zeile genau eine Aufwandseinheit erfordert.

Die innere while-Schleife hat i Durchläufe, macht den Aufwand 3i+1, jeweils i für die Zeile 4 und 5, i+1 für Zeile 3.
Die äußere Schleife wird n-mal druchlaufen, für i=n,..,1. Das heißt: Zeilen 2 und 6 werden n mal durchlaufen, Zeile 1 n+1 mal, macht zusammen $3n+1+\sum_{i=1}^{n}(3i+1)=3\frac{n(n+1)}{2}+4n+1$

b) Zeigen oder widerlegen Sie, dass der asymptotische Aufwand $O(n^2)$ ist **(Diese Teilaufgabe braucht für A&M nicht bearbeitet zu werden.**

Ab n=2 gilt $3\frac{n(n+1)}{2}+4n+1\le 3\frac{2n^2}{2}+4n+1\le 3n^2+4n+1\le 7n^2$

Aufgabe 4 (12 Punkte)

Sortieren Sie die folgenden Zahlen Array-basiert, indem Sie die nachstehenden Tabellen vervollständigen.

a) Sortieren durch Auswählen (selection sort)

8	**2**	**4**	**1**	**7**	**5**	**3**	**6**
1	2	4	8	7	5	3	6
1	2	4	8	7	5	3	6
1	2	3	8	7	5	4	6
1	2	3	4	7	5	8	6
1	2	3	4	5	7	8	6
1	2	3	4	5	6	8	7
1	2	3	4	5	6	7	8

b) Sortieren durch Quicksort (Algorithmus unten)

8	2	4	1	7	5	3	**6**
3	2	4	1	**5**	**6**	8	**7**
3	2	4	**1**	**5**		**7**	8
1	2	4	**3**				**8**
	2	**3**	**4**				
	2		**4**				

```
public void quickSort(int l, int r) {
   int i = l, j = r-1;
   if (l >= r) return;
   Comparable pivot = data[r];
      while( data[i].compareTo(pivot) < 0 ) i++;
      while( j>=l && data[j].compareTo(pivot) >= 0) j--;
      while( i < j ) {
        swap(i,j);
        while( data[i].compareTo(pivot) < 0 ) i++;
        while( data[j].compareTo(pivot) >= 0 ) j--;
      }
      swap(i,r);
      quickSort(l, i-1);
      quickSort(i+1, r);
    }
  }
```

Aufgabe 5 (12 Punkte)

Aus dem Paket Kapitel13 kennen Sie die Klasse Tree und das

```
public interface NodeActionInterface {
  /**
   * Methode, deren Implementierung auf einem Knoten arbeitet.
   * @param n Der zu bearbeitende Knoten
   */
  public void action(Node n);
}
```

Wir gehen im Folgenden von einem Baum aus, dessen Knoten Integer-Objekte beinhalten. Die Knoten sehen dann so aus:

```
Public class Node {
      Object data;
      Node left;
      Node right;
…
}
```

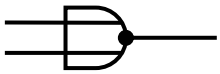
Ist n ein Objekt von Typ `Node`, so wird mit `n.data = new Integer(7)` der Wert 7 als Objekt gespeichert. Der int-Wert kann so ausgelesen und beispielsweise der Variablen i zugewiesen werden: `int i = ((Integer)n.data).intValue()`.

Leiten Sie aus `NodeActionInterface` eine Klasse `public class MaxNodeValue` ab, die das Maximum der Knoteninhalte aller Knoten ermittelt, auf die die Methode action angewandt wird. Wir geben den Rahmen vor, Sie brauchen nur die Methode action auszuprogrammieren!

```
   public class MaxNodeValue implements NodeActionInterface {

      private int maxvalue=Integer.MIN_VALUE;  // kleinster Wert überhaupt

      public void action(Node n) { // hier programmieren!

            if (((Integer)n.data).intValue()>maxvalue)
                  maxvalue = ((Integer)n.data).intValue();









      }

      public String toString() {
            return„“Maximaler Wert im Baum = “+maxvalue);

}
}
```

Aufgabe 6 (12 Punkte)

Geben Sie eine Gatter-Schaltung an, die die Verknüpfung $a \vee b \wedge c$ ausschließlich mit NOR-Gattern realisiert.



$$
\begin{aligned}
& a \vee b \wedge c \\
&= (a \vee b) \wedge (a \vee c) \\
&= \neg\neg((a \vee b) \wedge (a \vee c)) \\
&= \neg(\neg(a \vee b) \vee \neg(a \vee c))
\end{aligned}
$$