Name, Vorname, Matrikelnummer Studiengang

# Klausur
# Algorithmen und Datenstrukturen
# WS 2018/19 – 28.01.2019

## Hinweise:

- Die Bearbeitungszeit beträgt 120 Minuten.
- Zum Bestehen der Klausur sind 50 Punkte erforderlich.
- Schreiben Sie auf die ersten beiden Blätter Ihren Namen, Matrikelnummer und Studiengang.
- Erlaubte Hilfsmittel: keine
- Lösen Sie nicht die Klammerung der Klausur!
- Tragen Sie Ihre Lösungsvorschläge in die Klausurvorlage ein. Weiteres Papier können Sie bei der Klausuraufsicht anfordern.
- Alle vorgegebenen und zu erstellenden Programmtexte beziehen sich auf die Programmiersprache Java.
- Bitte schreiben Sie deutlich.

*Viel Erfolg !*

| **Aufgabe** | **Maximalpunkte** | **Erreichte Punkte** |
|---|---|---|
| 1 (Multiple Choice, Wissen) | 10 | |
| 2 (Graphen) | 18 | |
| 3 (Collection, Listen) | 14 | |
| 4 (Rekursion, Komplexität) | 18 | |
| 5 (Listen) | 16 | |
| 6 (Hashing) | 6 | |
| 7 (Sortieren) | 16 | |
| 8 (B-Bäume) | 12 | |
| **Summe** | 110 | |

Name, Vorname, Matrikelnummer Studiengang

## *Aufgabe 1 (Multiple Choice, Wissen)* *10 Punkte*

a) Kreuzen Sie in den folgenden Teilaufgaben an, welche Aussagen richtig und welche falsch sind. Für jede korrekt markierte Aussage erhalten Sie 0,5 Punkte, für jede falsch markierte werden 0,5 Punkte abgezogen. Pro Teilaufgabe erhalten Sie mindestens 0 Punkte.

| Bewerten Sie folgende Aussagen über Sortieralgorithmen | **Richtig** | **Falsch** |
|---|---|---|
| Insertion-Sort benötigt im günstigsten Fall maximal O(n) Vergleiche, um n Schlüssel zu sortieren | | |
| Quicksort benötigt im ungünstigsten Fall (worst case) O(n log n) Vergleiche, um n Schlüssel zu sortieren | | |
| Heap-Sort und Bottom-Up-Heap-Sort unterscheiden sich nur in der versickern-Methode | | |
| Merge-Sort benötigt bei der günstigsten Eingabe maximal O(n log(n)) Vergleiche zum Sortieren von n Schlüsseln | | |

| Welche der folgenden Aussagen sind korrekt? | **Richtig** | **Falsch** |
|---|---|---|
| $n\ (\log_3 n)^3 \in O(n^3 \log_2 n)$ | | |
| $n^2 \in O(2^n)$ | | |
| $n^2 + n^3 \in O(n^3)$ | | |
| $15\ (n + 2^n) \in O(2^n)$ | | |

| Welche Aussagen über Suchverfahren sind korrekt? | **Richtig** | **Falsch** |
|---|---|---|
| AVL-Bäume mit n Knoten haben stets eine Höhe von maximal log(n) | | |
| Durchläuft man einen AVL-Baum in Preorder-Reihenfolge, so erhält man die Knoten in sortierter Reihenfolge | | |
| Durchläuft man einen AVL-Baum in Inorder-Reihenfolge, so erhält man die Knoten in sortierter Reihenfolge | | |
| Das Suchen in einem AVL-Baum mit n Knoten benötigt im ungünstigsten Fall O(log(n)) Schritte | | |

| Welche der folgenden Aussagen sind korrekt? | **Richtig** | **Falsch** |
|---|---|---|
| P ist die Klasse der berechenbaren Probleme | | |
| Probleme in NP sind nicht berechenbar | | |
| Das Traveling Salesman Problem ist berechenbar | | |
| Das Traveling Salesman Problem ist in NP enthalten | | |

b) Wann nennen wir einen binären Suchbaum einen AVL-Baum? Wenn Sie den Begriff Balancefaktor benutzen, so erläutern Sie diesen Begriff ebenfalls.

Name, Vorname, Matrikelnummer Studiengang

## *Aufgabe 2 (Graphen)* *18 Punkte*

Folgender Graph ist durch seine Adjazenzmatrix gegeben:

| | | Nach | | | | |
|---|---|---|---|---|---|---|
| | | **A** | **B** | **C** | **D** | **E** |
| **Von** | **A** | | 3 | 1 | | |
| | **B** | | | 2 | | 6 |
| | **C** | 1 | 1 | | 3 | |
| | **D** | | 4 | | | 1 |
| | **E** | | | | 1 | |

a) Kann man für obigen Graphen eine topologische Sortierung angeben (Begründung)? Falls eine topologische Sortierung existiert, so geben Sie eine topologische Sortierung an.

b) Zeichnen Sie den gegebenen Graphen.

c) Führen Sie auf dem obigen Graphen den Algorithmus von Dijkstra zur Bestimmung minimaler Wege mit dem Startknoten A aus. Geben Sie nach jeder Runde die Menge der markierten Knoten, sowie die bis dahin errechneten Abstände und den jeweiligen Vorgänger in Tabellenform an.

d) Bestimmen Sie auf Basis der Tabelle aus Aufgabenteil c) die kürzesten Wege von Knoten A zu den Knoten D und E. Erläutern Sie kurz, wie Sie dabei vorgehen.

## *Aufgabe 3 (Collections, Listen)* *14 Punkte*

Betrachten Sie die folgende Implementierung einer Liste von Personen, wobei die notwendigen import-Anweisungen nicht mit angegeben sind.

```
public class Person
{
    public String name;
    public String vorname;
    public int alter;

    public Person(String name, String vorname, int alter)
    {
        this.name = name;
        this.vorname = vorname;
        this.alter = alter;
    }
}

public class PersonenListe
{
    private List<Person> liste = new LinkedList<Person>();

    public String toString()
    {
        // Aufgabenteil a)
    }
    public PersonenListe kehreUm()
    {
        // Aufgabenteil b)
    }
}
```

a) Ergänzen Sie die Klasse `PersonenListe` um die Methode `toString()`, die die in der Liste enthaltenen Personen als String zurückgibt. Der Name, Vorname und das Alter der Person soll jeweils durch ein Komma getrennt in einer eigenen Zeile angegeben werden.

```
public String toString()
{
```

```
    }
```

b) Ergänzen Sie nun die Klasse `PersonenListe` um die Methode `kehreUm()`, die eine Kopie der Liste zurückgibt, in der aber die Listenelemente in umgekehrter Reihenfolge enthalten sind.

```
    public PersonenListe kehreUm()
    {
```

```
}
```

## *Aufgabe 4 (Rekursion, Komplexität)* *18 Punkte*

Betrachten Sie die folgende (teilweise) Implementierung eines Suchbaumes

```
class Node
{
   public int data;
   public Node linkesKind, rechtesKind;
}

public class Suchbaum
{
   private Node wurzel;

   public int berechne1()
   {
      return berechne1(wurzel);
   }

   private int berechne1(Node node)
   {
      if (node == null) return 0;
      else return 1 + berechne1(node.linkesKind)
                    + berechne1(node.rechtesKind);
   }

   public boolean berechne2(int d)
   {
      return berechne2(wurzel, d);
   }

   private boolean berechne2(Node node, int d)
   {
      if (node == null) return false;
      else if (node.data == d) return true;
      else
         if (node.data>d) return berechne2(node.linkesKind, d);
         else return berechne2(node.rechtesKind, d);
   }
}
```

a) Was berechnen die Methoden `public int berechne1()` und `public boolean berechne2(int d)`?

`berechne1()`:

`berechne2(d)`:

b) Bestimmen Sie Rekursionsbasis und Rekursionsvorschrift der Methode `private int berechne1(Node node)`

Rekursionsbasis:

Rekursionsvorschrift:

c) Bestimmen Sie die Rekursionstiefe und die asymptotische Laufzeit der Methoden `public int berechne1()` und `public boolean berechne2(int d)` auf Suchbäumen der Höhe t mit n Knoten:

`berechne1()`:

`berechne2(d)`:

d) Implementieren Sie **eine** der Methoden `public int berechne1()` **oder** `public boolean berechne2(int d)` iterativ. Geben Sie auch die asymptotische Laufzeit Ihrer iterativen Implementierung an.

Name, Vorname, Matrikelnummer Studiengang

## *Aufgabe 5 (Listen)* *16 Punkte*

Es sei die folgende teilweise Implementierung einer Listen-Klasse gegeben.

```
public class Link
{
   private int wert;
   private Link naechster;

   public Link(int wert, Link naechster)
   {
      this.wert = wert;
      this.naechster = naechster;
   }
   public int getWert(){
      return wert;
   }
   public Link getNaechster(){
      return naechster;
   }
   public void setNaechster(Link naechster){
      this.naechster = naechster;
   }
}

public class Liste
{
   private Link start;

   public Liste(){
   }
   public Liste(Link start){
      this.start = start;
   }
   public Link getStart(){
      return start;
   }

   public boolean findeWert(int wert){
      // Aufgabenteil a)
   }
   public void fuegeWertEin(int wert){
      // Aufgabenteil b)
   }
   public void loescheWert(int wert){
      // Aufgabenteil c)
   }

}
```

a) Implementieren Sie in der Klasse `Liste` die Methode `findeWert(int wert)`, die genau dann `true` zurückgibt, wenn der Wert `wert` in der Liste vorhanden ist.

```
public boolean findeWert(int wert)
{









}
```

b) Implementieren Sie nun in der Klasse `Liste` die Methode `fuegeWer-tEin(int wert)`, die den Wert `wert` in die Liste einfügt, wenn dieser noch nicht in der Liste enthalten ist, d.h. Werte dürfen nicht doppelt in die Liste eingetragen werden.

```
public void fuegeWertEin(int wert)
{









}
```

c) Implementieren Sie nun in der Klasse `Liste` die Methode `loesche-Wert(int wert)`, die, falls der Wert `wert` in der Liste vorkommt, ein Element mit dem Wert `wert` aus der Liste löscht.

```
public void loescheWert(int wert)
{

}
```

## *Aufgabe 6 (Hashing)* *6 Punkte*

Es sei die folgende teilweise Implementierung einer Hashtabellen-Klasse gegeben, die Kollissionen mittels verketteter Listen auflöst. Die Klasse `Link` sei hierbei wie in Aufgabe 5 definiert.

```
public class HashTabelle
{
    private Link[] tabelle;
    public HashTabelle(int n)
    {
       tabelle = new Link[n];
    }
    public boolean suche(int d){
          // Aufgabe
    }
}
```

Ergänzen Sie die Klasse `HashTabelle` um die Methode `public boolean suche(int d)`, die das Suchen von Werten in der Hashtabelle implementiert, d.h. true zurückgibt, falls der Wert d in der Hash-Tabelle enthalten ist. Legen Sie hierzu eine geeignete Hashfunktion fest. Sie dürfen für die Implementierung der Methode `suche()` die Klasse `Liste` aus Aufgabe 5 benutzen.

Hashfunktion:

```
public boolean suche(int d)
{

}
```

Name, Vorname, Matrikelnummer Studiengang

## *Aufgabe 7 (Sortieren) 16 Punkte*

a) Sortieren Sie das folgende Feld nach dem Heapsort-Verfahren aufsteigend. Markieren Sie vor jedem Versickerungsvorgang das zu versickernde Element durch Einkreisen. Markieren Sie die Zeile, in der der erste Teil des Verfahrens, d.h. das Erstellen des Heaps, beendet ist. Trennen Sie danach bereits sortierte Teilfelder durch einen senkrechten Strich voneinander und notieren Sie die Reihenfolge zusätzlich auch vor jedem Tauschvorgang. Die Tabelle enthält mehr Zeilen als für eine vollständige Lösung erforderlich sind.

| 1 | 8 | 5 | 6 | 2 | 9 | 7 |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

b) Sortieren Sie die folgenden Werte mit Quicksort aufsteigend. Schreiben Sie nach jeder Runde (Teilen des Feldes) die neue Reihenfolge auf. Markieren Sie das jeweilige Pivotelement durch Einkreisen und trennen Sie die Teilfelder durch Striche. Das Pivotelement soll jeweils als mittleres Element des Teilfeldes gewählt werden. Gibt es zwei mittlere Elemente, so soll das linke der beiden Elemente gewählt werden. Die Tabelle enthält mehr Zeilen als für eine vollständige Lösung erforderlich sind.

| 3 | 8 | 5 | 6 | 2 | 9 | 7 |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

Name, Vorname, Matrikelnummer Studiengang

## *Aufgabe 7 (B-Bäume)* *12 Punkte*

a) Zeichnen Sie den B-Baum der Ordnung 1, der entsteht, wenn man in einen anfangs leeren Baum nacheinander die Schlüssel `3`, `17`, `9` einfügt.

b) Zeichnen Sie den B-Baum, nachdem zusätzlich die Schlüssel `21`, `1`, `29`, `30`, `10`, `23` und `22` (in dieser Reihenfolge) eingefügt wurden.

c) Zeichnen Sie nun den B-Baum, nachdem die Schlüssel `29` und `10` gelöscht wurden.

d) Zeichnen Sie nun den B-Baum, nachdem die Schlüssel `17` und `30` gelöscht wurden.