Name, Vorname, Matrikelnummer Studiengang

Klausur
Algorithmen und Datenstrukturen
SS 2019 – 05.02.2020

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 (evtl. auf den Rückseiten) 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)	12	
2 (Graphen)	18	
3 (Listen)	16	
4 (Komplexität)	12	
5 (Rekursion und Komplexität)	10	
6 (B-Bäume)	15	
7 (Sortieren)	10	
8 (Bäume)	12	
Summe	105	

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 1 (Multiple Choice, Wissen) *12 Punkte*

a) Kreuzen Sie in den folgenden Teilaufgaben an, welche Lösungen 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	**Richtig**	**Falsch**
Ein Stapel (engl. stack) arbeitet nach dem FIFO (first in-frist out)-Prinzip.		
Eine Warteschlange (engl. queue) arbeitet nach dem FIFO-Prinzip.		
Ein Stapel arbeitet nach dem LIFO (last in-first out) Prinzip.		
Eine Warteschlange arbeitet nach dem LIFO-Prinzip.		

Bewerten Sie folgende Aussagen	**Richtig**	**Falsch**
Bubblesort (Sortieren durch Austauschen) ist ein elementares Sortierverfahren.		
Quicksort ist ein Divide-and-Conquer-Algorithmus.		
Die Zeitkomplexität elementarer Sortierverfahren ist im schlechtesten Fall O(n log n).		
Die Zeitkomplexität elementarer Sortierverfahren ist im schlechtesten Fall $O(n^2)$.		

Bewerten Sie folgende Aussagen	**Richtig**	**Falsch**
Collections vom Typ LinkedList und ArrayList unterscheiden sich im Speicherverbrauch nicht.		
Der Zugriff auf das i-te Element ist in einer ArrayList in Zeit O(1) möglich.		
Der Zugriff auf das i-te Element ist in einer LinkedList in Zeit O(1) möglich.		
Das Suchen nach einem Element in einer ArrayList besitzt im schlechtesten Fall eine Zeitkomplexität von O(n).		

Welche der folgenden Aussagen sind korrekt?	**Richtig**	**Falsch**
Die Probleme, die zur Komplexitätsklasse P gehören, sind effizient lösbar.		
Das Halte-Problem gehört zur Komplexitätsklasse P.		
Jede Funktion in NP ist auch berechenbar.		
P ist eine Teilmenge von NP.		

Name, Vorname, Matrikelnummer Studiengang

b) Sie haben ein Array mit unsortierten Werten gegeben. Welches elementare Suchverfahren setzen Sie ein, um festzustellen, ob ein Wert im Array vorhanden ist (mit Begründung).

c) Sie haben ein Array mit sortierten Werten gegeben. Welches elementare Suchverfahren setzen Sie ein, um auch im schlechtesten Fall effizient festzustellen, ob ein Wert im Array vorhanden ist (mit Begründung).

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 2 (Graphen) *18 Punkte*

Folgender Graph ist durch seine Adjazenzmatrix gegeben:

Von \ Nach	A	B	C	D	E
A		6	2	7	
B			3		2
C				1	
D		1			
E	4		3		

a) Zeichnen Sie den gegebenen Graphen.

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

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.

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 3 (Listen) *16 Punkte*

Betrachten Sie die folgende Implementierung einer einfach verketteten Liste:

```
public class Link
{
    public int daten;
    public Link naechster;

    public Link(int daten, Link naechster)
    {
        this.daten = daten;
        this.naechster = naechster;
    }
}

public class Liste
{
   private Link anfang, ende;

    public Liste(){
       anfang = ende = null;
    }

   public Link getAnfang(){
       return anfang;
    }

   public Link getEnde(){
       return ende;
    }

   public void setAnfang(Link neu){
       anfang = neu;
    }

   public void setEnde(Link neu){
       ende = neu;
    }

   public int zaehlen(int wert)
    {
       // Aufgabenteil a)
    }

   public void verketten(Liste liste2)
    {
       // Aufgabenteil b)
    }

   public void anhaengenKopie(Liste liste2)
    {
       // Aufgabenteil c)
    }
}
```

Name, Vorname, Matrikelnummer Studiengang

a) Ergänzen Sie die Klasse `Liste` um eine Methode `zaehlen`, die zählt, wie oft `wert` in der Liste enthalten ist.

```
public int zaehlen(int wert)
{









}
```

b) Ergänzen Sie die Klasse `Liste` um eine Methode `verketten`, welche die Elemente von `liste2` am Ende der Liste anhängt. `liste2` soll nach dem Verketten leer sein.

```
public void verketten(Liste liste2)
{









}
```

c) Ergänzen Sie die Klasse `Liste` um eine weitere Methode, die Kopien aller Elemente von `liste2` am Ende der Liste anhängt.
Hinweis: `liste2` darf nicht verändert werden!

```
public void anhaengenKopie(Liste liste2)
{









}
```

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 4 (Komplexität) *12 Punkte*

Bestimmen Sie für die nachstehenden Prozeduren folgende Informationen:

- Berechnen Sie die Anzahl der Aufrufe von tuwas() für n=8.
- Bestimmen Sie die Anzahl der Aufrufe von tuwas() als Funktion von n. Hinweis: Sie können für Ihre Formel davon ausgehen, dass n eine Zweierpotenz ist.
- Bestimmen Sie die asymptotische Zeitkomplexität in O-Notation unter der Annahme, dass die asymptotische Zeitkomplexität von tuwas() O(1) ist.

```
void proz1(int n)
{
    for (int a=0; a<n; a++)
       tuwas();

    for (int b=0; b<=2*n; b++)
       tuwas();
}
```

```
void proz2(int n)
{
    for (int a=0; a<=n; a++)
       for (int b=0; b<=n*n; b++)
           tuwas();
    }

}
```

```
void proz3(int n)
{
    for (int a=1; a<=n; a++)
        for (int b=a; b<n; b++)
            tuwas();

}
```

```
void proz4(int n)
{
    for (int a=0; a<n; a++)
        for (int b=1; b<=n; b*=2)
            tuwas();
}
```

Methode	**Anzahl Aufrufe für n=8**	**Als Funktion von n**	**O-Notation**
proz1			
proz2			
proz3			
proz4			

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 5 (Rekursion und Komplexität) *10 Punkte*

Betrachten Sie die folgende Java-Klassenmethode `ausgabe`.

```
public static void ausgabe(int n)
{
      System.out.print(n + " ");
      if (n>1)
      {
            ausgabe(n-1);
            System.out.print(n + " ");
      }
}
```

a) Welche Ausgabe ergibt sich für einen Aufruf ausgabe(3) und ausgabe(4)?

ausgabe(3):

ausgabe(4):

b) Wie viele Zahlen werden in Abhängigkeit von n ausgeben?

c) Wie ist die Zeitkomplexität der Methode `ausgabe` in O-Notation?

d) Wie groß ist die Rekursionstiefe in Abhängigkeit von n?

e) Programmieren Sie die Methode `ausgabeIter` iterativ in Java, sodass sie die gleiche Ausgabe wie `ausgabe` erzeugt und eine Zeitkomplexität von O(n) besitzt.

```
public static void ausgabeIter(int n)
{




}
```

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 6 (B-Bäume) *15 Punkte*

Betrachten Sie die folgende teilweise Implementierung eines B-Baumes:

```
public class BKnoten
{
   public int[] schluessel;
   public BKnoten[] kinder;

   // Für Blätter
   public BKnoten(final int[] schluessel)
   {
      assert(schluessel!=null);

      this.schluessel = schluessel;
      this.kinder = new BKnoten[schluessel.length + 1];
   }

   // Für innere Knoten
   public BKnoten(final int[] schluessel, final BKnoten[] kinder)
   {
      assert(schluessel != null);
      assert(kinder != null);
      assert(schluessel.length+1 == kinder.length);

      this.schluessel = schluessel;
      this.kinder = kinder;
   }
}

public class BBaum
{
   private BKnoten wurzel;

   public int bestimmeGroesstenSchluessel() {
      // Aufgabenteil a)
   }

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

   private boolean istBlatt(BKnoten knoten){
      // Aufgabenteil b)
   }

   private int bestimmeAnzahlBlaetter(BKnoten knoten) {
      // Aufgabenteil c)
   }
}
```

Wichtiger Hinweis:
Das Array `schluessel` ist für jeden Knoten vollständig mit Schlüsseln gefüllt, kann aber für verschiedene Knoten eine unterschiedliche Länge besitzen!

Name, Vorname, Matrikelnummer Studiengang

a) Ergänzen Sie die Klasse BBaum um eine Methode
`public int bestimmeGroesstenSchluessel()`,
die iterativ den größten Schlüssel im BBaum bestimmt.
Hinweis: Sie können davon ausgehen, dass die Wurzel mindestens einen Schlüssel enthält.

```
public int bestimmeGroesstenSchluessel()
{
      assert(wurzel != null);









}
```

b) Ergänzen Sie die Klasse BBaum um eine Methode
`private boolean istBlatt(BKnoten knoten)`,
die prüft, ob `knoten` ein Blatt ist.

```
private boolean istBlatt(BKnoten knoten)
{









}
```

Name, Vorname, Matrikelnummer Studiengang

c) Ergänzen Sie die Klasse BBaum um eine Methode
private int bestimmeAnzahlBlaetter(BKnoten knoten),
die rekursiv die Anzahl der Blätter im Teilbaum mit der Wurzel `knoten` bestimmt.

```
private int bestimmeAnzahlBlaetter(BKnoten knoten)
{

}
```

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 7 (Sortieren) *10 Punkte*

a) In der folgenden Tabelle ist die Implementierung einer Variante eines bekannten Sortieralgorithmus angegeben. Ergänzen Sie die Tabelle um den Namen des Verfahrens, die asymptotische Anzahl an Vergleichen im besten (best case) und im schlechtesten Fall (worst case) sowie die Angabe, ob das Verfahren stabil ist.

Implementierung	**Name**	**Anz. Vergleiche** *O-Notation*		**Stabil** *Ja/Nein*
		Best case	**Worst case**	

`public static void sortiere(int[] feld)`
`{`
`  for (int i=feld.length-1; i>0; i--)`
`  {`
`  int m=i;`
`  for (int j=i-1; j>=0; j--)`
`  if (feld[j]>feld[m])`
`  m=j;`
`  int tmp=feld[m];`
`  feld[m]=feld[i];`
`  feld[i]=tmp;`
`  }`
`}`				

b) Wenden Sie die Implementierung aus Aufgabenteil a) auf das unten angegebene Feld an. Beachten Sie die Sortierrichtung und geben Sie die Reihenfolge der Schlüssel nach jedem Durchlauf der äußeren Schleife an. Benutzen Sie einen senkrechten Strich **|**, um Teilfelder voneinander abzutrennen. Die Tabelle enthält mehr Zeilen als erforderlich sind.

4	**8**	**5**	**9**	**6**	**3**	**2**	**8**	**7**

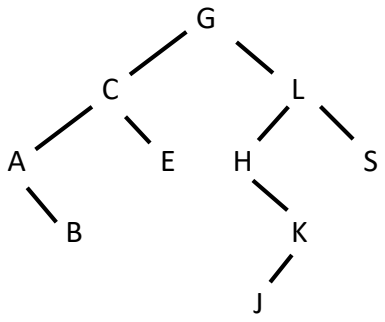
Name, Vorname, Matrikelnummer Studiengang

Aufgabe 8 (Bäume) *12 Punkte*

a) Konstruieren Sie einen binären Suchbaum mit minimaler Knotenanzahl, der kein AVL-Baum ist. Geben Sie für jeden Knoten den Schlüssel an.

b) Konstruieren Sie einen binären Suchbaum mit minimaler Knotenanzahl, der kein Heap ist. Geben Sie für jeden Knoten den Schlüssel an.

c) Gegeben sei der folgende binäre Suchbaum. Geben Sie für jede Traversierungsstrategie jeweils die Reihenfolge der Schlüssel an.



Pre-Order:

In-Order:

Post-Order:

d) Löschen Sie aus dem Baum aus Teilaufgabe c) den Schlüssel A und zeichnen Sie den entstehenden Baum.

e) Löschen Sie anschließend den Schlüssel G und zeichnen Sie den entstehenden Baum.