

Klausur
Algorithmen und Datenstrukturen
SS 2024 – 20.08.2024

Hinweise:

- Die Bearbeitungszeit beträgt **90 Minuten!**
- Zum Bestehen der Klausur sind 50 Punkte erforderlich.
- Erlaubte Hilfsmittel: keine
- Alle vorgegebenen und zu erstellenden Programmtexte beziehen sich auf die Programmiersprache Java. **Aus der Java-Klassenbibliothek darf nur die Klasse `Math` verwendet werden!**
- Bevor Sie mit der Bearbeitung der Aufgaben beginnen, **müssen** Sie auf allen Blättern Ihren Namen, Ihre Matrikelnummer und Ihren Studiengang eintragen.
- Der Klausurtext enthält ausreichend Platz zur Lösung der Aufgaben. Sollten Sie trotzdem zusätzliches Papier benötigen, wenden Sie sich an die Klausuraufsicht. Die Nutzung eigenen Papiers ist nicht gestattet.
- Sollte Ihre Lösung nicht unmittelbar unter oder neben der Aufgabenstellung stehen, machen Sie bitte einen entsprechenden Hinweis. Streichen Sie diejenigen Teile der von Ihnen geschriebenen Texte deutlich durch, die nicht in die Bewertung eingehen sollen.
- Bitte schreiben Sie deutlich.

Viel Erfolg!

Aufgabe	Maximalpunkte	Erreichte Punkte
1 (Wissen, Multiple Choice)	15	
2 (Komplexität, Rekursion)	15	
3 (Listen)	18	
4 (B-Bäume)	22	
5 (Suchen)	12	
6 (Graphen)	18	
Bonuspunkte	10	
Summe	110	

Diese Seite ist absichtlich leer!

Aufgabe 1 (Wissen, Multiple Choice)**15 Punkte**

Bearbeiten Sie die folgenden Multiple-Choice-Aufgaben und beachten Sie dabei die folgenden Bewertungsregeln:

- Für jede richtige Antwort erhalten Sie die genannte Anzahl an Punkten.
- Bei Überschreitung der genannten Anzahl an Antworten wird die Teilaufgabe mit 0 Punkten bewertet. Beispiel: zwei Antworten gefordert, drei Antworten gegeben => 0 Punkte!
- Falsche Antworten führen nicht zu Punktabzug.

a) Welche zwei der folgenden Aussagen charakterisiert den Begriff Algorithmus am besten?

(2 Antworten, 2 Punkte pro richtiger Antwort)

☐	Ein Algorithmus muss vor der Ausführung in seine elementaren Algorithmenbausteine zerlegt werden.
☐	Ein Algorithmus löst ein allgemeines Problem (eine Problemklasse).
☐	Ein Algorithmus muss für jede zulässige Eingabe nach endlich vielen Schritten ein Ergebnis liefern und terminieren.
☐	Ein Algorithmus definiert für eine gegebene Problemklasse die beste Lösungsmöglichkeit.
☐	Ein Algorithmus beschreibt die Lösung eines Problems in einer konkreten Programmiersprache.
☐	Ein Algorithmus ist eine vage Beschreibung der Problemlösung, die problemspezifisch vervollständigt werden muss.

b) Sie müssen $n/2$ viele Datensätze in einem ungeordneten Array mit der Länge n suchen. Welche zwei der folgenden Kombinationen von Sortiert- und Suchalgorithmen sind bezogen auf ihre asymptotische Worst-Case-Komplexität in dieser Situation am schnellsten?

(2 Antworten, 3 Punkte pro richtiger Antwort)

☐	Quicksort mit anschließender binärer Suche
☐	Heapsort mit anschließender linearer Suche
☐	Bubblesort mit anschließender binärer Suche
☐	Mergesort mit anschließender binärer Suche
☐	Insertionsort mit anschließender Interpolationssuche
☐	Bottom-Up-Heapsort mit anschließender binärer Suche

c) Welche der folgenden Aussagen über Datenstrukturen sind richtig und welche sind falsch?

(3 Antworten, 1 Punkt pro richtiger Antwort)

richtig	falsch	Aussage
☐	☐	Eine einfach verkettete Liste besteht aus einer Menge von Elementen, wobei jedes Element mit maximal einem Nachfolger verknüpft ist.
☐	☐	Ein Binärbaum besteht aus einer Menge von Elementen, wobei jeder Knoten im Baum bis zu zwei Binärbäume als Kindknoten haben kann.
☐	☐	Ein Heap besteht aus einer Menge von Knoten, wobei für jeden Knoten gilt, dass die Schlüssel der linken Kinder kleiner und die Schlüssel der rechten Kinder größer sind als der Schlüssel des Elternknotens.

d) Welche der folgenden Aussagen charakterisieren einen AVL-Baum korrekt?

(1 Antwort, 2 Punkte pro richtiger Antwort)

☐	Im AVL-Baum sind die Blätter von links nach rechts absteigend sortiert gespeichert.
☐	Im AVL-Baum darf sich die Höhe der Teilbäume unter der Wurzel um maximal 1 unterscheiden.
☐	Im AVL-Baum darf sich die Höhe der Teilbäume an jedem Knoten um maximal 1 unterscheiden.
☐	Im AVL-Baum darf die Höhe des Baumes unter der Wurzel den Wert $\log(\text{Anzahl Knoten})$ nicht unterschreiten.
☐	Im AVL-Baum muss für jeden Knoten gelten, dass sein Schlüssel größer oder gleich den Schlüsseln seiner Kinder ist.

Aufgabe 2 (Komplexität, Rekursion)**15 Punkte**

Bestimmen Sie für die nachstehenden Prozeduren folgende Informationen:

- die Anzahl der Aufrufe von `tuwas()` für $n=4$.
- die Anzahl der Aufrufe von `tuwas()` als Funktion von n . Sie können für die Formeln davon ausgehen, dass n eine Zweierpotenz ist.
- die asymptotische Zeitkomplexität in O-Notation unter der Annahme, dass die **Methode `tuwas()` eine asymptotische Zeitkomplexität von $O(1)$ besitzt.**

```
static void proz1(int n)
{
    for (int i=n; i>0; i/=2){
        tuwas();
    }
}
```

```
static void proz2(int n)
{
    if (n > 0)
    {
        proz2(n-1);
        proz2(n-1);
    }
    else
        tuwas();
}
```

a) Notieren Sie die Lösung in der folgenden Tabelle.

Methode	Anzahl Aufrufe für $n=4$	Anzahl Aufrufe als Funktion von n	Zeitkomplexität in O-Notation
<code>proz1</code>			
<code>proz2</code>			

b) Bestimmen Sie die asymptotische Zeitkomplexität von `proz1` in O-Notation, wenn `tuwas()` durch `tuwas(n)` ersetzt wird und `tuwas(n)` eine asymptotische Zeitkomplexität von $O(n)$ besitzt.

c) Bestimmen Sie die maximale Rekursionstiefe von `proz2` für $n=4$.

Aufgabe 3 (Listen)**18 Punkte**

Betrachten Sie die folgende teilweise Implementierung einer einfach verketteten Liste:

```
public class Link<T>
{
    public T daten;
    public Link<T> naechster;

    public Link(T daten, Link<T> naechster){
        this.daten = daten;
        this.naechster = naechster;
    }
}

public class Liste<T>
{
    public Link<T> anfang;

    public Liste() {
        anfang = null; // leere Liste
    }

    public Link<T> ende(){
        // Aufgabenteil a)
    }

    public void verketten(Liste<T> zweiteListe){
        // Aufgabenteil b)
    }
}
```

- a) Vervollständigen Sie die Methode `ende`, welche einen Zeiger (eine Referenz) auf das letzte Element zurückgibt. Ist die Liste leer, soll null zurückgegeben werden.

```
public Link<T> ende()  
{
```

```
}
```

- b) Vervollständigen Sie die Methode `verketten`, welche die Elemente der zweiten Liste am Ende anhängt. Die zweite Liste soll nach dem Verketten leer sein.
Hinweis: Nutzen Sie die Methode aus Aufgabenteil a).

```
public void verketten(Liste<T> zweiteListe)  
{  
    assert(this != zweiteListe);
```

```
}
```

Aufgabe 4 (B-Bäume)**22 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 boolean suchenIter(int wert) {
        // Aufgabenteil a)
    }

    public String absteigend() {
        return absteigend(wurzel);
    }

    private String absteigend(BKnoten k) {
        // Aufgabenteil b)
    }
}
```

Wichtiger Hinweis:

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

Eine rekursive Suche im B-Baum nach einem Wert sieht wie folgt aus:

```
public boolean suchen(int wert)
{
    return suchen(wurzel, wert);
}

private boolean suchen(BKnoten k, int wert)
{
    if (k == null) return false;

    for (int a=0; a < k.schluessel.length; a++)
    {
        if (wert == k.schluessel[a])
            return true;
        if (wert < k.schluessel[a])
            return suchen(k.kinder[a], wert);
    }

    return suchen(k.kinder[k.schluessel.length], wert);
}
```

a) Vervollständigen Sie die Methode `suchenIter`, die iterativ nach dem Wert im B-Baum sucht.

```
public boolean suchenIter(int wert)
{
```

```
}
```

- b) Vervollständigen Sie die Methode `absteigend`, welche rekursiv die Schlüssel der Knoten im Teilbaum mit der Wurzel `k` absteigend sortiert in einer Zeichenkette sammelt. Die Schlüssel sollen durch ein Leerzeichen voneinander getrennt werden.

Tipp: Zeichnen Sie sich als Beispiel einen kleinen B-Baum.

```
public String absteigend(BKnoten k)
{
```

```
}
```

Aufgabe 5 (Suchen)**12 Punkte**

- a) Führen Sie auf dem folgenden Feld eine binäre Suche nach dem Schlüssel 13 durch. Geben Sie für jeden Schritt den Indexbereich der Suche und den Index des Vergleichselements an.

0	1	2	3	4	5	6	7	8
2	5	6	9	10	12	15	16	18

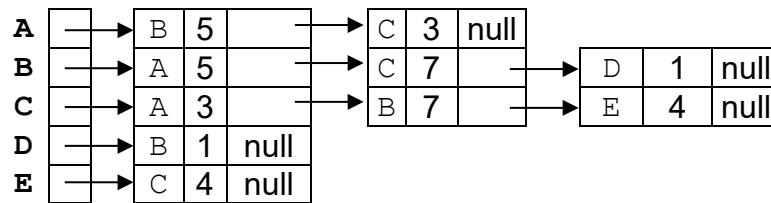
- b) Tragen Sie nacheinander die folgenden Schlüssel in der angegebenen Reihenfolge in die Hash-Tabelle ein: 5, 12, 23, 1 und 15. Die Auflösung von Kollisionen soll durch Quadratisches Sondieren erfolgen. Geben Sie für jeden Schlüssel die Berechnung des Index in die Tabelle an.

Als Hash-Funktion soll $h(k) = k \bmod 7$ verwendet werden.

0	1	2	3	4	5	6

Aufgabe 6 (Graphen)**18 Punkte**

Folgender Graph ist durch seine Adjazenzlistenstruktur gegeben:



a) Geben Sie die zugehörige Adjazenzmatrix an.

b) Zeichnen Sie den gegebenen Graphen. Nutzen Sie alle Informationen aus der Adjazenzlistenstruktur.

- c) Kann der Graph als ungerichteter Graph betrachtet werden? Begründen Sie dies anhand der Adjazenzmatrix.
- d) Führen Sie auf dem obigen Graphen den Algorithmus von Kruskal zur Bestimmung eines minimalen Spannbaumes aus. Geben Sie die Kantenreihenfolge der Bearbeitung, den minimalen Spannbaum sowie die Union-Find-Datenstruktur am Ende des Algorithmus' an.

Name, Vorname, Matrikelnummer

Studiengang

Zusatzblatt