

Klausur
Algorithmen und Datenstrukturen
WS 2024/25 – 31.01.2025

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 (Graphen)	18	
4 (Listen)	18	
5 (B-Bäume)	22	
6 (Suchen)	12	
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 löst ein allgemeines Problem (eine Problemklasse).
☐	Ein Algorithmus beschreibt die Lösung eines Problems in einer konkreten Programmiersprache.
☐	Ein Algorithmus muss für jede zulässige Eingabe nach endlich vielen Schritten ein Ergebnis liefern und terminieren.
☐	Ein Algorithmus ist eine vage Beschreibung der Problemlösung, die problemspezifisch vervollständigt werden muss.
☐	Ein Algorithmus definiert für eine gegebene Problemklasse die beste Lösungsmöglichkeit.
☐	Ein Algorithmus muss vor der Ausführung in seine elementaren Algorithmenbausteine zerlegt werden.

b) Welche der folgenden Aussagen über die asymptotische Komplexität von Algorithmen sind richtig und welche sind falsch?

(4 Antworten, 1,5 Punkt pro richtiger Antwort)

richtig	falsch	Aussage
☐	☐	Quicksort hat im Worst- und Average-Case eine Laufzeit von $O(n \log n)$
☐	☐	Heapsort hat im Worst- und Average-Case eine Laufzeit von $O(n \log n)$
☐	☐	Lineare Suche hat im Average-Case eine Laufzeit von $O(\log \log n)$
☐	☐	Suche in einer Hashtabelle hat im Worst- und Average-Case eine Laufzeit von $O(1)$.

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 genau einem Nachfolger verbunden ist.
☐	☐	Ein Binärbaum besteht aus einer Menge von Elementen, wobei jedes Element mit höchstens einem Elternknoten verbunden ist.
☐	☐	Ein Graph besteht aus einer Menge von Knoten und Kanten, wobei jeder Knoten mit mindestens einer Kante verbunden sein muss.

d) Welche der zwei folgenden Aussagen zu rekursiven bzw. iterativen Implementierung von Algorithmen sind korrekt?

(2 Antworten, 1 Punkt pro richtiger Antwort)

☐	Heapsort kann iterativ implementieren werden.
☐	Quicksort kann iterativ, aber nicht rekursiv implementiert werden.
☐	Binäres Suchen kann rekursiv und iterativ implementiert werden.
☐	Mergesort kann rekursiv, aber nicht iterativ implementiert werden.
☐	Die Fakultätsfunktion ($n! = n \times (n-1) \times \dots \times 1$) kann ausschließlich iterativ implementiert werden.
☐	Pre-Order-Traversierung eines binären Suchbaums kann ausschließlich rekursiv implementiert werden.

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=1; i<=n; i*=2){
        tuwas();
    }
}
```

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

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, wobei `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 (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) Zeichnen Sie den Graphen, der sich aus der Matrix ergibt. Nutzen Sie dabei alle Informationen aus der Adjazenzmatrix.

- b) Kann man für den 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. Ergänzen Sie dazu die folgende Tabelle.

Neu markierter Knoten	A length/pred	B length/pred	C length/pred	D length/pred	E length/pred

- d) Bestimmen Sie auf Basis der Tabelle aus Aufgabenteil c) den kürzesten Weg von Knoten A zum Knoten E und dessen Länge. Erläutern Sie kurz, wie Sie dabei vorgehen.

Aufgabe 4 (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 int laenge(){
        // Aufgabenteil a)
    }

    public Liste<T> teilen(){
        // Aufgabenteil b)
    }
}
```

- a) Vervollständigen Sie die Methode `laenge`, welche die Anzahl der Elemente in der Liste zurückgeben soll.

```
public int laenge()
{
```

```
}
```

- b) Vervollständigen Sie die Methode `teilen`, welche die Liste in der Mitte unterteilen soll. Die Elemente der vorderen Hälfte bleiben in der Liste. Die Elemente der zweiten Hälfte sollen in einer neuen Liste als Ergebnis zurückgegeben werden. Bei ungerader Elementzahl soll die erste Liste ein Element mehr enthalten. Ist die ursprüngliche Liste leer, so soll eine leere Liste zurückgegeben werden.

Hinweis: Nutzen Sie die Methode aus Aufgabenteil a)

```
public Liste<T> teilen()  
{
```

```
}
```

Aufgabe 5 (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 int hoehe() {
        // Aufgabenteil a)
    }

    public boolean hatOrdnungM(int m) {
        return hatOrdnungM(wurzel, m, true);
    }

    private boolean hatOrdnungM(BKnoten k, int m, boolean istWurzel)
    {
        // Aufgabenteil b)
    }
}
```

Wichtiger Hinweis:

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

a) Vervollständigen Sie die Methode `hoehe`, die iterativ die Höhe des B-Baumes bestimmt.

```
public int hoehe()  
{
```

```
}
```

- b) Vervollständigen Sie die Methode `hatOrdnungM`, welche rekursiv überprüft, ob alle Knoten des Baumes die Ordnung m besitzen. Diese bedeutet, dass die Wurzel zwischen 1 und $2m$ Schlüssel und jeder andere Knoten zwischen m und $2m$ Schlüssel enthält. Für einen leeren Baum soll das Ergebnis `true` sein.

```
private boolean hatOrdnungM(BKnoten k, int m, boolean istWurzel)
{
```

```
}
```

Aufgabe 6 (Suchen)**12 Punkte**

- a) Führen Sie auf dem folgenden Feld a eine Interpolationssuche nach dem Schlüssel $k=7$ durch.

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

Die Formel für den Index m des Vergleichselements lautet wie folgt, wobei k der gesuchte Schlüssel ist:

$$m = l + \left\lfloor \frac{k - a[l].\text{Schlüssel}}{a[r].\text{Schlüssel} - a[l].\text{Schlüssel}} \cdot (r - l) \right\rfloor$$

Geben Sie für jeden Schritt den Indexbereich der Suche als Intervall $[l, r]$ und den Index m des Vergleichselements m an.

l	r	m

Hinweis: Die Tabelle enthält mehr Zeilen als notwendig.

- b) Tragen Sie nacheinander die folgenden Schlüssel in der angegebenen Reihenfolge in die Hash-Tabelle unterhalb ein: 5, 12, 20, 1 und 15. Die Auflösung von Kollisionen soll durch *Lineares Sondieren* mit Schrittweite $c = 3$ erfolgen. Geben Sie für jeden eingefügten 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

Zusatzblatt