

Nachklausur
Algorithmen und Datenstrukturen
SS 2025 – 06.02.2026

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)	20	
5 (Binärer Suchbaum)	20	
6 (Hashtabelle)	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) Eine verkettete Liste mit n Einträgen soll an eine verkettete Liste mit m Einträgen angehängt werden. Geben Sie die Laufzeitkomplexität dieser Operation an, wenn beide verketteten Listen mit Ende-Zeiger implementiert wurden.

(1 Antwort, 3 Punkte)

☐	$O(1)$
☐	$O(n)$
☐	$O(m)$
☐	$O(n + m)$
☐	$O(n \cdot m)$
☐	$O(\log m)$

b) Existiert ein AVL-Baum mit den genannten Eigenschaften?

Als Erinnerung: Ein Baum hat Höhe 1 genau dann, wenn er nur aus Wurzelknoten besteht.

(2 Antworten, 2 Punkte pro richtige Antwort)

☐	Jeder Knoten hat 3 Kindknoten und genau einen Elternknoten.
☐	Jeder Knoten hat höchstens zwei Kindknoten und mindestens einen Elternknoten
☐	Der AVL-Baum besitzt genau 2^{n+1} Knoten und hat Höhe n (n kann jede positive ganze Zahl sein)
☐	Der AVL-Baum besitzt genau $2^n - 1$ Knoten und hat Höhe n (n kann jede positive ganze Zahl sein)
☐	Der AVL-Baum besitzt genau 2^n Knoten und hat Höhe n (n kann jede positive ganze Zahl sein)
☐	Der Wurzelknoten hat keinen Elternknoten.

c) In einem Suchbaum, in dem kein Schlüssel mehrfach vorkommt, wähle einen festen Knoten, der weder Wurzelknoten noch Blatt ist und der einen nichtleeren linken und rechten Teilbaum besitzt. Dann gilt zwangsläufig immer, dass...

(5 Antworten, 1 Punkt pro richtige Antwort)

richtig	falsch	Aussage
☐	☐	Alle Schlüssel in dem linken Teilbaum sind kleiner als alle Schlüssel entlang des Pfades des Knotens zum Wurzelknoten
☐	☐	Alle Schlüssel in dem linken Teilbaum sind größer als alle Schlüssel entlang des Pfades des Knotens zu einem beliebigen Blatt
☐	☐	Der kleinste Schlüssel eines linken Teilbaums ist immer kleiner als der kleinste Schlüssel des rechten Teilbaums
☐	☐	Der größte Schlüssel eines linken Teilbaums ist immer kleiner als der kleinste Schlüssel des rechten Teilbaums
☐	☐	Der größte Schlüssel eines linken Teilbaums ist immer größer als der kleinste Schlüssel des rechten Teilbaums

d) Welche der folgenden Aussagen sind korrekt?

(3 Antworten, 1 Punkt pro richtige Antwort)

☐	Bubblesort hat die gleiche Worst-Case Komplexität wie Quicksort
☐	Bubblesort und Heapsort haben die gleiche Worst-Case Komplexität
☐	Insertionsort und Selectionsort haben die gleiche Worst-Case Komplexität
☐	Pre-Order Traversierung und Post-Order Traversierung eines binären Suchbaums haben die gleiche Worst-Case Komplexität.
☐	Die Worst-Case Komplexität der binären Suche ist $O(\log n)$, wobei n der größte auftretende Wert eines Schlüssels ist.
☐	Die Worst-Case Komplexität der binären Suche ist $O(\log n)$, wobei n die Anzahl der zu durchsuchenden Schlüssel ist.

Anmerkung: Diese Aufgabe besitzt 4 statt 3 korrekte Antworten. Dies wurde bei der Korrektur berücksichtigt.

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--)
    {
        for (int j=1; j<=n; j*=2)
        {
            tuwas();
        }
    }
}
```

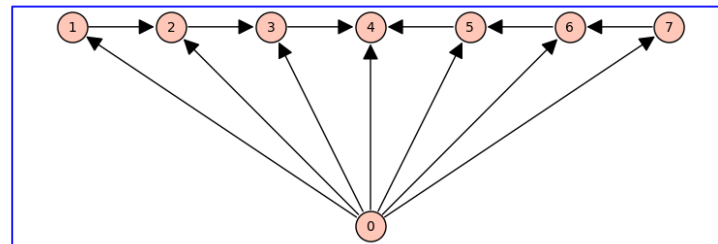
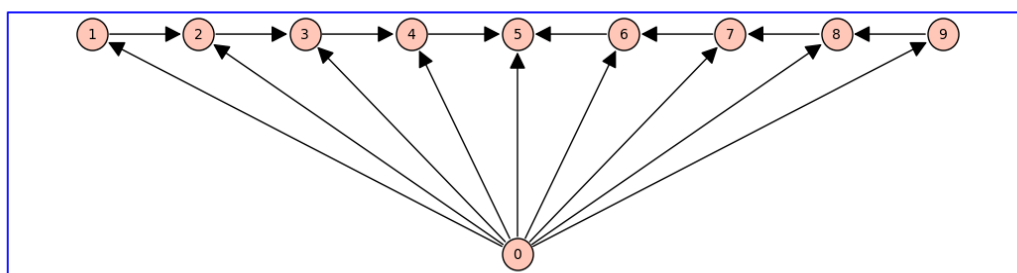
```
static void proz2(int n)
{
    if (n > 0)
    {
        proz2(n-1);
        tuwas();
    }
}
```

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
proz1			
proz2			

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**Der Graph $A(4)$ istund der Graph $A(5)$ ist

und analog gibt es Graphen $A(n)$, wobei in der oberen Zeile die Knoten nach dem Schema $1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow n$ und $n \leftarrow n+1 \leftarrow \dots \leftarrow 2 \cdot n - 1$ verkettet sind. Stets gibt es unten einen weiteren Knoten 0, der zu jedem anderen Knoten eine gerichtete Kante besitzt. Die obigen Abbildungen zeigen die Beispiele $A(4)$ und $A(5)$ ganz konkret.

a) Zeichnen Sie den entsprechenden Graphen für $n=3$.

b) Geben Sie die Adjazenzmatrix des Graphen für $n=3$ an.

c) Ist der Graph $A(3)$ topologisch sortierbar?

☐ Ja

☐ Nein

d) Wie viele Kanten besitzt der Graph $A(12)$?

Antwort: _____

Skizze (freiwillig):

e) Ist es bei dem Graphen $A(1000)$ möglich, eine gerichtete Kante (Pfeil) zusätzlich in den Graphen einzufügen, sodass der resultierende Graph nicht topologisch sortierbar ist? Geben Sie eine konkrete gerichtete Kante (Pfeil) an, wenn dies möglich ist, und erklären Sie Ihre Antwort.

☐ Das ist nicht möglich

☐ Das ist möglich, z.B. durch Hinzufügen eines Pfeils von

_____ nach _____

Erklärung:

f) Entfernen Sie den Knoten mit der Zahl 60 aus dem Graphen $A(60)$ mitsamt aller in ihn eingehenden oder ausgehenden Kanten. Wie viele Zusammenhangskomponenten besitzt der resultierende Graph?

Antwort: _____

g) Entfernen Sie die Knoten mit den Zahlen 0, 10, und 20 aus dem Graphen $A(120)$ mitsamt aller in sie eingehenden oder ausgehenden Kanten. Wie viele Zusammenhangskomponenten besitzt der resultierende Graph?

Antwort: _____

Aufgabe 4 (Listen)**20 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
    }

    private void einfuegen(T daten)
    {
        // Aufgabenteil a)
    }

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

- a) Vervollständigen Sie die Methode `einfuegen`, die ein neues Element am Anfang der Liste einfügt.

```
private void einfuegen(T daten)
{

}

}
```

- b) Vervollständigen Sie die Methode `invertieren`, die eine Kopie der Liste erzeugt, welche die Elemente in umgekehrter Reihenfolge enthält.

Hinweis: Nutzen Sie die Methode aus Aufgabenteil a)

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

```
}
```

Aufgabe 5 (Binärer Suchbaum)**20 Punkte**

Betrachten Sie die folgende teilweise Implementierung eines binären Suchbaumes:

```
public class Knoten
{
    public int schluessel;
    public Knoten[] kinder = new Knoten[2];
    // Achtung: Kindknoten sind in einem Feld (Array) gespeichert
}

public class BinSuchBaum
{
    private Knoten wurzel;

    ...
    public int kleinsterSchluessel() {
        // Aufgabenteil a)
    }

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

    private boolean istInnererKnoten(Knoten knoten) {
        // Aufgabenteil b)
    }

    private int anzahlInnereKnoten(Knoten knoten) {
        // Aufgabenteil c)
    }
}
```

- a) Vervollständigen Sie die Methode `kleinsterSchluessel`, die iterativ den kleinsten Schlüssel im Binärbaum bestimmen soll.

Hinweis: Sie können davon ausgehen, dass die Wurzel existiert.

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

    ...

}
```

- b) Vervollständigen Sie die Methode `istInnererKnoten`, die prüft, ob `knoten` ein innerer Knoten ist, d.h. mindestens ein Kind besitzt.

Hinweis: Sie können davon ausgehen, dass der Knoten existiert.

```
private boolean istInnererKnoten(Knoten knoten)
{
    assert(knoten != null);
```

```
}
```

- c) Vervollständigen Sie die Methode `anzahlInnereKnoten`, die rekursiv die Anzahl der inneren Knoten im Teilbaum mit der Wurzel `knoten` bestimmen soll.

Hinweis: Nutzen Sie die Methode aus Aufgabenteil b)

```
private int anzahlInnereKnoten(Knoten knoten)
{
```

```
}
```

Aufgabe 6 (Hash-Tabelle)**12 Punkte**

Wir erhalten die bereits gefüllte Hash-Tabelle

0	leer
1	31
2	leer
3	43
4	4
5	leer
6	6
7	16
8	leer
9	9

Die Hashfunktion sei $h(x) = x \bmod 10$ (bzw. $x \% 10$). Zur Behandlung von Kollisionen wurde lineare Sondierung mit Schrittweite +1 genutzt.

a) Markieren Sie alle Einträge in der Hash-Tabelle **mit einem KREUZ X**, bei denen beim Einfügen eine Kollision stattgefunden haben *muss*.

b) Wir fügen nun die Zahl 26 in die Hash-Tabelle ein. An welche Position müssen wir sie setzen?

Antwort: _____

c) Wir fügen nun zusätzlich die Zahl 8 in die Hash-Tabelle ein. An welche Position müssen wir sie setzen?

Antwort: _____

d) Wir wollen nun eine zufällige Zahl im Bereich 0 bis 9 in die Hash-Tabelle zusätzlich einfügen. Wie groß ist die Wahrscheinlichkeit, dass eine Kollision auftritt?

Antwort: _____

e) Ist die folgende Aussage wahr?

Falls eine Hashtabelle (nicht notwendigerweise die oben betrachtete) in einem Zustand ist, sodass genau ein einziger Eintrag 'leer' ist und alle anderen belegt, hat dann lineares Sondieren mit Schrittweite +1 immer die Eigenschaft, dass ein weiteres zusätzliches Element auf die einzig verbleibende leere Position gelegt wird, ganz unabhängig vom Hashwert des neuen Eintrags?

☐ Ja

☐ Nein

Begründen Sie Ihre Antwort ausführlich:

Name, Vorname, Matrikelnummer

Studiengang

Zusatzblatt