

Klausur
Algorithmen und Datenstrukturen
SS 2023 – 02.08.2023

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)	20	
4 (Binäre Suchbäume)	20	
5 (Sortieren)	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, 1,5 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 ist eine vage Beschreibung der Problemlösung und muss problem-spezifisch vervollständigt werden.
☐	Ein Algorithmus muss vor der Ausführung in seine elementaren Algorithmenbausteine zerlegt werden.
☐	Ein Algorithmus muss für jede zulässige Eingabe nach endlich vielen Schritten ein Ergebnis liefern und terminieren.

b) Welche zwei der folgenden Aussagen über die Komplexität von Sortieralgorithmen treffen zu?

(2 Antworten, 2 Punkte pro richtiger Antwort)

☐	Im Worst-Case sind Quicksort und Bubblesort gleich schnell.
☐	Im Average-Case ist Insertionsort schneller als Bubblesort.
☐	Im Average-Case ist Quicksort schneller als Heapsort.
☐	Im Worst-Case sind Heapsort und Mergesort gleich schnell.
☐	Im Worst-Case ist Bottom-Up-Heapsort schneller als Heapsort.
☐	Im Average-Case ist Bubble-Sort schneller als Mergesort.

c) Welche zwei der folgenden Aussagen zum Suchverfahren auf Datenstrukturen treffen zu?

(2 Antworten, 2 Punkte pro richtiger Antwort)

☐	Lineares Suchen hat bei ungeordneten Schlüsseln im Average Case eine Komplexität von $O(\log_2(n))$.
☐	Binäres Suchen hat bei ungeordneten Schlüsseln im Average Case eine Komplexität von $O(n^2)$.
☐	Interpolationssuche hat bei geordneten Schlüsseln im Average Case eine Komplexität von $O(\log_2(n))$.
☐	Binäres Suchen hat bei geordneten Schlüsseln im Worst Case eine Komplexität von $O(\log_2(n))$.
☐	Lineares Suchen hat bei geordneten Schlüsseln im Worst Case eine Komplexität von $O(n)$.
☐	Interpolationssuche hat bei geordneten Schlüssel im Worst Case eine Komplexität von $O(n^2)$.

d) Welche zwei der folgenden Aussagen charakterisieren Aspekte des Hash-Verfahrens korrekt?

(2 Antworten, 2 Punkte pro richtiger Antwort)

☐	Für das Ablegen eines Datensatzes wird durch die Hash-Funktion für einen gegebenen Schlüssel eine Hash-Adresse berechnet.
☐	Für das Ablegen eines Datensatzes wird in der Hash-Tabelle der Schlüssel der Hash-Adresse nachgeschlagen.
☐	Bei einer Adresskollision ist das Quadratische Sondieren eine mögliche Strategie, um einen passenden Platz in der Hash-Tabelle zu ermitteln.
☐	Bei einer Adresskollision ist die Verwendung eines Hash-Puffers eine mögliche Strategie, um zusätzlichen Platz in der Hash-Tabelle zu erzeugen.
☐	Bei der Suche nach einem Schlüssel in einer Hash-Tabelle führt die Hash-Funktion das Verfahren der binären Suche durch.
☐	Bei der Suche nach einem Schlüssel in der Hash-Tabelle führt die Hash-Funktion das Verfahren der Interpolationssuche durch.

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 . Bei `proz2` können Sie für die Formel davon ausgehen, dass n eine Zweierpotenz ist.
- die asymptotische Zeitkomplexität in O-Notation unter der Annahme, dass die **asymptotische Zeitkomplexität von `tuwas(n)` $O(n)$** ist.

```
static void proz1(int n)
{
    for (int a=0; a<=n; a++){
        for (int b=1; b<n; b++)
            tuwas(n);
    }
}
```

```
static void proz2(int n)
{
    if (n > 1)
    {
        tuwas(n);
        proz2(n/2);
        proz2(n/2);
    }
}
```

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 maximale Rekursionstiefe von `proz2` für $n=4$.

c) Bestimmen Sie die maximale Rekursionstiefe von `proz2` in Abhängigkeit von n . Sie können voraussetzen, dass n eine Zweierpotenz ist.

Aufgabe 3 (Listen)**20 Punkte**

Betrachten Sie die folgende teilweise Implementierung einer Warteschlange als einfach verkettete Liste:

```
public class Person
{
    private String vorname, name;

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

    public boolean equals(Person p) {
        return this.vorname.equals(p.vorname) &&
            this.name.equals(p.name);
    }
}

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 Warteschlange
{
    private Link<Person> anfang;

    public Warteschlange(){
        anfang = null; // Leere Warteschlange
    }

    public Person dequeue(){
        // Aufgabenteil a)
    }

    public void enqueue(Person p){
        assert(p != null);
        // Aufgabenteil b)
    }
}
```

Hinweis: Die verkettete Liste besitzt keinen ende-Zeiger!

- a) Vervollständigen Sie die Methode `dequeue`, welche die Person am Anfang der Warteschlange zurückgibt und aus der Warteschlange entfernt. Ist die Warteschlange leer, soll `null` zurückgegeben werden.

```
public Person dequeue()
{
```

}

- b) Vervollständigen Sie die Methode `enqueue`, welche eine Person am Ende der Warteschlange einfügt, wenn sie noch nicht in der Warteschlange enthalten ist.

```
public void enqueue(Person p)
{
    assert(p != null);
}
```

}

Aufgabe 4 (Binäre Suchbäume)**20 Punkte**

Betrachten Sie die folgende teilweise Implementierung eines binären Suchbaums:

```
public class Knoten
{
    public int schluessel;
    public Knoten linkesKind;
    public Knoten rechtesKind;
}

public class BinSuchbaum
{
    private Knoten wurzel;

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

    public boolean istAVL(){
        return istAVL(wurzel)>= 0;
    }

    private int istAVL(Knoten k){
        // Aufgabenteil c)
    }
}
```

Hinweis: Sie können davon ausgehen, dass

- **der Binärbaum ein Suchbaum ist**
- **die Schlüssel größer 0 sind.**

a) Vervollständigen Sie die Methode `maximum`, die iterativ den größten Schlüssel im Suchbaum bestimmt. Ist der Baum leer, so soll -1 zurückgegeben werden.

```
private int maximum()
{
```

```
}
```

b) In Teilaufgabe c) sollen Sie die Methode `istAVL` vervollständigen, die rekursiv für den Baum mit der Wurzel `knoten` wie folgt einen Wert berechnet:

- Ist der Baum ein AVL-Baum, so soll seine Höhe als Ergebnis zurückgeliefert werden.
- Ist der Baum kein AVL-Baum, so soll -1 zurückgegeben werden.

Hinweis: Ein leerer Baum ist ein AVL-Baum.

Welche Fälle müssen Sie im Rekursionsschritt (nicht-leerer Baum) unterscheiden?

c) Vervollständigen Sie die Methode `istAVL`:

```
public int istAVL(Knoten knoten)
{
```

```
}
```

Aufgabe 5 (Sortieren)**12 Punkte**

Gegeben ist die folgende Implementierung des Algorithmus' cocktailSort zum Sortieren von Feldern.

```
static void cocktailSort(int[] array) {
    boolean getauscht = true;
    int start = 0;
    int ende = array.length-1;

    while (getauscht && start < ende) {
        getauscht = false;

        for (int i = start; i < ende; ++i) {
            if (array[i] > array[i+1]) {
                int temp = array[i];
                array[i] = array[i+1];
                array[i+1] = temp;
                getauscht = true;
            }
        }
        ende--;

        for (int i = ende-1; i >= start; i--) {
            if (array[i] > array[i+1]) {
                int temp = array[i];
                array[i] = array[i+1];
                array[i+1] = temp;
                getauscht = true;
            }
        }
        start++;
    }
}
```

- a) Geben Sie die asymptotische Anzahl an Vergleichen im besten Fall (best case) und im schlechtesten Fall (worst case) in O-Notation für ein Feld mit n Elementen an.

Bester Fall:

Schlechtester Fall:

- b) Wenden Sie die Implementierung aus Aufgabenteil a) auf das unten angegebene Feld an. Geben Sie die Reihenfolge der Schlüssel nach jeder Iteration jeder For-Schleife sowie nach jeder Änderung der Variablen `start` und `ende` an. Benutzen Sie einen senkrechten Strich `|`, um unsortierte und sortierte Teilfelder voneinander abzutrennen. Die Tabelle enthält mehr Zeilen als erforderlich sind.

[illegible]

Aufgabe 6 (Graphen)**18 Punkte**

Folgender Graph G ist durch seine Adjazenzmatrix gegeben:

		Nach					
		A	B	C	D	E	F
Von	A		45		33		72
	B						
	C	49			83	24	
	D		39	83			
	E						
	F			69			

a) Entscheiden und begründen Sie anhand der Adjazenzmatrix, ob der Graph gerichtet oder ungerichtet ist und gewichtet oder ungewichtet ist.

b) Zeichnen Sie den Graphen, der sich aus der Matrix ergibt.

- c) Kann man für obigen Graphen eine topologische Sortierung angeben (Begründung)? Falls eine topologische Sortierung existiert, so geben Sie eine topologische Sortierung an.
- d) Betrachten Sie nun den G zu Grunde liegenden ungerichteten Graphen G' . Zeichnen Sie den minimalen Spannbaum, der sich aus dem Algorithmus von Kruskal ergibt. Teilgraphen und Union-Find-Datenstruktur müssen nicht angegeben werden.

Name, Vorname, Matrikelnummer

Studiengang

Zusatzblatt