Name, Vorname, Matrikelnummer Studiengang

# Klausur
# Algorithmen und Datenstrukturen
# WS 2023/24 – 09.02.2024

## 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) | 20 | |
| 4 (Binäre Suchbäume) | 20 | |
| 5 (Sortieren) | 12 | |
| 6 (Graphen) | 18 | |
| Bonuspunkte | 10 | |
| **Summe** | 110 | |

Name, Vorname, Matrikelnummer Studiengang

***Diese Seite ist absichtlich leer!***

Name, Vorname, Matrikelnummer Studiengang

## *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)*

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

***b) Welche zwei der folgenden Aussagen über die asymptotische Laufzeitkomplexität von Sortieralgorithmen treffen zu?***

*(2 Antworten, 2 Punkte pro richtiger Antwort)*

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

Name, Vorname, Matrikelnummer Studiengang

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

*(2 Antworten, 2 Punkte pro richtiger Antwort)*

| | |
|---|---|
| O | Binäres Suchen hat bei geordneten Schlüsseln im Average Case eine Komplexität von $O(n^2)$. |
| O | Binäres Suchen hat bei geordneten Schlüsseln im Worst Case eine Komplexität von $O(\log_2(n))$ |
| O | Interpolationssuche hat bei geordneten Schlüssel im Worst Case eine Komplexität von $O(n^2)$. |
| O | Interpolationssuche hat bei gleichverteilten Schlüsseln im Average Case eine Komplexität von $O(\log_2(n))$. |
| O | Lineare Suche hat bei ungeordneten Schlüsseln im Worst Case eine Komplexität von $O(\log_2(n))$. |
| O | Lineare Suche hat bei geordneten Schlüsseln im Wort Case eine Komplexität von $O(n)$. |

### *d) Welche der folgenden Aussagen zur rekursiven bzw. iterativen Implementierung von Algorithmen sind richtig und welche sind falsch?*

*(6 Antworten, 0,5 Punkte pro richtiger Antwort)*

| **richtig** | **falsch** | **Aussage** |
|---|---|---|
| O | O | Die Fakultätsfunktion (n! = n x (n-1) x … x 1) kann iterativ, aber nicht rekursiv implementiert werden. |
| O | O | Heapsort kann iterativ und rekursiv implementiert werden. |
| O | O | Lineares Suchen kann iterativ und rekursiv implementiert werden. |
| O | O | Mergesort kann rekursiv, aber nicht iterativ implementiert werden. |
| O | O | Post-Order-Traversierung eines binären Suchbaums kann rekursiv und iterativ implementiert werden. |
| O | O | Pre-Order-Traversierung eines binären Suchbaums kann rekursiv, aber nicht iterativ implementiert werden |

Name, Vorname, Matrikelnummer Studiengang

## *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 **Methode tuwas() eine asymptotische Zeitkomplexität von O(1)** besitzt.

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

}
```

```
static void proz2(int n)
{
    if (n > 1)
    {
        tuwas();
        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** |
|---|---|---|---|
| `proz1` | | | |
| `proz2` | | | |

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 eines Auftragsstapels als einfach verkettete Liste:

```
public class Auftrag
{
   private int id;
   private String bezeichnung;

   public Auftrag(int id, String bezeichnung) {
      this.id = id;
      this.bezeichnung = bezeichnung;
   }

   public boolean equals(Auftrag a) {
      return this.id == a.id &&
             this.bezeichnung.equals(a.bezeichnung);
   }
}

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 AuftragsStapel
{
   private Link<Auftrag> oben;

   public AuftragsStapel() {
      oben = null;  // leerer Stapel
   }

   public void push(Auftrag a){
      assert(a != null);
      // Aufgabenteil a)
   }

   public Auftrag pop(){
      // Aufgabenteil b)
   }
}
```

Name, Vorname, Matrikelnummer Studiengang

a) Vervollständigen Sie die Methode `push`, welche den Auftrag oben auf den Stapel legt. Ist dieser schon vorher im Stapel enthalten, wird er an der bisherigen Stelle entfernt.

```
public void push(Auftrag a)
{
    assert(a != null);









}
```

b) Vervollständigen Sie die Methode `pop`, welche den Auftrag, der oben auf dem Stapel liegt als Ergebnis zurückgibt und diesen vom Stapel entfernt. Ist kein Auftrag vorhanden, so soll `null` zurückgegeben werden.

```
public Auftrag pop()
{









}
```

Name, Vorname, Matrikelnummer Studiengang

## *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 boolean istLineareListe(){
      // Aufgabenteil a)
   }

   private String absteigend(Knoten k){
      // Aufgabenteil b)
   }

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

}
```

**Hinweis: Sie können davon ausgehen, dass der Binärbaum ein Suchbaum ist.**

---

a) Vervollständigen Sie die Methode `istLineareListe`, die iterativ prüft, ob der Baum zu einer linearen Liste degeneriert ist.
**Hinweis**: Ein leerer Baum ist eine lineare Liste.

```
public boolean istLineareListe()
{
```

```
}
```

b) Vervollständigen Sie die Methode `absteigend`, welche rekursiv die Schlüssel der Knoten im Teilbaum mit der Wurzel k absteigend in einer Zeichenkette sammelt. Die Schlüssel sollen durch ein Leerzeichen voneinander getrennt werden.
**Tipp**: Zeichnen Sie sich als Beispiel einen kleinen Suchbaum.

```
public String absteigend(Knoten k)
{




}
```

Name, Vorname, Matrikelnummer Studiengang

## *Aufgabe 5 (Sortieren)* *12 Punkte*

Gegeben ist die folgende Implementierung des Algorithmus‘ gnomeSort zum Sortieren von Feldern.

```
static void gnomeSort(int[] array)
{
   int i = 0;

   while (i < array.length) {
      if (i == 0 || array[i-1] <= array[i])
         i++;
      else
      {
         int temp = array[i];
         array[i] = array[i-1];
         array[i-1] = temp;
         i--;
      }
   }
}
```

a) Wenden Sie die gegebene Implementierung von gnomeSort auf das unten angegebene Feld an. Geben Sie den Wert von i und die Reihenfolge der Schlüssel am Ende jeder Iteration der While-Schleife an. Die Tabelle enthält mehr Zeilen als erforderlich sind.

| **i** | **array** | | | |
|---|---|---|---|---|
| 0 | 65 | 34 | 23 | 43 |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

| | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

b) 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:

Name, Vorname, Matrikelnummer Studiengang

## *Aufgabe 6 (Graphen)* *18 Punkte*

Folgender Graph G ist durch seine Adjazenzmatrix gegeben:

| Von \ Nach | A | B | C | D | E |
|---|---|---|---|---|---|
| **A** | | 9 | 5 | | |
| **B** | | | 2 | 1 | |
| **C** | | 3 | | 9 | 2 |
| **D** | | | | | 4 |
| **E** | | | | 6 | |

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

b) Zeichnen Sie den Graphen, der sich aus der Matrix ergibt. Nutzen Sie alle Informationen aus der Adjazenzmatrix.

Name, Vorname, Matrikelnummer Studiengang

c) Führen Sie auf dem obigen Graphen den Algorithmus von Dijkstra zur Bestimmung aller minimalen 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 zu Knoten D. Geben Sie die besuchten Knoten in der richtigen Reihenfolge und die Gesamtlänge des Weges an. Erläutern Sie kurz, wie Sie dabei vorgehen.

Name, Vorname, Matrikelnummer Studiengang

Name, Vorname, Matrikelnummer Studiengang

## *Zusatzblatt*