Name, Vorname, Matrikelnummer Studiengang

Klausur
Algorithmen und Datenstrukturen
SS 2022 – 11.08.2022

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.
- 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)	16	
2 (Komplexität, Rekursion)	14	
3 (Listen)	20	
4 (Binärbäume)	25	
5 (Suchen)	12	
6 (Graphen)	18	
Summe	105	

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 1 (Wissen) *16 Punkte*

a) Notieren Sie zu jeder Aussage, ob sie richtig oder falsch ist.

1. $(n+1)\cdot(n+1) = O(n^2)$
2. $n^2 = O(n\cdot\log(n))$
3. $n^2 = O(2^n)$

b) Ergänzen Sie in der Tabelle die Zeitkomplexität bei n enthaltenen Objekten für den schlechtesten Fall (worst case) in O-Notation.

	Binärer Suchbaum	AVL-Baum
Einfügen		
Suchen		

c) Formulieren Sie die charakteristische Eigenschaft von List<E>, Set<E> und Map<K,V> jeweils in einem Satz.

List<E>:

Set<E>:

Map<K,V>:

d) Welche Eigenschaften muss ein Binärbaum erfüllen, damit er ein Heap ist?

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 2 (Komplexität, Rekursion) *14 Punkte*

Bestimmen Sie für die nachstehenden Prozeduren folgende Informationen:

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

}
```

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

- Berechnen Sie die Anzahl der Aufrufe von tuwas() für n=4.
- Bestimmen Sie die Anzahl der Aufrufe von tuwas() als Funktion von n.
- Bestimmen Sie die asymptotische Zeitkomplexität in O-Notation unter der Annahme, dass die asymptotische Zeitkomplexität von tuwas() O(1) ist.

a) Notieren Sie die Lösung in der folgenden Tabelle.

Methode	**Anzahl Aufrufe für n=4**	**Anzahl Aufrufe als Funktion von n**	**O-Notation**
`proz1`			
`proz2`			

b) Bestimmen Sie die Rekursionstiefe von proz2 für `n=4`.

c) Bestimmen Sie die Rekursionstiefe von proz2 in Abhängigkeit von `n`.

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 3 (Listen) *20 Punkte*

Betrachten Sie die folgende teilweise Implementierung einer Prioritätsliste für Aktionen als einfach verkettete Liste:

```
public class Link
{
    public String aktion;
    public int prioritaet;
    public Link naechster;

    public Link(String aktion, int prioritaet, Link naechster){
        this.aktion = aktion;
        this.prioritaet = prioritaet;
        this.naechster = naechster;
    }
}

public class Prioritaetsliste
{
   private Link anfang;

    public Prioritaetsliste(){
       anfang = null; // Leere Liste
    }
    …
    public void einfuegen(String aktion, int prioritaet){
        assert(aktion!=null);
       // Aufgabenteil a)
    }

    public String entferneAktionMitHoechsterPrioritaet(){
       // Aufgabenteil b)
    }
}
```

Wichtig:
Die Aktionen sollen in der Liste nach absteigender/fallender Priorität gespeichert werden!
Bei gleicher Priorität ist die Reihenfolge beliebig.

a) Vervollständigen Sie die Methode `einfuegen`, welche die Aktion an einer passenden Stelle (absteigende Priorität) in die verkettete Liste einfügt. **Hinweis**: Sie können davon ausgehen, dass der Parameter `aktion` ungleich null ist.

```
public void einfuegen(String aktion, int prioritaet)
{
    assert(aktion!=null);









}
```

b) Vervollständigen Sie die Methode `entferneAktionMitHoechsterPrioritaet`, welche die Aktion mit höchster Priorität aus der Liste entfernt und als Ergebnis zurückgibt. Ist die Liste leer, soll null zurückgegeben werden.

```
public String entferneAktionMitHoechsterPrioritaet()
{









}
```

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 4 (Binärbäume) *25 Punkte*

Betrachten Sie die folgende teilweise Implementierung eines Binärbaums:

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

public class Binaerbaum
{
   private Knoten wurzel;


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

   private int hoehe(Knoten knoten)
   {
      // Aufgabenteil a)
   }

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

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

   private int anzahlBlaetter(Knoten k)
   {
      // Aufgabenteil c)
   }

   public boolean istVollstaendig()
   {
      // Aufgabenteil d)
   }

}
```

Name, Vorname, Matrikelnummer Studiengang

a) Vervollständigen Sie die Methode `hoehe`, die rekursiv die Höhe des Baums mit der Wurzel `knoten` bestimmen soll. **Hinweis**: Die Höhe eines leeren Baums ist 0.

```
private int hoehe(Knoten knoten)
{









}
```

b) Vervollständigen Sie die Methode `istBlatt`, die prüft, ob `knoten` ein Blatt ist.

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









}
```

c) Vervollständigen Sie die Methode `anzahlBlaetter`, die rekursiv die Anzahl der Blätter im Teilbaum mit der Wurzel `knoten` bestimmen soll.

```
private int anzahlBlaetter(Knoten knoten)
{




}
```

d) Vervollständigen Sie die Methode `istVollstaendig`, die prüft, ob der Baum vollständig ist. Nutzen Sie die vorhandenen Methoden. **Hinweis**: Ein leerer Baum gilt als vollständig.

```
public boolean istVollstaendig()
{




}
```

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 5 (Suchen) *12 Punkte*

a) Führen Sie auf dem folgenden Feld eine binäre Suche nach dem Schlüssel 9 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: 4, 13, 26, 8, 24 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 mod 11 verwendet werden.

0	1	2	3	4	5	6	7	8	9	10

Name, Vorname, Matrikelnummer Studiengang

Aufgabe 6 (Graphen) *18 Punkte*

Folgender Graph G ist durch seine Adjazenzmatrix gegeben:

		Nach				
	A	**B**	**C**	**D**	**E**	**F**
Von **A**		1				
B			1	1		
C						1
D					1	1
E						
F					1	

a) Zeichnen Sie den gegebenen Graphen.

b) Kann man für obigen Graphen eine topologische Sortierung angeben (Begründung)? Falls eine topologische Sortierung existiert, so geben Sie eine topologische Sortierung an.

c) Betrachten Sie nun den G zu Grunde liegenden ungerichteten Graphen G‘. Führen Sie auf G‘ eine Breitensuche beginnend ab E durch und notieren Sie für jeden Schritt den Zustand der Warteschlange. Bei Wahlmöglichkeiten verwenden Sie die alphabetische Reihenfolge.