Klausur 2017WS ALGDS

Samstag, 21. Januar 2017 19:58

Aufgabe 1
Geben Sie exakt an, wie oft der Befehl **addiere 1 zu sum** ausgeführt wird und welche **Zeitkomplexität** das darstellt.

a) Setze sum = 0
Wiederhole für alle i von 1 bis n
  Wiederhole für alle j von 1 bis n
    addiere 1 zu sum

b) Setze sum = 0
Wiederhole für alle i von 1 bis n
  Wiederhole für alle j von 1 bis 5
    addiere 1 zu sum

c) Setze sum = 0
Wiederhole für alle i von 1 bis n
  Wiederhole für alle j von 1 bis i
    addiere 1 zu sum

Aufgabe 2

Sie haben zwei Rechner zur Verfügung, R1 und R2. R1 ist ca. um einen Faktor 100 schneller als R2.

Für die Lösung eines Problems stehen zwei Algorithmen zur Verfügung, A1 und A2. A1 ist ein O(n)-Algorithmus, A2 ein $O(n^2)$-Algorithmus. Leider läuft der schnelle Algorithmus A1 nur auf dem langsamen Rechner R2.

Zeigen Sie, welcher Kombination für den **Wert n=**10 sowie für den **Wert n=1000** schneller das Ergebnis liefert.
Zeigen Sie ebenfalls, bei welchem n-Wert beide Kombinationen das gleiche Ergebnis liefern

Aufgabe 3: Suchbäume

a) Bilden Sie einen Suchbaum mit den folgenden Schlüsseln **6, 1, 4, 5, 9, 2, 8, 10. Dieses Reihenfolge beachten!**
b) Zeigen und Erklären Sie, wie viele Suchbäume es mit **5 Knoten** und den Schlüsseln **1, 2, 3, 4, 5** gibt**.**

Aufgabe 4: Türme von Hanoi (16 Punkte)

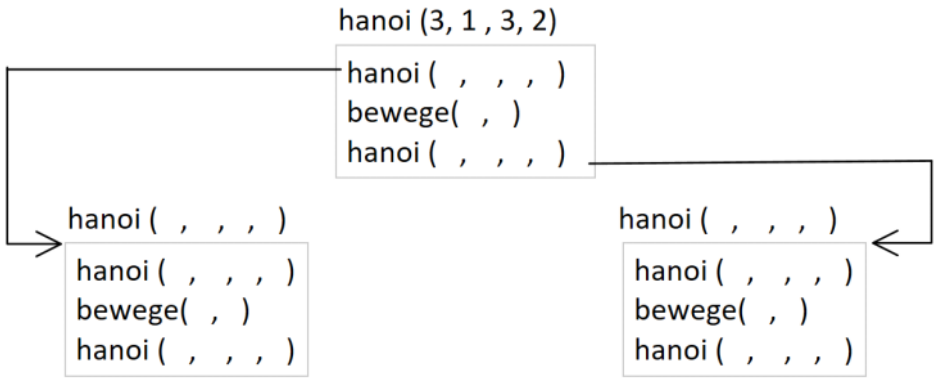
a) Ablauf der Rekursion
Erklären Sie den Ablauf des Programms der Türme von Hanoi, in dem Sie die Werte einiger Instanzen des rekursiven Aufrufs in das folgende Schema eintragen. Wie Sie in dem Schema sehen könne, soll die Methode für das Versetzen eines Turms der Größe **3** von **Stapel 1** nach **Stapel 3** aufgerufen werden.

Zur Erinnerung der in der Vorlesung besprochene Programmcode:

```
static void hanoi(int groesse, int von, int nach, int arrber){
   if(groesse > 0){
```

```
            hanoi(groesse-1, von, arbber, nach);
            bewege(von, nach);
            hanoi(groesse-1, arbber, nach, von);
        }
    }
```

Lösung



Aufgabe 5

In dem Projekt **Aufgab05** finden Sie eine Klasse **Liste,** die eine verkettete Liste von Integern darstellt. Ändern Sie die Liste ab, so dass als Nutzinhalt Objekte der Klasse Student gespeichert werden könne.

Testen Sie mit dem Programm **ListeStudentTestprogramm** (das nicht verändert werden darf).

Hinweis: es sind sowohl die Klasse Liste als auch die Klasse Knoten zu ändern.

Wichtig: Bitte tragen Sie als erstes Ihre Matrikelnummer in eine Kommentarzeile ein.

`Liste.java` (Rekonstruktion)

```
public class Liste {
   private Knoten anker;
public Liste() {
      this.anker = null;
   }

   // MatNr: 6999999
public void einfuegenAnfang (int Zahl) {
      this.anker = new Knoten (Zahl, this.anker);
   }
}
```

`Knoten.java` (Rekonstuktion)

```
public class Knoten {
   int Zahl;
   Knoten Nf;
public Knoten(int Zahl, Knoten Nf) {
      this.Zahl = Zahl;
      this.Nf = Nf;
   }
}
```

`ListeStudentTestprogramm.java` (Rekonstruktion)

```
public class ListeStudentTestprogramm {
```

```
   public static void main(String[] args) {
      Liste L = new Liste ();
      Student studi = new Student("Karla Karlson",4711);
      Student weitererStudi = new Student("Herbert Feuerstein",4712);
      L.einfuegenAnfang(studi);
      L.einfuegenAnfang(weitererStudi);
   }
}
```

Klausur WS2015

Aufgabe 1: Anzahl Schleifen (14 Punkte)
Ermitteln Sie in drei verschiedenen Fällen, wie oft der Befehl **addiere 1 zu sum** ausgeführt wird. Begründen Sie jeweils ihre Antwort!

a) (Punkte)

Setze sum = 0
Wiederhole für alle i von 1 bis n
  Wiederhole für alle j von 1 bis n
    addiere 1 zu sum

Lösung

erstes i	erstes j	letztes j	anzahl j	summe j
1	1	n	n	n
2	1	n	n	2n
3	1	n	n	3n
…	…	…	…	…
n	1	n	n	n * n

=> Zeitkomplexität: n * n => $O(n^2)$

b) (4 Punkte)
Setze sum = 0
Wiederhole für alle i von 1 bis n
  Wiederhole für alle j von 1 bis 5
    addiere 1 zu sum

Lösung

erstes i	erstes j	letztes j	anzahl j	summe j
1	1	5	5	5
2	1	5	5	5+5
3	1	5	5	5+5+5
…	…	…	…	…
n	1	5	5	n * 5

=> Zeitkomplexität: n * 5 => $O(n)$

c) (6 Punkte)
Setze sum = 0
Wiederhole für alle i von 1 bis n
  Wiederhole für alle j von 1 bis i
    addiere 1 zu sum

Lösung:

erstes i	erstes j	letztes j	anzahl j	summe j
1	1	1	1	1
2	1	2	2	1+2
3	1	3	3	1+2+3
…	…	…	…	…
n	1	n	n	1+2+3..+n

=> Zahlen von 1..n aufsummieren → Gauß'sche Summenformel: $\frac{n(n+1)}{2}$

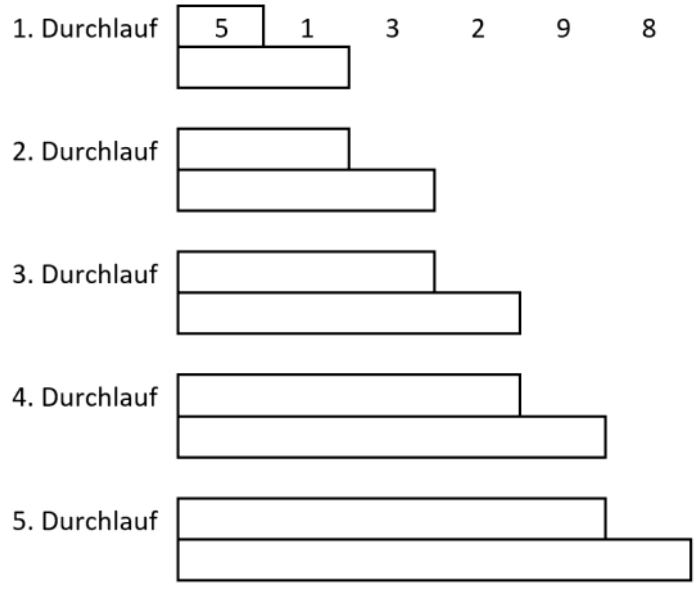
Zeitkomplexität: $\frac{n(n+1)}{2} = \frac{1}{2} * n^2 + \frac{1}{2} * n \Rightarrow in\ O\text{-}Notation: O(n^2)$

Aufgabe 2: Insertion Sort (10 Punkte)

Erklären Sie den Ablauf des Sortierens eins Arrays nach der Methode **Insertion Sort** in dem unten stehenden Schema.

Wie Sie in der ersten Zeile sehen können, soll das Array {5, 1, 3, 2, 9, 8} sortiert werden.

Lösung



1. Durchlauf	5	1	3	2	9	8
	1	5	3	2	9	8
2. Durchlauf	1	5	3	2	9	8
	1	3	5	2	9	8
3. Durchlauf	1	3	5	2	9	8
	1	2	3	5	9	8
4. Durchlauf	1	2	3	5	9	8
	1	2	3	5	9	8
5. Durchlauf	1	2	3	5	9	8
	1	2	3	5	8	9

Aufgabe 3: Türme von Hanoi (16 Punkte)

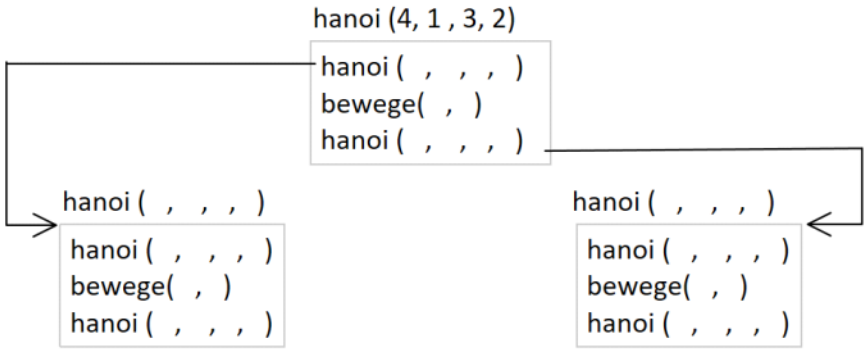
a) Ablauf der Rekursion

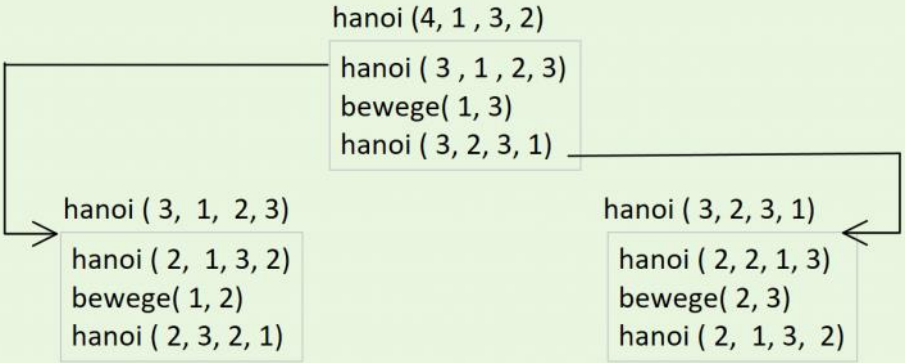
Erklären Sie den Ablauf des Programms der Türme von Hanoi, in dem Sie die Werte einiger Instanzen des rekursiven Aufrufs in das folgende Schema eintragen. Wie Sie in dem Schema sehen könne, soll die Methode für das Versetzen eines Turms der Größe 4 von Stapel 1 nach Stapel 3 aufgerufen werden.

Zur Erinnerung der in der Vorlesung besprochene Programmcode:

```
static void hanoi(int groesse, int von, int nach, int arrber){
   if(groesse > 0){
      hanoi(groesse-1, von, arbber, nach);
      bewege(von, nach);
      hanoi(groesse-1, arbber, nach, von);
   }
}
```

Lösung





a) Zeitkomplexität (6 Punkte)

Welche Zeitkomplexität besitzt der Algorithmus? Begründen Sie ihre Antwort!

Lösung:

Scheiben	Schritte	als Zweierpotenz
1	1	$2^1 - 1$

2	3	$2^2 - 1$
3	7	$2^3 - 1$
4	15	$2^4 - 1$
n		$2^n - 1$

=> Zeitkomplexität $O(2^n)$

(Die Konstante 1 entfällt)

Aufgabe 4: Verkettete Liste (20 Punkte)

a) Maximum (10 Punkte)

In dem Projekt **Aufgabe04a** finden Sie eine Klasse **Liste,** die eine verkettete Liste von Integern darstellt.

Schreiben Sie eine Methode **max** vom Typ integer, welche das Maximum der Werte in der Liste zurückgibt.

Testen Sie ihre Methode mit **ListeTestprogramm** (das nicht verändert werden darf). Testen Sie auch, dass Ihre Methode in den Grenzfällen (z. **B. der leeren Liste) richtig arbeitet.**

Wichtig: Bitte tragen Sie als erstes ihre Matrikelnummer in eine Kommentarzeile ein.

```
public int max() {
    int result = Integer.MIN_VALUE;
    Knoten aktuellerKnoten = this.anker;

    while(aktuellerKnoten != null){
            if(aktuellerKnoten.Zahl >= result){
                    result = aktuellerKnoten.Zahl;
            }
            aktuellerKnoten = aktuellerKnoten.Nf;
    }

    return result;
  }
```

b) Ändern Nutzinhalt

In dem Projekt **Aufgab04b** finden Sie eine Klasse **Liste,** die eine verkettete Liste von Integern darstellt. Ändern Sie die Liste ab, so dass als Nutzinhalt Objekte der Klasse Student gespeichert werden könne.

Testen Sie mit dem Programm **ListeStudentTestprogramm** (das nicht verändert werden darf).

Hinweis: es sind sowohl die Klasse Liste als auch die Klasse Knoten zu ändern.

Wichtig: Bitte tragen Sie als erstes Ihre Matrikelnummer in eine Kommentarzeile ein.

```
// Knoten.java
   - int Zahl
   + Student Zahl

// Liste.java

   - einfuegenAnfang (int Zahl)
   + einfuegenAnfang (Student Zahl)
```

```
- einfuegenEnde (int Zahl)
+ einfuegenEnde (Student Zahl)

// Suchmethode (ändern / neu programmieren)
public Student suchen(String schluessel) {

	Knoten knoten = this.anker;

	while(null != knoten){
		if(schluessel.equals(knoten.Zahl.getName())){
			return knoten.Zahl;
		}
		knoten = knoten.Nf;
	}

	return null;
}
```