Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme
Prof. Dr. A. Nürnberger

Magdeburg, 11.02.08

Prüfung

Algorithmen und Datenstrukturen I

Name:	Matrikelnummer:
Vorname:	Studiengang:
Blattanzahl:	
Unterschrift Student/in:	Unterschrift Aufsicht:

Aufg. 1	Aufg. 2	Aufg. 3	Aufg. 4	Summe
(von 9)	(von 10)	(von 10)	(von 11)	(von 40)

Allgemeine Hinweise

- Schreiben Sie auf jedes Blatt Ihren Namen, Ihre Matrikelnummer und die Seitennummer.
- Die Programme sind gut zu kommentieren! Programmstrukturen sind je nach Aufgabenstellung als Java-ähnlicher Pseudocode oder in Java anzugeben.
- Beginn und Ende einer Aufgabenlösung sind durch einen waagerechten Strich deutlich zu kennzeichnen. Ungültige Lösungen sind durchzustreichen.
- Zugelassene Hilfsmittel: ausschließlich Schreibmaterialien
- Die Klausur besteht aus 4 Aufgaben, die Bearbeitungszeit beträgt 120 Minuten.
- Bitte **nicht** mit grünem oder rotem Stift schreiben.

Aufgabe 1 (2 + 6 + 1 = 9 Punkte)

a) Erklären Sie das Prinzip des *binären Suchens* in einer Folge *a[0..m-1]*.
b) Schreiben Sie eine rekursive **und** eine iterative Java-Methode zum binären Suchen eines Elementes in einer Folge *a[0..m-1]*. Der Rückgabewert der Methoden ist die Position des ersten gefundenen Elementes oder *–1*, falls das Element nicht gefunden wurde. Die Elemente seien vom Typ *String*.
c) Wie viele Vergleiche sind beim binären Suchen in einer Folge von 1000 Elementen maximal erforderlich? Begründen Sie kurz.

Aufgabe 2 (2 + 1 + 3 + 1 + 2 + 1 = 10 Punkte)

a) Zeigen Sie anhand eines Bildes das Prinzip einer einfach verketteten Liste. Aus welchen Bestandteilen besteht diese?
b) Nennen Sie den Unterschied zwischen statischen und dynamischen Datenstrukturen. Wozu gehört die Liste? Begründen Sie das kurz.
c) Im Folgenden wird eine geordnete Liste betrachtet, deren Elemente aufsteigend geordnet sind. Schreiben Sie eine Java-Klasse *SortedList* mit einem geeigneten Konstruktor und der Methode *insert(T o)*, welche ein neues Element *o* geordnet in die Liste einfügt. Die Klasse *SortedList* verwaltet die geordnete Liste intern mit einer einfach verketteten Liste (*SimpleList*).
Hier die zu erweiternde Klasse SortedList:

```
public class SortedList<T extends Comparable<T>> {
        private SimpleList<T> list;
        …
}
```

d) Welche Komplexität hat Ihre *insert*-Methode? Begründen Sie kurz.
e) Geben Sie die Vor- und Nachbedingungen ihres *insert*-Algorithmus an. Wie lautet die Schleifen-Invariante?
f) Nehmen Sie an, Sie möchten die sortierte Liste in umgekehrter Reihenfolge ausgeben. Wie müsste man die Listenstruktur erweitern, damit dies effizient möglich wird?

Hinweis:
Die Klassen *SimpleList<T>* und *Node<T>* stehen mit den folgenden Methoden zur Verfügung:

```
Node<T>
        public Node(T o, Node<T> n);
        public Node();
        public void setElement(T o);
        public T getElement();
        public void setNext(Node<T> n);
        public Node<T> getNext();

SimpleList<T>
        public SimpleList();
        public boolean isEmpty();
        public int size();
        public Node<T> getHead();
        public void addFirst(T o);
        public T getFirst();
        public void addLast(T o);
```

Aufgabe 3 (8 + 2 = 10 Punkte)

a) Aus der Vorlesung ist folgende algebraische Spezifikation für den Abstrakten Datentyp (ADT) Nat bekannt:

type Nat
operators
0: $\rightarrow$ Nat
suc: Nat $\rightarrow$ Nat -- Nachfolger
add: Nat x Nat $\rightarrow$ Nat -- Addition
axioms $\forall$ i, j: Nat
add(i, 0) = i
add(i, suc(j)) = suc (add(i, j))

Erweitern Sie den ADT *Nat* um die Signaturen, Axiome und Vorbedingungen (Preconditions) für folgende Funktionen:

pred	-- Vorgänger
sub	-- Subtraktion
less	-- Vergleich (kleiner als)
div	-- Ganzzahldivision
even	-- Prädikat der Geradzahligkeit

b) Definieren Sie auf einem Alphabet bestehend aus Ziffern, Buchstaben und den Sonderzeichen „#“ und „*“ mit Hilfe der Backus-Naur-Form ein Konzept, das
- mit einem beliebigen Buchstaben oder einer Ziffer beginnt,
- dann das Zeichen „#“ gefolgt von 2 Ziffern und einem „*“ beinhaltet und
- mit einer beliebig langen Folge von Zeichen des Alphabets endet.

Aufgabe 4 (5 + 3 + 2 + 1 = 11 Punkte)

a) Erläutern Sie anhand eines Java-ähnlichen Pseudocodes die Funktionsweise der Sortierverfahren *QuickSort* und *BubbleSort*.
b) Betrachten Sie das Feld [3, 5, 35, 12, 15, 16, 20, 22] und wenden Sie darauf beide Sortierverfahren an. Geben Sie dabei alle Zwischenschritte an. Welches Sortierverfahren ist für dieses Feld besser geeignet und warum?
c) Geben Sie in O-Notation für beide Verfahren die Anzahl der notwendigen Vergleiche im durchschnittlichen Fall an, wobei *n* die Größe des zu sortierenden Feldes ist. Begründen Sie ihre Aussage.
d) Betrachten Sie einen Datensatz mit zwei Sortierschlüsseln A und B. Nacheinander wird erst nach Schlüssel A und dann nach Schlüssel B sortiert. Welches Sortierverfahren würden Sie anwenden, um bei Datensätzen mit identischen Schlüsseln auf B die Reihenfolge des Schlüssels A beizubehalten? Begründen Sie Ihre Antwort kurz.

Wir wünschen Ihnen viel Erfolg!