LV „ Algorithmen und Datenstrukturen“
Dr. D. Habich / Dipl.-Math. R. Vater

WS 09/10

Aufgabenblatt zur 9. Übung

Zeitraum: 14.12. bis 18.12.2009

1. Aufgabe: (AGS 5.3*)

Ein Feld a folgenden Typs sei gegeben:

```
typedef int feld[n];
```

Dieses Feld soll mit Hilfe eines Programms aufsteigend sortiert werden, das heißt es soll also nach Abschluss der Sortierung gelten: $a_0 \leq a_1 \leq ... \leq a_{n-1}$

(a) Schreiben Sie ein C-Programm, welches folgenden Lösungsgedanken (auch unter dem Namen Bubble–Sort bekannt) realisiert:
Durchlaufe das Feld mehrfach von links nach rechts und vertausche jeweils benachbarte Elemente, falls diese nicht in der gewünschten Ordnung stehen. Das Sortieren ist beendet, wenn in einem Durchlauf keine Vertauschung mehr stattfindet. Machen Sie sich zunächst anhand eines Beispiels mit diesem Algorithmus vertraut.

(b) Ermitteln Sie die Komplexität dieses Algorithmus. Definieren Sie zunächst Ihre Basisoperationen. Unterscheiden Sie in Ihren Betrachtungen in Abhängigkeit von der Eingabefolge $\{a_i\}$, $0 \leq i \leq n-1$, Sortierungen mit geringstem (best case) und mit größtem Aufwand (worst case).

(c) Verbessern Sie den Sortieralgorithmus, so dass eine „Gleichbehandlung“ von kleinsten und größten Elementen erfolgt und bereits sortierte Teilfolgen zur Aufwandssenkung genutzt werden.

2. Aufgabe: (AGS 6.6)

Gegeben sei die Folge: 8 , 5 , 4 , 7 , 3.

Wenden Sie auf diese Folge den Quicksort-Algorithmus an und dokumentieren Sie den Rechenablauf wie folgt:

- Pivotelement jeweils kennzeichnen
- Stellung der Indizes i, j unmittelbar vor dem Tausch von Elementen
- Stellung der Indizes i, j unmittelbar vor den rekursiven Aufrufen
- Teilfolgen nach den rekursiven Aufrufen

3. Aufgabe: (AGS 6.4)

Geben Sie eine Folge mit 7 selbst gewählten Zahlen an, die bei der Sortierung eine typische Situation entstehen lässt, wo der QuickSort-Algorithmus die worst-case Komplexität besitzt. Führen Sie die Sortierung mit entsprechender Protokollierung durch und geben Sie eine Abschätzung für die auszuführenden Operationen an.

4. Aufgabe: (AGS 3.27)

(a) Gegeben seien die folgende Typdefinition für Elemente eines Binärbaumes sowie eine Variable `Tree`:

```
typedef struct b_elem *TPtr;
typedef struct b_elem { int key;
                        TPtr left, right;
                        } elementtyp;
TPtr Tree;
```

Schreiben Sie in *C* eine Funktion `Tree_to_List`, die aus einem beliebigen Binärbaum `t` des o. g. Typs eine Liste der ungeraden Schlüsselwerte (`key`) von `t` generiert. Die Durchmusterung der Schlüsselwerte in `t` soll von links nach rechts erfolgen. Sie dürfen eine Funktion `void append(LPtr *l, int n)` verwenden, die ein neues Listenelement mit dem Schlüsselwert `n` erzeugt, an die Liste `*l` anhängt und den Zeiger auf den Nachfolger für dieses Listenelement NULL setzt. Geben Sie dazu alle erforderlichen Typ– und Variablendeklarationen sowie einen Aufruf Ihrer Funktion an.

(b) Eine einfach verkettete Liste sei aus Elementen aufgebaut, für die folgende Typdefinition gilt:

```
typedef struct ele *LPtr;
typedef struct ele { int zahl;
                     LPtr next;
                    } elementtyp;
LPtr Liste;
```

Geben Sie in *C* eine Funktion `reverse` an, die eine beliebige Liste dieses Typs „umdreht“, d. h. die Reihenfolge der Listenelemente umkehrt. Geben Sie einen Aufruf der Funktion an. Sie dürfen zur Lösung der Aufgabe eine Funktion `void append_element(LPtr *l, LPtr x)` verwenden, die ein Listenelement `x` an das Ende einer Liste `*l` anfügt und die Liste mit `x->next=NULL` abschließt.