Aufgabenblatt zur 9. Übung

Zeitraum: 13.12. bis 17.12.2010

Hinweis: Es wird sehr empfohlen, die bisher erworbenen C-Kenntnisse durch praktische Programmierung zu festigen. In den Kabinetten der Fakultät Informatik (gültiges LOGIN vorausgesetzt) finden Sie die technische Basis, um diese praktischen Schritte umzusetzen. Durch den Link „Praktische Einführung“ auf der Aufgabenseite „AuD“ gelangen Sie zu einer kurzen Einführung, die Ihnen diese ersten praktischen Schritte erleichtern soll.

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 6.14)

Wenden Sie auf die Folge: 10 , 3 , 1 , 5 , 6 , 8 , 19 , 4 , 17 , 20 , 2 , 7 den ShellSort-Algorithmus an. Wählen Sie hierbei für die Folge der Sortierabstände: 3 , 1. Dokumentieren Sie jeweils:
- Abstand h und Anzahl anz der Teilfolgenglieder,
- die Auswahl der zu sortierenden Teilfolge,
- die Wirkung der Funktion „InsertSort“ auf diese Teilfolge.

4. Aufgabe: (AGS 6.19)

Gegeben sei die Folge: 3, 5, 15, 8, 13, 18, 12, 7, 6, 17.

Wenden Sie auf diese Folge den HeapSort-Algorithmus an. Dokumentieren Sie dazu in der Phase 1 das schrittweise (knotenweise) Herstellen der heap-Eigenschaft; hier insbesondere die Vernderungen durch die Funktion „sinkenlassen“.
In der Phase 2 brauchen Sie nur <u>zwei</u> Sortierschritte auszufhren.
Dokumentieren Sie jeweils:
- das Abspalten des jeweils letzten Elementes (Blattes) im Wechsel mit der
- Wirkung der Funktion „sinkenlassen“.

Zusatzaufgabe 1: (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.

Zusatzaufgabe 2: (AGS 6.18*)

Gegeben sei die Folge: 1, 2, 3, 4, 5, 6, 7, 8.

Wenden Sie auf diese Folge den Heapsort-Algorithmus an. Dokumentieren Sie in der Phase 1:
- das Einordnen in den binren Baum,
- das schrittweise (knotenweise) Herstellen der Heap-Eigenschaft; hier insbesondere die Vernderungen durch die Funktion „sinkenlassen“.
In der Phase 2 brauchen Sie nur <u>drei</u> Sortierschritte auszufhren.
Dokumentieren Sie jeweils:
- das Abspalten des jeweils letzten Elementes (Blattes) im Wechsel mit der
- Wirkung der Funktion „sinkenlassen“.