BERGISCHE UNIVERSITÄT
GESAMTHOCHSCHULE WUPPERTAL
GAUSS-STRASSE 20
42097 WUPPERTAL
(Korrespondenzanschrift)
42119 WUPPERTAL
(Lieferanschrift)
TELEX 8 592 262 bughw
TELEFAX (0202) 439-2901
TELEFON (0202) 439-1

Fachbereich 7

MATHEMATIK

Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik / Numerik

e-mail: Juergen.Buhl@math.uni-wuppertal.de

Algorithmen und Datenstrukturen (Informatik II)

SS2001 – Übungsblatt 10

Abgabetermin: 9. Juli 2001

Aufgabe 1. *Sortieren von C-Vektoren, 4 Punkte*
Schreiben Sie eine Funktion

```
void sortiere(double v[], int n),
```

die die n Elemente $v_0, v_1, ..., v_{n\text{-}1}$ des Vektors v aufsteigend sortiert.

Benutzen Sie dabei folgenden Algorithmus (**Sortieren durch Auswählen**):

Suchen Sie im ersten Schritt das kleinste (ein kleinstes) Element und bringen Sie es an die Indexstelle 0, wobei der Wert der alten Stelle 0 an die Stelle gebracht wird, wo zuvor das gerade an die Indexstelle 0 gebrachte stand. Das erste Element ist jetzt $\leq$ als alle folgenden Elemente.

Benutzen Sie jetzt die selbe Idee, um das Restfeld $(v_1, v_2, ..., v_{n\text{-}1})$ analog dem sortierten Endzustand näher zu bringen. ...

Aufgabe 2. *Sortierverbesserung, 4 Punkte*
Verbessern Sie die Funktion `sortiere()` aus Aufgabe 1, indem Sie jetzt in jedem Schritt gleichzeitig ein größtes und ein kleinstes Element bestimmen und dann im zweiten Schritt "nur" noch das Restfeld $(v_1, v_2, ..., v_{n\text{-}2})$ zu bearbeiten haben. ...

Aufgabe 3. *Sortieren durch "Divide and Conquer", 4 Punkte*
Schreiben Sie eine Funktion

```
void sortiere2(double v[], int l, int h),
```

die die h–l+1 Elemente $v_l, v_{l+1}, ..., v_h$ des Vektors v aufsteigend sortiert. Benutzen Sie dabei folgenden Algorithmus (**Sortieren durch "Divide and Conquer"**):

Sortieren Sie in `sortiere2(double v[], int l, int h)` zuerst durch Selbstaufruf (Rekursion) mit `m = l + (h-l)/2` die Teilfelder $v_l, v_{l+1}, ..., v_m$ und $v_{m+1}, v_{m+2}, ..., v_h$ und mischen Sie diese dann sortierten beiden Teilfelder zunächst in einem (dynamisch erzeugten) Hilfsfeld zu einem insgesamt sortierten Feld zusammen, um dieses Hilfsfeld schließlich vor Funktionsreturn an die Positionen $l, l+1, ..., h$ des Parameters `v[]` zu kopieren. Brechen Sie die Rekursion bei der Länge 1 oder 2 des zu sortierenden Feldes durch spezielle Behandlung ab.

Aufgabe 4. *Index, 4 Punkte*
Wenn die Feldelemente viel Speicherplatz benötigen, ist es nicht sehr sinnvoll, diese Elementinhalte hin- und herzukopieren. Statt dessen arbeitet man dann mit einem Indexvektor.

Sei also wiederum $v_0, v_1, ..., v_{n\text{-}1}$ zu sortieren. Man läßt den Vektor v unverändert, führt einen Integer-wertigen Indexvektor $ind_0, ind_1, ..., ind_{n\text{-}1}$ ein, der zunächst die Werte $ind_i = i$ $(i = 0, 1, ..., n\text{-}1)$ zugewiesen bekommt. Anstatt die Werte der Komponenten von v umzukopieren bearbeitet man dann die Werte der Komponenten von *ind* so, dass am Ende gilt:

$$v_{ind_0} \leq v_{ind_1} \leq \ldots \leq v_{ind_{n\text{-}1}}$$

Schreiben Sie eine Funktion `sortiere3()` mit geeignetem Funktionskopf (geeigneter Signatur), die die Sortierung nach dem Algorithmus von Aufgabe 2 durchführt, aber einen Index benutzt.

Welche anderen Vorteile hat die Benutzung von Indices?

Aufgabe 5. *Binäre Suche, 4 Punkte*
Schreiben Sie eine Funktion

```
int suche(double v[], int l, int h, double d),
```

und mit deren Hilfe

```
int suche(double v[], int n, double d),
```

die im sortierten Vektor $(v_1, v_2, ..., v_{n\text{-}1})$ nach dem Vorkommen von `d` sucht und, existiert ein solches, den (ersten) Index als Funktionsergebnis liefert, an dessen Stelle `d` in `v` auftritt; andernfalls soll das Ergebnis -1 geliefert werden.

Arbeiten Sie bei der Suche bitte genauso, wie Sie in einem Telefonbuch suchen würden: In der Mitte des (noch) zu durchsuchenden Telefonbuch(teils) aufschlagen. Gesuchter Wert hier vorhanden? Wenn ja: unmittelbar vorher auch schon vorhanden ... Wenn nein: Das noch zu durchsuchende Resttelefonbuch (vordere oder hintere Hälfte des im letzten Schritt zu durchsuchenden Teils) rekursiv analog weiter durchsuchen bis der Eintrag gefunden wurde oder der zu durchsuchende Rest leer ist.