

Datenstrukturen und Algorithmen – Informatik I

1. Übung

Abgabe der Lösungen: 3. Mai 2004

Scheinbedingungen:

Für den Erwerb eines Übungsscheins in “Datenstrukturen und Algorithmen” müssen folgende Leistungen erbracht werden:

- 50% der Punkte aus den Übungsaufgaben je Semesterhälfte
- Vorrechnen einer Übungsaufgabe je Semesterhälfte
- Anwesenheit bei 80% der Kleingruppenübungstermine
- Alternative Methode zum Erwerb von Punkten: “Softwarewettbewerb”

Die Übungen sollen in Zweier- oder Dreiergruppen bearbeitet werden. Jede Gruppe gibt pro Übung eine gemeinsam erstellte Lösung ab. Die Ausarbeitungen müssen mit Namen und Matrikelnummern versehen werden. Die Abgabe erfolgt in den Kleingruppenübungen.

Termine:

Vorlesung	Di 14:00-15:30	AM
	Fr 14:00-15:30	AM
Frontalübung	Mi 08:15-09:45	Aula 1
Kleingruppenübungen	Mo, 08:15 - 09:45	
	Mo, 12:15 - 13:45	versch. Räume
	Mo, 15:45 - 17:15	

Webseite zur Vorlesung:

<http://www.informatik.rwth-aachen.de/I6/web/Teaching/Lectures/SS04/Datenstrukturen/index.html>

Newsgroups (zur Vorlesung):

rwth.informatik.dsal, rwth.informatik.grundstudium

Aufgabe 1:**2+1 Punkte**

Für ein bestimmtes Problem stehen vier unterschiedliche Algorithmen mit unterschiedlichen Laufzeitkomplexitäten zur Verfügung:

$$A: \log_2 n \quad B: n \quad C: n^3 \quad D: 2^n$$

Auf dem vorhandenen Rechner kann mit jedem dieser Algorithmen ein Problem der Größe $n = 2000$ innerhalb einer Stunde gelöst werden.

- (a) Nun soll ein neuer Rechner angeschafft werden, der in gleicher Zeit doppelt so viele Instruktionen abarbeitet. Was ist die maximale Problemgröße n die bewältigt werden kann, wenn weiterhin nur eine Stunde zur Verfügung steht?
- (b) Welchen Algorithmus würden Sie für Probleme mit $n = 300$ bzw. $n = 10\,000$ verwenden?

Aufgabe 2: Komplexität von Funktionen**4 Punkte**

Sortieren Sie folgende Funktionen nach ihrer Komplexitätsordnung. Teilen Sie Ihre Liste derart in Äquivalenzklassen auf, daß f und g genau dann in derselben Äquivalenzklasse sind, wenn $f \in \Theta(g)$. Begründen Sie Ihre Antwort.

Es ist $\text{ld } n := \log_2 n$.

$$n^3 \quad n^2 \quad n! \quad 2^{(5+\text{ld } n)} \quad n^n \quad n \quad n \cdot 2^n \quad \left(\frac{4}{3}\right)^n \quad 42 \quad 3^n \quad (\text{ld } n)^{\text{ld } n}$$

Aufgabe 3: Laufzeitanalyse**1+1+1 Punkte**

Betrachten Sie die folgenden drei (in Pseudocode angegebenen) Prozeduren und bestimmen Sie die Laufzeit als Funktion von n . Geben Sie zusätzlich mit der O -Notation jeweils die obere Schranke an. Begründen Sie Ihre Behauptung.

```
(a) PROCEDURE matrixmult(n: INTEGER) =
  VAR i, j, k: INTEGER;
  BEGIN
    FOR i := 1 TO n DO
      FOR j := 1 TO n DO
        C[i, j] := 0;
        FOR k := 1 TO n DO
          C[i, j] := C[i, j] + A[i, k] * B[k, j];
        END;
      END;
    END;
  END matrixmult;
```

```
(b) PROCEDURE veryodd(n: INTEGER) =
  VAR i, j, x, y: INTEGER;
  BEGIN
    FOR i := 1 TO n DO
      IF odd(i) THEN
        FOR j := i TO n DO
          x := x+1 ;
        END;
      ELSE
        FOR j := 1 TO i DO
          y := y+1 ;
        END;
      END;
    END;
  END veryodd;
```

```

(c) PROCEDURE mystery(n: INTEGER) =
  VAR i, j, k: INTEGER ;
  BEGIN
    FOR i := 1 TO n-1 DO
      FOR j := i+1 TO n DO
        FOR k := 1 TO j DO
          /* eine Anweisung mit Zeitaufwand O(1) */
        END ;
      END ;
    END ;
  END mystery ;

```

Aufgabe 4: Programmieraufgabe: Implementierung verketteter Listen 2+1+1+1 Punkte

Implementieren Sie den abstrakten Datentyp `List` von ganzen Zahlen als verkettete Liste. Vervollständigen Sie dazu die Methoden folgender Klasse:

```

class List {
  public List() {} //initialisieren der Liste
  public boolean empty() {} //testen, ob die Liste leer ist
  public boolean end() {} //testen, ob man am Listenende steht
  public void next() {} //zum nächsten Listenelement gehen
  public void rewind() {} //zum ersten Listenelement gehen
  public void last() {} //zum letzten Listenelement gehen
  public int getValue() {} //Wert des aktuellen Listenelements zurückgeben
  public void setValue(int val) {} //Wert des aktuellen Listenelements setzen
  public void insert(int val) {} //hinter dem aktuellen Listenelement einfügen
  public void append(int val) {} //ans Listenende anfügen
  public void remove() {} //aktuelles Listenelement löschen
  public void print() {} //ganze Liste ausgeben
}

```

- (a) Verwenden Sie die Methoden der Klasse `List`, um die Elemente zweier Listen unterschiedlicher Länge in eine Liste zusammenzuführen, ohne dabei eine dritte Liste anzulegen. Dabei sollen die Elemente aus beiden Listen jeweils abwechselnd eingefügt werden.
- (b) Erweitern Sie die Implementierung zu einer *doppelt* verketteten Liste, die zusätzlich folgende Methoden bietet:

```

  public void back() //zum vorhergehenden Listenelement gehen
  public boolean start() //testen, ob man am Listenanfang steht
  public void insertBefore(int val) //vor dem aktuellen Listenelement einfügen

```

- (c) Ändern Sie die Listenimplementierung so ab, daß Sie einen Stack mit den Operationen *Push* und *Pop* erhalten.
- (d) Ändern Sie die Listenimplementierung so ab, daß Sie eine Queue mit den Operationen *Put* und *Get* erhalten.