Institut für Computergraphik und Algorithmen

Abteilung für Algorithmen und Datenstrukturen

TECHNISCHE UNIVERSITÄT WIEN

Prof. Petra Mutzel
Gunnar W. Klau
Gabriele Kodydek
Günther Raidl
René Weiskircher

Wintersemester 2001/2002

Klausur zur Vorlesung
Algorithmen und Datenstrukturen 1
14. Dezember 2001

a) Machen Sie bitte die folgenden Angaben in deutlicher Blockschrift:

Vorname: _____ Nachname: _____

Matrikelnummer: _____ Studienkennzahl: _____

b) Klausur soll gewertet werden für (nur ein Kreuz!):

- ○ VO-Prüfung Algorithmen und Datenstrukturen 1 (*default*)
- ○ VO-Prüfung Rekursive Prozeduren und flexible Datenstrukturen

c) Legen Sie während der Klausur Ihren Studentenausweis vor sich auf das Pult.

d) Schreiben Sie die Lösungen direkt auf das jeweilige Aufgabenblatt. Wenn Ihnen das Papier ausgeht, bitten Sie die Aufsicht um Nachschub. Es ist nicht erlaubt, eigenes Papier zu verwenden!

e) Denken Sie daran, dass keinerlei Hilfsmittel erlaubt sind – weder Taschenrechner, irgendwelche Unterlagen, Handys,...

VOR DER ABGABE AUSZUFÜLLEN:

Geben Sie bitte die Anzahl der zusätzlich abgegebenen Blätter an: _____

Resultat:

Aufgabe	A 1	A 2	A 3	A 4	A 5		Note
maximale Punktzahl	11	9	10	10	10	50	———
erreichte Punktzahl							

Viel Erfolg!

Aufgabe 1: O/Θ/Ω-Notation **(11 Punkte)**

a) Seien $f(n)$ und $g(n)$ Funktionen mit positiven Wertebereichen. Sind die folgenden Aussagen wahr oder falsch?

Diese Aussage ist	Wahr	Falsch
$f(n) = \Omega(g(n)) \leftrightarrow f(n) = O(g(n))$.		
Aus $f(n) = \Theta(\log n)$ folgt $f(n) = O(\log n)$.		
Aus $f(n) = \Omega(n^2)$ folgt $f(n) = O(n)$.		
Aus $f(n) = \Omega(n^2)$ folgt $n^2 = \Theta(f(n))$.		
$f(n) - g(n) = \Omega(\min(f(n), g(n))$.		
$f(n) = \Theta(\sqrt{f(n^2)})$.		

b) Betrachten Sie die folgende Funktion:

$$f(n) = \begin{cases} 3n^2 & \text{falls } n \text{ durch drei teilbar ist} \\ 2n^3 & \text{falls } n \text{ nicht durch drei teilbar ist} \end{cases}$$

Gilt $f(n) = O(\frac{29}{10}n^3)$? Beweisen Sie Ihre Antwort.

Aufgabe 2: Balancierte Bäume **(9 Punkte)**

a) Konstruieren Sie den balancierten AVL–Baum, der sich aus dem leeren Baum ergibt, wenn man nacheinander die Elemente

$$6, 4, 3, 12, 8, 10, 11, 5$$

einfügt. Stellen Sie dabei den Baum nach **jedem** Einfügeschritt dar.

b) Löschen Sie nun das Element 8 und geben Sie die erreichte Konfiguration an.

Aufgabe 3: Mergesort **(10 Punkte)**

a) Geben Sie den Pseudocode von *Merge-Sort*(A, p, q) an. Halten Sie sich an folgende Bezeichnungen: Die N-elementige Folge ist in dem Array $A[1..N]$ gespeichert, und p und q geben den linken und rechten Index an. Nehmen Sie dabei an, dass die Prozedur Merge(A, p, q, m) bereits gegeben ist. (Es ist also nicht notwendig, den Pseudocode hierfür anzugeben.)

b) Führen Sie das Sortierverfahren *Merge-Sort* (passend zu Ihrem Pseudocode) mit der folgenden Folge durch:

10, 65, 32, 87, 88, 86, 33, 4, 44, 44

Geben Sie dabei für jeden Aufruf der Prozedur **Merge**(A, p, q, m) die Indexgrenzen p, q und m sowie die Folge A **vor** und **nach** dem Aufruf an.

Aufgabe 4: What am I? **(10 Punkte)**

Sei A ein Feld mit n Zahlen, wobei der erste Eintrag des Feldes $A[1]$ ist und der letzte Eintrag $A[n]$.

Algorithm 1 whatami(A)

```
wiederhole {
  v = 0;
  für  i = 1 bis n − 1  tue {
    falls A[i] > A[i + 1] dann {
      h = A[i];
      A[i] = A[i + 1];
      A[i + 1] = h;
      v = 1;
    }
  }
} bis  v ≡ 0 ;
```

a) Wie sieht das Feld A nach Aufruf des Algorithmus **whatami**(A) aus?

b) Wie ist die worst–case–Laufzeit des Algorithmus **whatami** in Abhängigkeit von der Länge n der Eingabefolge A? Wie sieht A in diesem Fall aus?

c) Wie ist die best–case–Laufzeit von **whatami** in Abhängigkeit von n? Wie sieht A nun aus?

d) Seien i und j zwei beliebige zulässige Indizes des Feldes A und seien $p = A[i]$ und $q = A[j]$ die zugehörigen Schlüssel. Jetzt wird der Algorithmus **whatami** aufgerufen; p steht nun an der Stelle k und q an der Stelle l.

Beweisen oder widerlegen Sie: Aus $i < j$ und $p \leq q$ folgt $k < l$.

Aufgabe 5: Durchmusterung **(10 Punkte)**

a) Gegeben sind Preorder– und Inorder–Reihenfolge der Elemente eines binären Baumes T:

Preorder–Reihenfolge: b, k, l, g, c, d, a

Inorder–Reihenfolge: k, b, d, c, g, a, l

(a) Konstruieren Sie den dazugehörigen binären Baum T.

(b) Wie lautet die Postorder-Reihenfolge von T?

b) Geben Sie den Pseudocode eines Algorithmus **LevelDurchmusterung** an, der einen binären Baum in Level-Reihenfolge durchläuft.

Die Level-Reihenfolge eines binären Baums T ist gegeben durch: Durchsuche zuerst alle Knoten der Stufe 1 (die Wurzel des Baumes), dann alle Knoten der Stufe 2 (Kinder der Wurzel) von links nach rechts, danach alle Knoten der Stufe 3 (Kinder der Kinder der Wurzel) von links nach rechts, usw.

Verwenden Sie hierzu die verallgemeinerte Listenstruktur, die über `p.key`, `p.left`, `p.right` ansprechbar ist.

Achten Sie darauf, dass der von Ihnen verwendete Pseudocode eindeutig ist. Verwenden Sie aussagekräftige Bezeichner und versehen Sie Ihren Pseudocode mit Kommentaren.