Institut für Computergraphik und Algorithmen

Abteilung für Algorithmen und Datenstrukturen

TECHNISCHE UNIVERSITÄT WIEN

Prof. Petra Mutzel
Gunnar Klau
Ivana Ljubic
René Weiskircher

Wintersemester 2000/2001

Klausur zur Vorlesung
Algorithmen und Datenstrukturen 1
29. Jänner 2001

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

Name: _____ Vorname: _____

Matrikelnummer: _____ Studienkennzahl: _____

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

c.) 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!

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

VOR DER ABGABE AUSZUFÜLLEN:

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

f.) Kreuzen Sie bitte die von Ihnen bearbeiteten Aufgaben in der ersten Zeile der Tabelle an:

Aufgabe	A 1	A 2	A 3	A 4	A 5	A 6		Note
bearbeitet							___	_____
maximale Punktzahl	10	13	13	14	10	10	70	_____
erreichte Punktzahl								

Viel Erfolg!

Aufgabe 1: Laufzeitabschätzungen **(10 Punkte)**

a.) Welche Laufzeit in Θ-Notation für n haben folgende Programme?

A
```
(0) Für i=1,...,n {
(1)   j = i + n*n;
(2)   printf("j");
(3) }
```

B
```
(0) Für i=1,...,n {
(1)   printf("i");
(2)   Für j=1,...,n {
(3)     sum = sum + a[i,j];
(4)   }
(5) }
```

C
```
(0) sum = 0;
(1) Für i=1,...,n {
(2)   sum = sum + i;
(3) }
(4) Für i=1,...,sum {
(5)   printf("i");
(6) }
```

D
```
(0) n=2^k;   /* (2 hoch k) */
(1) Für i=1,...,k {
(2)   sum = sum + 2^k + n;
(3) }
```

Dabei zählen wir die Dauer der Berechnung von 2^k als 1.

E Aufruf von `NONAME(1,n)`:

```
(0) Algorithmus NONAME(p,q);
(1) Falls p<q {
(2)   l=floor((p+q)/2); /* abgerundet */
(3)   NONAME(p,l);
(4)   NONAME(l+1,q);
(5)   Für i=p,...,q {
(6)     printf("i");
(4)   }
(5) }
```

Aufgabe 2: Sortierverfahren **(13 Punkte)**

a.) Ein Sortieralgorithmus heißt stabil, wenn er so implementiert werden kann, dass die Reihenfolge von Elementen mit gleichem Schlüssel während des Sortierverfahrens nicht vertauscht wird. Welche der folgenden Sortierverfahren sind stabil?

Sortierverfahren	stabil	nicht stabil
Insertion Sort		
Selection Sort		
Merge Sort		
Quick Sort		
Heap Sort		

Bewertung: Für jedes richtige Kreuz gibt es einen Pluspunkt, für jedes falsche Kreuz einen Punkt Abzug, für fehlende Kreuze keine Punkte. Es können insgesamt nicht weniger als 0 Punkte erreicht werden. D.h. kreuzen Sie am Besten nur die Antworten an, bei denen Sie sich sicher sind.

b.) Sortieren Sie die Folge

$$314, 263, 329, 298, 193, 223, 253, 110$$

mit dem Sortierverfahren `Selection Sort`. Stellen Sie die Teilfolge nach jeder Iteration dar. Welche Datenstruktur ziehen Sie bei diesem Sortierverfahren vor: Listen oder Felder? Begründen Sie Ihre Antwort.

Aufgabe 3: Binäre Suchbäume **(13 Punkte)**

a.) Gegeben ist ein binärer Suchbaum mit Schlüsseln im Bereich von $[1, \ldots, 1000]$. Wir suchen nach dem Schlüssel mit der Nummer 363. Welche der folgenden Suchsequenzen können nicht durch die Suche in einem korrekten binären Suchbaum entstanden sein?

(a) $2, 252, 401, 398, 330, 344, 397, 363$

(b) $924, 220, 911, 244, 898, 258, 362, 363$

(c) $925, 202, 911, 240, 912, 245, 363$

(d) $2, 399, 387, 219, 266, 392, 381, 278, 363$

(e) $935, 278, 347, 621, 299, 392, 358, 363$

Begründen Sie Ihre Antwort.

b.) Gegeben ist ein binärer Suchbaum. Entwerfen Sie einen nichtrekursiven Algorithmus (Pseudocode), der einen binären Suchbaum mit Hilfe der unten angegebenen Prozeduren `Minimum` und `Successor` in `Inorder`-Reihenfolge durchläuft und jeweils den Schlüssel des besuchten Knotens ausgibt.

```
(0) Algorithmus Minimum(p)
(1)     Solange (p.leftson! = NULL)
(2)         p = p.leftson
(3)     Return p;
```

```
(0) Algorithmus Successor(p)
(1)     Falls (p.rightson! = NULL)
(2)         Return Minimum(p.rightson)
(3)     q = p.father;
(4)     Solange ((q! = NULL) und (p==q.rightson)) {
(5)         p=q; q=q.father; }
(6)     Return q;
```

Aufgabe 4: Was macht der folgende Algorithmus? **(14 Punkte)**

Gegeben sei ein Array $A[1,...,n]$ mit n ganzen Zahlen und der Eigenschaft $A[i] \leq A[i+1]$ für alle $i = 1,\ldots,n-1$. Weiterhin seien die ganzzahligen Variablen x und y global definiert und mit Anfangswert 0 belegt.

```
(0) Algorithmus WhatAmI (int A[1..n], int i)
(1) /* Input: Sortierte Folge von n Zahlen A[1,...,n] */
(2) /* Output: ??? */
(3)     Falls i == 1 {
(4)         x = 1; y = A[1]; }
(5)     Sonst {
(6)         WhatAmI (A,i-1);
(7)         Falls (A[i] == A[i - x]) {
(8)             x = x + 1; y = A[i]; }
(9)     }
```

a.) Was können Sie nach Ablauf des Algorithmus mit Aufruf WhatAmI (A,n) über die Werte der globalen Variablen x und y sagen?

b.) Was können Sie nach Ablauf des Algorithmus mit Aufruf WhatAmI (A,n) über die Werte der globalen Variablen x und y sagen, wenn das Feld A lauter gleiche Zahlen enthält?

c.) Erklären Sie in eigenen Worten genau wie der Algorithmus funktioniert.

d.) Geben Sie die Laufzeit des Algorithmus für den Worst-Case und den Best-Case in Θ-Notation an.

Aufgabe 5: Queue mit einfach verketteter Liste **(10 Punkte)**

Es geht darum, eine Queue mittels einer einfach verketteten Liste in JAVA zu implementieren. Die Konstruktoren sowie die Operationen `size` und `isEmpty` haben wir schon für Sie implementiert. Den Code für die Klasse `Node` und die schon implementierten Funktionen der Klasse `queue` finden Sie am Ende der Aufgabe.

a.) Geben Sie die Implementierung der beiden Funktionen `enqueue` und `dequeue` in JAVA an. Ihre Funktionen müssen zu den von uns zur Verfügung gestellten Code-Stücken passen. Die Funktion `enqueue` hängt ein neues Element an das Ende (Tail) der Queue an, während die Funktion `dequeue` das Element am Anfang der Queue (Head) löscht und zurückliefert. Ihre Funktionen müssen die in unserem Code angegebenen Rückgabewerte und Argumente haben.

b.) Begründen Sie, warum es besser ist, am Kopf (Head) der Liste Elemente zu entfernen und am Ende (Tail) der Liste einzufügen als am Ende zu löschen und am Anfang einzufügen.

```
class Node {
  private Object element;
  private Node next;
  Node() {
    this(null,null);
  }
  public Node(Object e, Node n) {
    element = e;
    next = n;
  }
  void setElement(Object newElem) {element = newElem;}
  void setNext(Node newNext) {next = newNext;}
  Object getElement() { return element;}
  Node getNext() {return next;}
}

public class queue {
  private Node head;
  private Node tail;
  private int size;
  public queue() {
    head = null;
    tail = null;
    size = 0;
  }
  public int size() {
    return size;
  }
  public boolean isEmpty() {
    return (head == null);
  }
  public void enqueue(Object obj) {
          . . .
  }
  public Object dequeue() {
          . . .
  }
}
```

Aufgabe 6: Hashing **(10 Punkte)**

a.) Fügen Sie die folgenden Zahlen in eine Hashtabelle der Größe 15 ein.

10,8,23,20,3,5,12,17,19,9

Verwenden Sie zur Kollisionsbehandlung Double Hashing mit dem Algorithmus von Brent mit den beiden Hashfunktionen $h_1(k)$ und $h_2(k)$:

$$h_1(k) = k \bmod 12 \quad (1)$$
$$h_2(k) = k \bmod 15 \quad (2)$$

Stellen Sie die Hashtabelle nach jedem Einfügen dar.

b.) Sind $h_1(k)$ und $h_2(k)$ geeignete Hashfunktionen? Begründen Sie Ihre Antwort.

c.) Geben Sie zwei alternative Hashfunktionen für Double Hashing mit Hilfe von Brent an, die für eine Hashtabelle der Größe 15 gut geeignet sind.