Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 1**				VO-Prüfung 25. Juni 1999 Gruppe A	
Kennz.†	**Matrikelnr.**	**Beil.**‡	1 (25)	2 (20)	3 (25)	4 (30)
NACHNAME, Vorname						Σ (100)

† Geben Sie jene Kennzahl an, auf die das Zeugnis ausgestellt werden soll.

‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**)

Kreuzen sie bitte an welche Prüfung und/oder welchen Test sie absolvieren wollen:

Vorlesungsprüfung Rekursive Prozeduren	
Vorlesungsprüfung Algorithmen und Datenstrukturen 1	
3. Übungstest Algorithmen und Datenstrukturen 1	

(20) Aufgabe 1 – Quadtree

Die Bilder einer Digitalkamera sind als Quadtrees abgespeichert.

```
TYPE
  Grauwert = 0...255;
  Quadtree = ^Knoten;
  Knoten = RECORD
             blatt: BOOLEAN;
             info: Grauwert;
             next: ARRAY[1..4] OF Quadtree   { laut Abb. 4.14 }
           END;
```

Die Bilder werden von der Digitalkamera nicht entsprechend der Kameraausrichtung gespeichert. Um dieses Problem zu beheben, sollen Sie nun eine *effiziente* und *rekursive* Funktion

```
rotierteKopie(bild : Quadtree; rotiere : INTEGER) : Quadtree;
```

schreiben, die eine rotierte Kopie des übergebenen Bildes zurückliefert. Der Parameter `rotiere` soll dabei angeben, um wieviel die Kopie rotiert sein soll:

```
rotiere = 0: keine Rotation
rotiere = 1: 90 Grad im Uhrzeigersinn
rotiere = 2: 180 Grad
rotiere = 3: 270 Grad im Uhrzeigersinn
```

Zerstören Sie dabei auf keinen Fall die originalen Bildaten!

(20) Aufgabe 2 – Radix Exchange Sort

a) Sortieren Sie die in der Tabelle gegebene Zahlenfolge *absteigend* mit **Radix Exchange Sort**. Verwenden Sie den Algorithmus so, wie er im Skriptum (Listing 7.12) steht. Achten Sie darauf, daß **in jeder Zeile der Tabelle nur Teilfolgen derselben Rekursionstiefe** stehen.

Kennzeichnen Sie deutlich die Grenzen der Teilfolgen!

Als Hilfe enthält die erste Zeile der Tabelle die Indices der Elemente.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
39	**37**	**18**	**25**	**29**	**35**	**22**	**21**	**28**	**30**	**41**	**13**	**27**	**32**	**19**

Hilfe (Bittabelle):

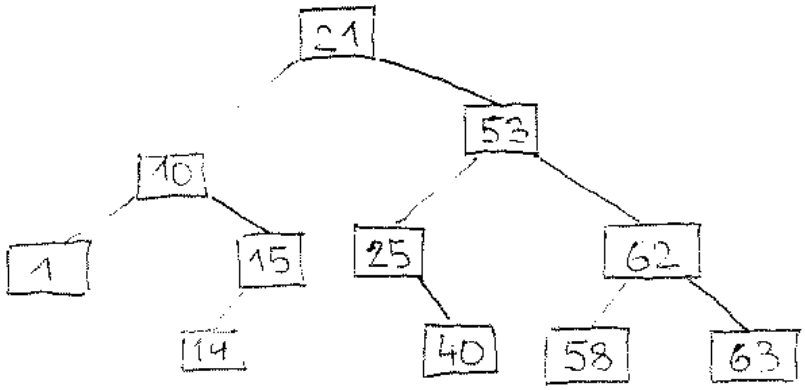
0	000000	16	010000	32	100000	48	110000
1	000001	17	010001	33	100001	49	110001
2	000010	18	010010	34	100010	50	110010
3	000011	19	010011	35	100011	51	110011
4	000100	20	010100	36	100100	52	110100
5	000101	21	010101	37	100101	53	110101
6	000110	22	010110	38	100110	54	110110
7	000111	23	010111	39	100111	55	110111
8	001000	24	011000	40	101000	56	111000
9	001001	25	011001	41	101001	57	111001
10	001010	26	011010	42	101010	58	111010
11	001011	27	011011	43	101011	59	111011
12	001100	28	011100	44	101100	60	111100
13	001101	29	011101	45	101101	61	111101
14	001110	30	011110	46	101110	62	111110
15	001111	31	011111	47	101111	63	111111

b) Welche der folgenden Sortieralgorithmen sind dazu geeignet, einfach verkettete Listen effizient zu sortieren (ohne Hilfsarrays)? *Begründen Sie Ihre Antwort!*

Algorithmus	**geeignet?**
Quicksort	
Heapsort	
Mergesort	

(25) Aufgabe 3 – Suchbaum

(10) **a)** Entfernen Sie aus dem folgenden binären Suchbaum das Element mit dem Schlüssel **53**, dann das Element mit dem Schlüssel **21**. **Holen Sie die Ersatzknoten immer aus dem rechten Teilbaum!** Zeichen Sie den Baum nach dem Löschen von jedem der beiden Schlüsseln (also zwei mal)!



(15) **b)** Gegeben ist die folgende Definitionen für einen binären Suchbaum:

```
TYPE
  KnotenPtr = ^Knoten;
  Baum = OBJECT
           pKnoten : KnotenPtr;
           FUNCTION istNil : BOOLEAN;       { TRUE, wenn pKnoten = NIL }
           FUNCTION getInfo : INTEGER;      { gibt pKnoten^.info }
           PROCEDURE drucke;                { vollstaendige Ausgabe }
           PROCEDURE druckeBis(bis : INTEGER);
         END
  Knoten = RECORD
             info : INTEGER;
             links : Baum;   { kleinere Elemente }
             rechts : Baum;  { groessere Elemente }
           END;
```

Schreiben Sie eine *effiziente*, *rekursive* Methode

```
Baum.druckeBis(bis : INTEGER);
```

die alle Knoten des Baumes, die kleiner gleich dem angegebenen Wert sind (`info <= bis`), in *aufsteigender* Reihenfolge mittels `writeln` ausgibt. Sie dürfen dazu die vorhandene Methode `Baum.drucke` verwenden, die *alle* Knoten eines Baumes in *aufsteigender* Reihenfolge ausgibt.

Bemerkung: Effizient heißt, daß keine Teilbäume ausgegeben werden, die sicher keinen Schlüssel enthalten der größer als `bis` ist. Weiters soll `Baum.drucke` verwendet werden, sobald klar ist, daß *alle* Knoten eines Teilbaumes ausgegeben werden müssen.

(30) **Aufgabe 4 – Rangierbahnhof**

Die Waggons, die in einem Rangierbahnhof eintreffen, sollen sofort in einer Datenbank erfaßt werden. Jeder Waggon hat eine eindeutige Identifikationsnummer (max. 20 Zeichen), einen Zielbahnhof (max. 30 Zeichen), und ein Gesamtgewicht (in Kilogramm).

Die Waggons werden entsprechend Ihres Zielbahnhofs weitergeleitet, daher soll es zu jedem Zeitpunkt möglich sein, alle Waggons in der Datenbank, die den selben Zielbahnhof haben, auszugeben. Desweiteren soll es möglich sein, zu jedem Zeitpunkt abzufragen ob ein Waggon mit einer bestimmten Identifikationsnummer am Rangierbahnhof ist. Sobald eine Waggon den Rangierbahnhof verläßt, wird er wieder aus der Datenbank entfernt.

(10) **a)** Entwerfen Sie eine kombinierte Datenstruktur, die alle geforderten Operationen (Eintragen, Abfrage nach Zielbahnhof, Abfrage nach Identifikationsnummer, Entfernen) *effizient* unterstützt. Welche Datenstrukturen sind notwending? Geben sie sowohl die Typdefinition als auch die globalen Variablen an, die dazu nötig sind.

(10) **b)** Schreiben Sie die Prozedur

```
PROCEDURE einfuegen(id : STRING20; ziel : STRING30;
                    gewicht : INTEGER);
```

die einen neu angekommenen Waggon in Ihre Datenbank einfügt.

(10) **c)** Schreiben Sie nun ebenfalls die Prozedur

```
PROCEDURE waggonsNach(ziel : STRING30);
```

die alle Waggons ausgibt, die zum angegebenen Zielbahnhof weitergeleitet werden sollen.

Um Identifikationsnummern und Zielbahnhöfe zu vergleichen, können sie einfach die normalen Vergleichsoperatoren auf Variablen der Typen `STRING20` und `STRING30` anwenden.