Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 1**				VO-Prüfung 20. März 1998 Gruppe A	
Kennz.†	**Matrikelnr.**	**Beil.**‡	1 (20)	2 (25)	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**!).

(20) Aufgabe 1 – Heap

(4) **a)** Ist das folgende Array ein Heap? Wenn nicht, dann markieren Sie das erste gefundene Zahlenpaar, das die Heapbedingung verletzt.

101	84	71	56	80	63	65	58	14			

o Heap
o kein Heap

(8) **b)** Tragen Sie in den folgenden Heap ein neues Element **25** ein! Verwenden Sie dazu den im Skriptum beschriebenen Algorithmus.

14	11	6	10	8	5	1	7	9			

anz = 9

Ergebnisheap:

anz = ___

(8) **c)** Entfernen Sie das grösste Element aus dem folgenden Heap! Verwenden Sie dazu den im Skriptum beschriebenen Algorithmus.

14	11	6	10	8	5	1	7	9			

anz = 9

Ergebnisheap:

anz = ___

(25) Aufgabe 2 – Mergesort

Gegeben ist folgende Definition einer Datenstruktur für Listen:

```
TYPE
    NodePtr = ^Node;
    Node = RECORD
             number: INTEGER;
             next:   NodePtr;
           END;
```

Schreiben Sie eine *effiziente, rekursive* Funktion

```
PROCEDURE mergesort(VAR list: NodePtr);
```

die die übergebene Liste mittels Mergesort sortiert. Verwendenen Sie dazu *keine Arrays*, arbeiten Sie direkt mit Listen.

Hinweis: Um die übergebene Liste in zwei gleich große Teillisten aufzuteilen, können Sie einfach jedes zweite Element in die eine, die anderen Elemente in die andere Teilliste geben.

(25) Aufgabe 3 – Rechts-balancierter Baum

Ein *aufsteigend* sortiertes Array `table`

```
VAR table: ARRAY [1..MAX] OF INTEGER;
```

soll in einen rechts-balancierten binären Baum umgespeichert werden. Ein binärer Baum ist rechts-balanciert, wenn für jeden Zwischenknoten gilt, daß entweder beide Teilbäume unter ihm gleich viele Knoten enthalten, oder daß der rechte Teilbaum genau einen Knoten mehr als der linke enthält.

Die Datenstruktur für den Baum ist gegeben durch:

```
TYPE Tree = ^Node;
     Node = RECORD
              value: INTEGER;
              left, right: Tree;
            END;
```

Schreiben Sie dazu eine rekursive Funktion:

```
FUNCTION righttree(lowIndex, hiIndex: INTEGER): Tree;
```

die einen rechts-balancierten Baum vom global zugreifbaren Array `table` erzeugt. Geben Sie auch den Aufruf dieser Prozedur vom Hauptprogramm an.

Hinweis: Verwenden Sie die Sortierung des Arrays `table`!

(30) Aufgabe 4 – Objektorientierte Mengen

In objektorientiertem Turbo Pascal sollen Mengen von Zahlen einerseits mittels Bäumen andererseits mittels Hashtables (mit äußerer Verkettung) von `INTEGER`s verwirklicht werden.

Die Basisklasse `Set` gibt das Klassen-Interface vor: Mit der Methode

```
numElements: INTEGER;
```

soll erfragt werden, wieviele Elemente in der Menge enthalten sind, mit der Methode

```
insert(value: INTEGER);
```

soll ein Wert in die Menge aufgenommen werden können, und mit der Methode

```
search(value :INTEGER) : BOOLEAN;
```

soll getestet werden können, ob ein Element in der Menge enthalten ist.

Die Subklasse `HashSet` verwaltet die Menge mittels einer Hashtabelle. Sie können davon ausgehen, daß maximal `CONST maxElements = ...;` Elemente in der Menge Platz finden müssen.

Die Subklasse `TreeSet` verwaltet die Menge als binären Suchbaum von Elementen.

Entwerfen und implementieren Sie die Klassen `Set, HashSet` und `TreeSet`, und deren Methoden `numElements` und `search`. Deklarieren sie `insert` in den jeweiligen Klassendefinitionen. Konstruktoren und Destruktoren müssen Sie nicht berücksichtigen. Auf Fehlerbehandlung können sie ebenfalls verzichten.