Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 1**			2. Übungstest 29. Mai 1998 Gruppe A	
Matrikelnr.	**Beil.†**	1 (13)	2 (12)	3 (13)	4 (12)
NACHNAME, Vorname					Σ (50)

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

(13) Aufgabe 1 – Heap

(3) **a)** Ist das folgende Array ein *umgekehrter* Heap (mit dem kleinsten Element an der ersten Stelle)? Wenn nicht, dann markieren Sie das erste gefundene Zahlenpaar, das die Heapbedingung verletzt.

10	11	20	13	15	22	30	14	12	31	32	

∘ Heap
∘ kein Heap

(5) **b)** Tragen Sie in den folgenden *umgekehrten* Heap ein neues Element **15** ein! Verwenden Sie dazu den im Skriptum beschriebenen Algorithmus.

26	29	34	30	32	35	39	33	31			

anz = 9

Ergebnisheap:

anz = ___

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

26	29	34	30	32	35	39	33	31			

anz = 9

Ergebnisheap:

anz = ___

(12) Aufgabe 2 – Stringsuche

Gegeben ist folgender Text:

```
ALLE ARTEN HABEN IM HARTEN GARTEN LANGE ZU WARTEN
```

und das Suchwort:

```
WARTEN
```

Suchen Sie dieses Suchwort im gegebenen Text mit der korrekten Version des Mismatched-Character Algorithmus (Kapitel 5.6, Stringsuche. **Achtung:** nicht den Sunday-Algorithmus verwenden). Wie oft werden dabei einzelne Buchstaben verglichen (Buchstabenvergleiche) und wie oft müssen Sie das Suchwort verschieben, bis Sie das Suchwort im Text gefunden haben (Verschiebungen)?

Geben Sie die Shift-Tabelle und alle Zwischenschritte an! Untersctreichen Sie die miteinander verglichenen Buchstaben! **Achtung:** Beachten Sie bitte, daß die Leerzeichen (im Text unten als "_" symbolisiert) auch wie Buchstaben behandelt werden!

W	A	R	T	E	N	sontige

ALLE_ARTEN_HABEN_IM_HARTEN_GARTEN_LANGE_ZU_WARTEN

Buchstabenvergleiche = ____
Verschiebungen = ___

(13) Aufgabe 3 – Quicksort

Gegeben sei eine Datenstruktur für sortierte Listen von ganzen Zahlen.

```
TYPE    NodePtr = ^Node;
        Node =  RECORD
                    value : INTEGER;
                    next  : NodePtr;
                END;
        List =  RECORD
                    first : NodePtr;
                    last  : NodePtr;
                END;
```

a) Schreiben Sie eine *effiziente* Prozedur:

```
PROCEDURE quicksort(VAR numlist : List);
```

die die übergebene, unsortierte Liste mit dem Quicksort-Algorithmus aufsteigend sortiert. Verwenden Sie dabei jeweils das erste Element der Liste als Vergleichselement.

(12) Aufgabe 4 – Graphen

Gegeben Sei folgende Datenstrukturen zur Speicherung von gewichteten Graphen als Adjazenzliste:

```
TYPE    NodePtr = ^Node;
        Node =  RECORD
                    number : INTEGER;
                    weight : INTEGER;
                    next   : NodePtr;
                END;
        GraphList = ARRAY [1..K] OF NodePtr;
```

Schreiben sie eine *effiziente* Prozedur:

```
FUNCTION longestMinWeg(i : INTEGER; list : GraphList) : INTEGER;
```

die eine modifizierte Version des `MinWeg` Algorithmus aus dem Skriptum dazu verwendet, um die Länge des jeweils minimalen Weges zu allen Knoten des Graphen zu finden, und die Länge des längsten dieser `K` Wege zurückliefert. *Geben Sie alle zusätzliche Datenstrukturen an, die sie benötigen, und geben Sie an wie diese initialisiert werden müssen!*

Hinweis: Nehmen Sie an das die Prozedur `PGet` aus dem Skriptum die Knotennummer `-1` zurückliefert, wenn die Priorityqueue leer ist!