Prof. Andreas Goerdt
M. Pippig, M. Riedel, U. Schönke

TU Chemnitz
Wintersemester 2003/2004

Algorithmen und Programmierung

8. Übung

1. Aufgabe:

   Realisieren Sie den Algorithmus der binären Suche für die Suche eines Elementes `x` in einem aufsteigend sortierten Feld der Größe *n* in Java. Testen Sie Ihr Programm für sehr große *n* und untersuchen Sie die Laufzeit im Vergleich zum einfachen sequentiellen Durchsuchen.

   Hinweis zur Erzeugung des Feldes: Lassen Sie sich die Feldelemente zufällig erzeugen (z. B. wie in den Programmen „Partition2“ und „Find2“ aus der Vorlesung) und sortieren Sie das Feld mit `Arrays.sort( <Feldname> )`. Dazu muss am Anfang folgende Anweisung stehen: `import java.util.*`.

2. Aufgabe:

   Gegeben sei folgender Algorithmus für das Partitionierungsproblem:

```
x = feld[k];
i = -1;
j = feld.length;

while (i < j ) {

  do {
    j = j - 1;
  } while (feld[j] > x);

  do {
    i = i + 1;
  } while (feld[i] < x);

  if (i <j)
    >> vertausche feld[i] und feld[j] <<
  else
    >> gib j aus <<
}
```

Zeigen Sie anhand der beiden nachstehenden Invarianten die Korrektheit des Algorithmus.

Inv1:

Falls $i_l < j_l$:
Es gibt $i' \geq i_l$, so dass $f_l[i'] \geq x$.
Es gibt $j' \leq j_l$, so dass $f_l[j'] \leq x$.

Inv2:

Falls $i_l \leq j_l$:
$f_l[0], \ldots, f_l[i_l] \leq x$
$f_l[j_l], \ldots, f_l[\texttt{feld.length} - 1] \geq x$

3. Aufgabe:

Zeigen Sie die Invariante $i_l - j_l \leq 2$ des Programms „Partition“ aus der Vorlesung.

4. Aufgabe:

Gegeben sind zwei natürliche Zahlen *a* und *b* sowie folgender Algorithmus im Pseudocode:

```
x := a;
y := b;

u := a;
v := b;

solange x <> y
beginn
  wenn x < y dann
    beginn
      y := y - x;
      v := v + u;
    ende;
  sonst
    beginn
      x := x - y;
      u := u + v;
    ende;
ende;
ergebnis1 := x;
ergebnis2 := (u + v) div 2;
```

Setzen Sie den Algorithmus in Java um und finden Sie heraus, was eigentlich berechnet wird.

5. Aufgabe:

   Gegeben ist ein bereits initialisiertes Schachbrett. Dieses wird durch eine N×N-Matrix repräsentiert, die als Elemente entweder eine `0` (kein Turm steht dort) oder eine `1` (dort steht ein Turm) hat. Türme können beim Schach nur waagerecht oder senkrecht ziehen und schlagen. Im vorliegenden Fall haben zur Vereinfachung alle Türme die gleiche Farbe und dürfen sich gegenseitig schlagen.

   Formulieren Sie einen Algorithmus, der überprüft, ob sich genau N Türme auf dem Schachbrett befinden **und** eine Konstellation vorliegt, dass sich diese N Türme nicht gegenseitig schlagen können. Setzen Sie diesen Algorithmus in Java um.