Prof. Dr. Gerd Stumme, Folke Eisterlehner, Dominik Benz
Fachgebiet Wissensverarbeitung

5./7.05.2009

3. Präsenzübung „Algorithmen und Datenstrukturen“

Sommersemester 2009

Zufallssuche: Betrachten Sie den folgenden in Pseudocode formulierten Algorithmus zur Suche eines Schlüssels k in einem sortierten Feld F und geben Sie *Worst-Case-*, *Average-Case-* und *Best-Case-Laufzeit* in der O-Notation an.

```
algorithm RandSearch (F, k) → p
Eingabe: Folge F der Länge n die Schlüssel k enthält

   r :=random(n)
   while F[r] ≠ k do
      r :=random(n)
   od
   return r
```

Dabei liefert die Funktion **random(*n*)** in einem Schritt eine gleichverteilte ganze Zufallszahl zwischen 0 und $n-1$.

Sieb des Eratosthenes: Betrachten Sie folgende Java-Methode zur Bestimmung aller Primzahlen kleiner oder gleich einer gegebenen natürlichen Zahl n:

```
public static int[] primes(int n) {
   Integer[] sieve = new Integer[n];

   // Feld initialisieren: sieve[i] = i+2
   for( int i=0; i<n-1; i++ ) {
       sieve[i]=i+2;
   }
   // Zaehler fuer 'ausgesiebte' Zahlen
   int nrRemoved = 0;

   // Sieb des Eratosthenes
   int i = 2;
   while( i*i < n ) {
       if( sieve[i-2]!=null ) {
           for( int j=i*i; j<=n; j+=i ) {
              if( sieve[j-2]!=null )
                 nrRemoved++;
              // markiere Vielfaches von i
```

```
                sieve[j-2]=null;
             }
          }
          i++;
      }
      // entferne alle markierten Elemente
      int[] primes = new int[n-nrRemoved-1];
      int j=0;
      for( i=0; i<n; i++ ) {
          if( sieve[i]!=null ) {
             primes[j] = sieve[i];
             j++;
          }
      }

      return primes;
  }
```

Bestimmen Sie die Laufzeit dieser Methode

a) bezogen auf die *Zahl n* und

b) bezogen auf die *Länge* der Zahl *n* (d.h. die Länge der Binärdarstellung von *n*).

Boolesche Matrixmultiplikation: Folgender Code berechnet eine Art Multiplikation zweier Boolescher $n \times n$-Matrizen; anstatt der üblichen Bildung des Skalarprodukts zwischen den Zeilen- und Spaltenvektoren wird hierbei eine logische UND-Verknüpfung vorgenommen:

```
static void boolMult(boolean[][] A, boolean[][] B, int n) {
  // init result matrix (assume it is initialized
  // with all entries set to true)
  boolean[][] C = new boolean[n][n];

  // calculate "product"
  for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) {
      if (A[i][j] == false || B[j][i] == false) {
        C[i][j] = false;
      }
      else {
        for (int k=0; k<n; k++) {
          C[i][j] = C[i][j] && A[i][k] && B[k][j];
          if (C[i][j] == false) {
            break;
```

```
            }
          }
        }
      }
    }
}
```

Nehmen Sie hierbei an, dass bei Initialisierung der Ergebnis-Matrix C alle Einträge auf `true` gesetzt sind. Analysieren Sie den Algorithmus hinsichtlich der *Worst-case-Laufzeit*. Wie müsste eine Eingabe aussehen, damit der Algorithmus möglichst schnell terminiert?