**Script** generated by TTT

Title: Seidl: Informatik_1 (05.11.2012)

Date: Mon Nov 05 18:51:59 CET 2012

Duration: 51:43 min

Pages: 24

## 6 Eine erste Anwendung: Sortieren

Gegeben: eine Folge von ganzen Zahlen.

Gesucht: die zugehörige aufsteigend sortierte Folge.



## 6 Eine erste Anwendung: Sortieren

Gegeben: eine Folge von ganzen Zahlen.

Gesucht: die zugehörige aufsteigend sortierte Folge.

Idee:

- speichere die Folge in einem Feld ab;
- lege ein weiteres Feld an;
- füge der Reihe nach jedes Element des ersten Felds an der richtigen Stelle in das zweite Feld ein!

$\Longrightarrow$ Sortieren durch Einfügen ...



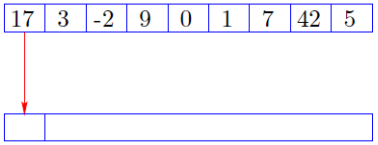
```
public static int[] sort (int[] a) {
   int n = a.length;
   int[] b = new int[n];
   for (int i = 0; i < n; ++i)
      insert (b, a[i], i);
         // b    = Feld, in das eingefügt wird
         // a[i] = einzufügendes Element
         // i    = Anzahl von Elementen in b
   return b;
   } // end of sort ()
```

Teilproblem: Wie fügt man ein ???

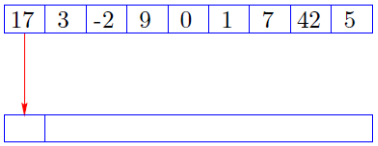
```
public static int[] sort (int[] a) {
   int n = a.length;
   int[] b = new int[n];
   for (int i = 0; i < n; ++i)
      insert (b, a[i], i);
         // b    = Feld, in das eingefügt wird
         // a[i] = einzufügendes Element
         // i    = Anzahl von Elementen in b
   return b;
   } // end of sort ()
```

Teilproblem: Wie fügt man ein ???

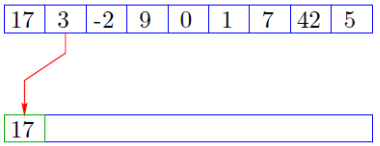


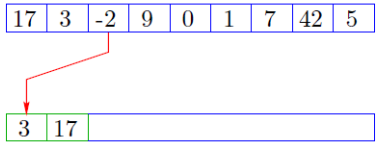




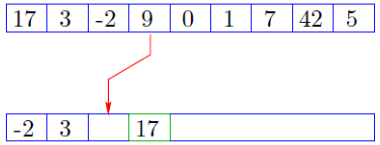





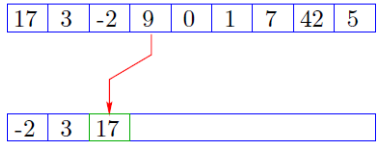












```
public static void insert (int[] b, int x, int i) {
   int j = locate (b,x,i);
      // findet die Einfügestelle j für x in b
   shift (b,j,i);
      // verschiebt in b die Elemente b[j],...,b[i-1]
      // nach rechts
   b[j] = x;
   }
```

## Neue Teilprobleme:

- Wie findet man die Einfügestelle?
- Wie verschiebt man nach rechts?


```
public static int locate (int[] b, int x, int i) {
   int j = 0;
   while (j < i && x > b[j]) ++j;
   return j;
   }

public static void shift (int[] b, int j, int i) {
   for (int k = i-1; k >= j; --k)
       b[k+1] = b[k];
   }
```

- Warum läuft die Iteration in `shift()` von `i-1` abwärts nach `j` ?
- Das zweite Argument des Operators `&&` wird nur ausgewertet, sofern das erste `true` ergibt (Kurzschluss-Auswertung!). Sonst würde hier auf eine uninitialisierte Variable zugegriffen !!!

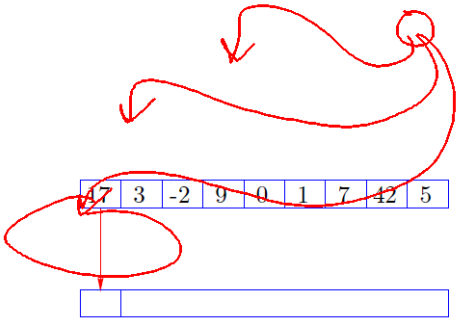


- Das Feld `b` ist (ursprünglich) eine lokale Variable von `sort()`.
- Lokale Variablen sind nur im eigenen Funktionsrumpf sichtbar, nicht in den aufgerufenen Funktionen !
- Damit die aufgerufenen Hilfsfunktionen auf `b` zugreifen können, muss `b` explizit als Parameter übergeben werden !

  Achtung:

  Das Feld wird nicht kopiert. Das Argument ist der Wert der Variablen `b`, also nur eine Referenz !
- Deshalb benötigen weder `insert()`, noch `shift()` einen separaten Rückgabewert ...
- Weil das Problem so klein ist, würde eine erfahrene Programmiererin hier keine Unterprogramme benutzen ...








- Das Feld `b` ist (ursprünglich) eine lokale Variable von `sort()`.
- Lokale Variablen sind nur im eigenen Funktionsrumpf sichtbar, nicht in den aufgerufenen Funktionen !
- Damit die aufgerufenen Hilfsfunktionen auf `b` zugreifen können, muss `b` explizit als Parameter übergeben werden !

  Achtung:

  Das Feld wird nicht kopiert. Das Argument ist der Wert der Variablen `b`, also nur eine Referenz !
- Deshalb benötigen weder `insert()`, noch `shift()` einen separaten Rückgabewert ...
- Weil das Problem so klein ist, würde eine erfahrene Programmiererin hier keine Unterprogramme benutzen ...

```
public static int[] sort (int[] a) {
   int[] b = new int[a.length];
   for (int i = 0; i < a.length; ++i) {
        // begin of insert
        int j = 0;
        while (j < i && a[i] > b[j]) ++j;
           // end of locate
        for (int k = i-1; k >= j; --k)
           b[k+1] = b[k];
           // end of shift
        b[j] = a[i];
        // end of insert
        }
    return b;
    } // end of sort
```



## Diskussion:

- Die Anzahl der ausgeführten Operationen wächst quadratisch in der Größe des Felds a ??
- Glücklicherweise gibt es Sortier-Verfahren, die eine bessere Laufzeit haben (↑Algorithmen und Datenstrukturen).



$1 + 2 + 3 + \ldots + n =$

$\frac{n \cdot (n+1)}{2}$

```
public static int[] sort (int[] a) {
   int[] b = new int[a.length];
   for (int i = 0; i < a.length; ++i) {
        // begin of insert
        int j = 0;
        while (j < i && a[i] > b[j]) ++j;
           // end of locate
        for (int k = i-1; k >= j; --k)
           b[k+1] = b[k];
           // end of shift
        b[j] = a[i];
        // end of insert
        }
    return b;
    } // end of sort
```

149

Diskussion:

- Die Anzahl der ausgeführten Operationen wächst quadratisch in der Größe des Felds a ??
- Glücklicherweise gibt es Sortier-Verfahren, die eine bessere Laufzeit haben (↑Algorithmen und Datenstrukturen).

150

# 7 Eine zweite Anwendung: Suchen

Nehmen wir an, wir wollen herausfinden, ob das Element 7 in unserem Feld `a` enthalten ist.

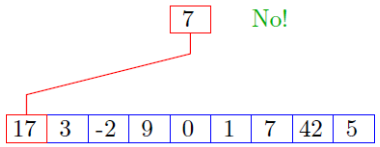
## Naives Vorgehen:

- Wir vergleichen 7 der Reihe nach mit den Elementen `a[0]`, `a[1]`, usw.
- Finden wir ein `i` mit `a[i] == 7`, geben wir `i` aus.
- Andernfalls geben wir `-1` aus: "Sorry, gibt's leider nicht !"



```
public static int find (int[] a, int x) {
   int i = 0;
   while (i < a.length && a[i] != x)
      ++i;
   if (i == a.length)
      return -1;
   else
      return i;
   }
```








- Im Beispiel benötigen wir 7 Vergleiche.
- Im schlimmsten Fall benötigen wir bei einem Feld der Länge $n$ sogar $n$ Vergleiche ??
- Kommt 7 tatsächlich im Feld vor, benötigen wir selbst im Durchschnitt $(n+1)/2$ viele Vergleiche ??

Geht das nicht besser ???

## Idee:

- Sortiere das Feld.
- Vergleiche 7 mit dem Wert, der in der Mitte steht.
- Liegt Gleichheit vor, sind wir fertig.
- Ist 7 kleiner, brauchen wir nur noch links weitersuchen.
- Ist 7 größer, brauchen wir nur noch rechts weiter suchen.

$\Longrightarrow$ binäre Suche ...