Daten, Algorithmen, Strategien

Algorithmus versus Implementierung

Beispiel: summiere $1 + 2 + \cdots + n$

```
int sumupLoop(int n) {
   int sum = 0;
   while (n > 0) {
      sum += n;
      n--;
   }
   return sum;
}
```

```
int sumupRec(int n) {
   if (n > 0) {
      return n + sumupRec(n-1);
   } else {
      return 0;
   }
}
```

gleicher Algorithmus – unterschiedliche Implementierungen

Algorithmus versus Problem

es gilt: $1 + 2 + \cdots + n = \frac{n \cdot (n+1)}{2}$

```
int triangleNum(int n) {
   if (n > 0) {
      return n * (n + 1) / 2;
   } else {
      return 0;
   }
}
```

gleiches Problem – unterschiedliche Algorithmen

Beispiel für Datenstruktur (Stack)

```
public class IntStack {
   private int[] elems;
   private int top = 0;

   public IntStack(int max) { elems = new int[max]; }

   public void push(int elem) { elems[top++] = elem; }

   public int pop() { return elems[--top]; }
}
```

Typische Datenstrukturen

- Array
- verkettete Liste
- binärer Baum
- Hashtabelle
- Stack

Datenstrukturen und Algorithmen

- Datenstruktur $\neq$ Implementierung der Datenstruktur
- Datenstrukturen verwenden andere Datenstrukturen
- Zugriffsoperationen bestimmen Datenstruktur
- Zugriffsoperationen durch Algorithmen festgelegt
- Datenstrukturen und Algorithmen hängen eng zusammen (wie Objektvariablen und Methoden einer Klasse)

Strategiesches Ziel: Einfachheit

Ursachen für Vernachlässigung dieses Ziels:

- Konzentration auf einzelne Teile statt dem Ganzen
- falsche Einschätzung der Komplexität
- offensichtlich $\neq$ einfach
  (einfache Lösungen erfordern mehr Wissen)

$\Rightarrow$ bewährte Strategien statt übertriebenem Effizienzdenken

Lösungsstrategien

- Teile und Herrsche
- Top Down
- Bottom Up
- Schrittweise Verfeinerung
- Verwendung vorgefertigter Teile

Rekursive Datenstrukturen

Verkettete Liste