Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

Algorithmen und Datenstrukturen
FS 2008

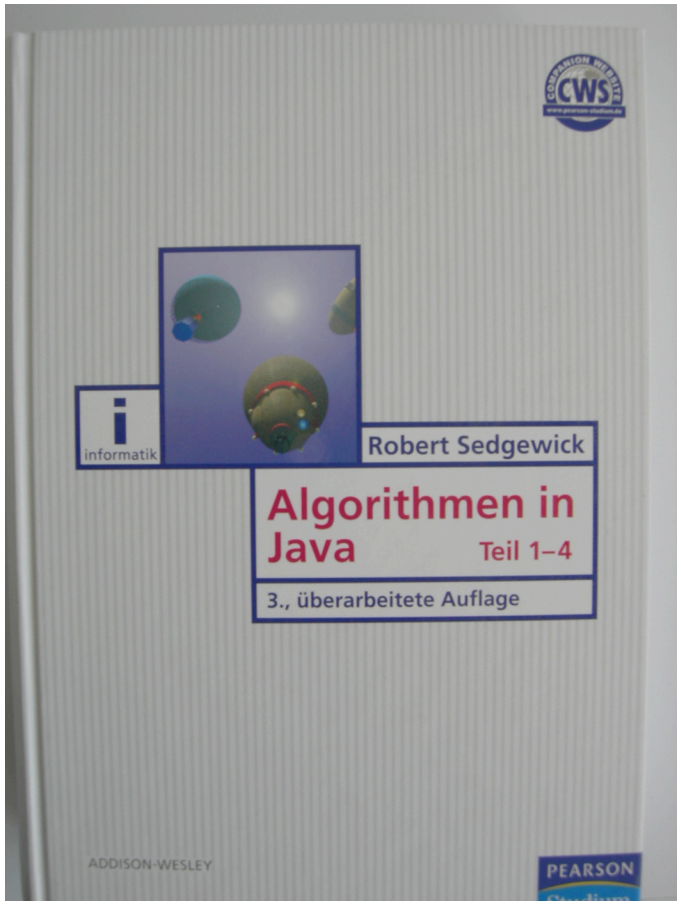
Helmut Schauer

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

Inhalt

- Listen, Stacks und Queues
- Suchen und Sortieren
- Bäume
- Graphen
- Geometrische Algorithmen
- Textverarbeitung

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

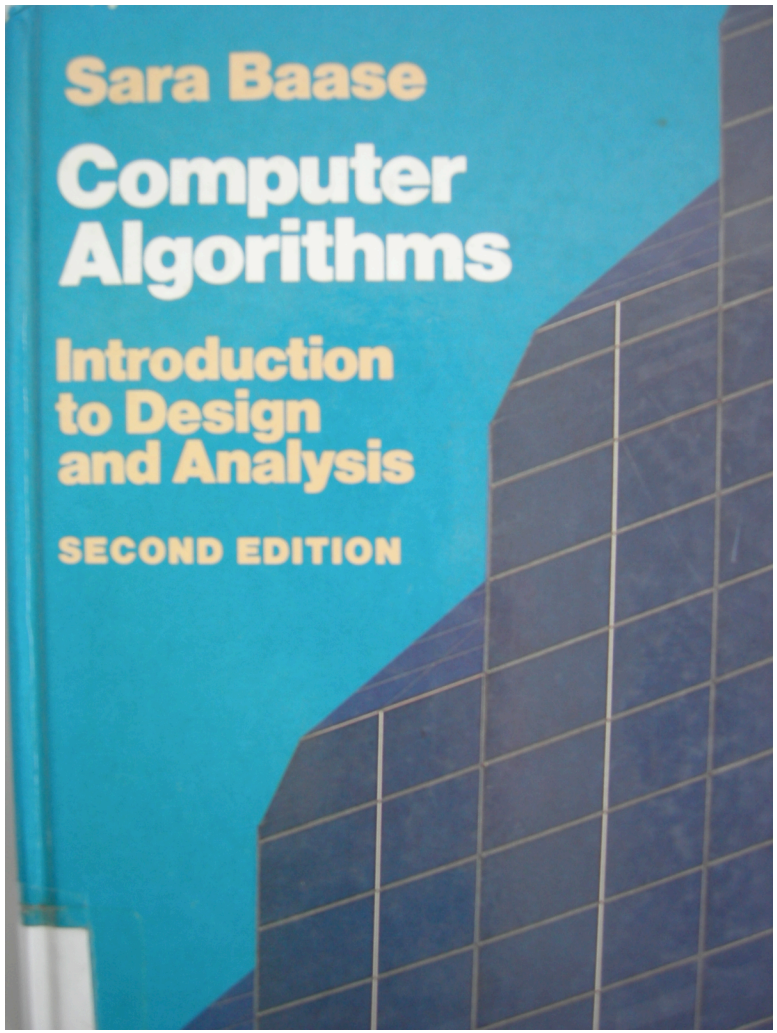


Literatur

Sedgewick, R.:
Algorithms in Java (1-4)
Algorithms in Java (5):
Graph Algorithms

Addison-Wesley, 2003

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

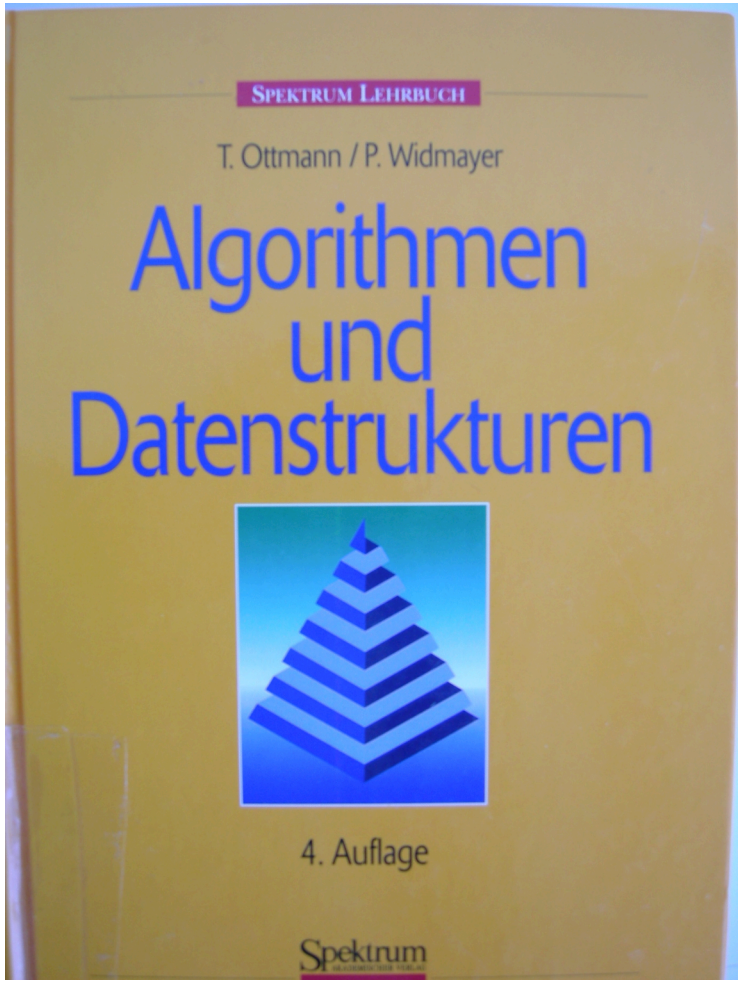


Literatur

Baase, S.:
Computer Algorithms

Addison-Wesley, 1988

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

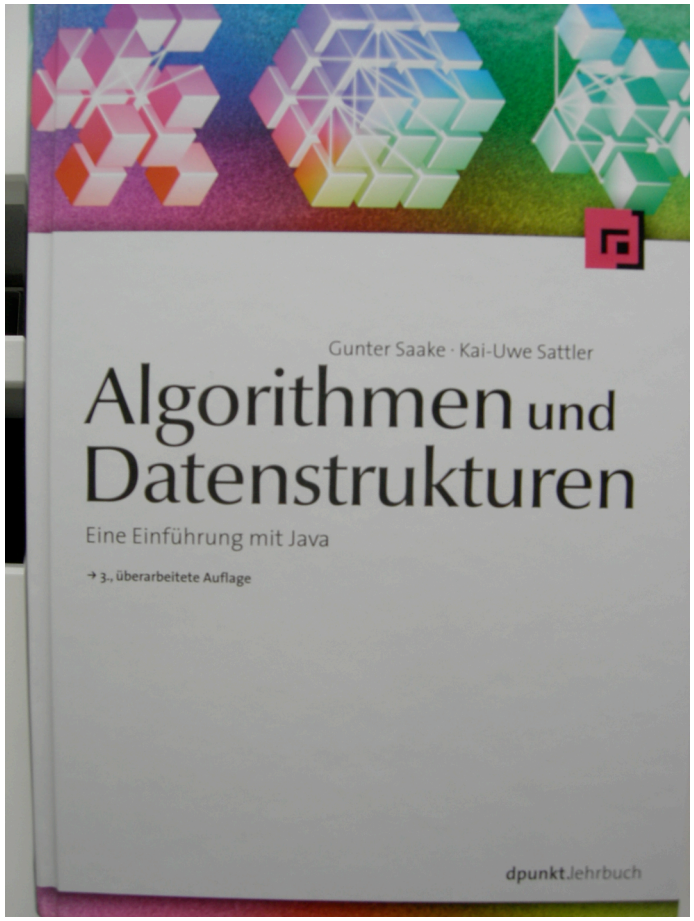


Literatur

**Ottmann, T.; Widmayer, P.:
Algorithmen und
Datenstrukturen**

Spektrum Akad. Verlag, 1996

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

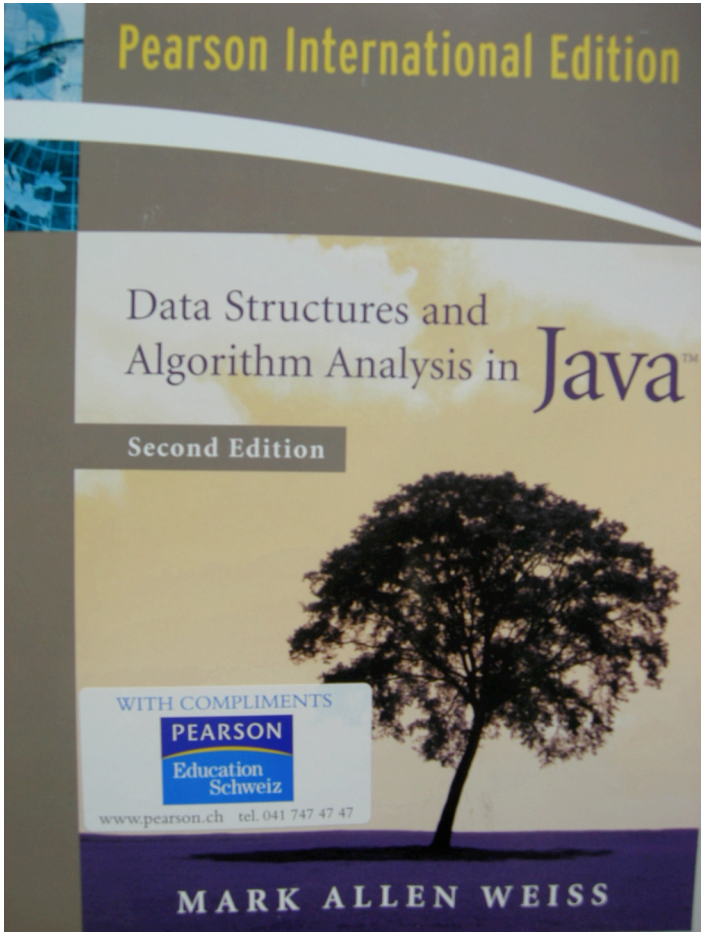


Literatur

Saake, G.; Sattler, K.:
Algorithmen und Datenstrukturen
Eine Einführung mit Java

dpunkt.lehrbuch, 2001

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

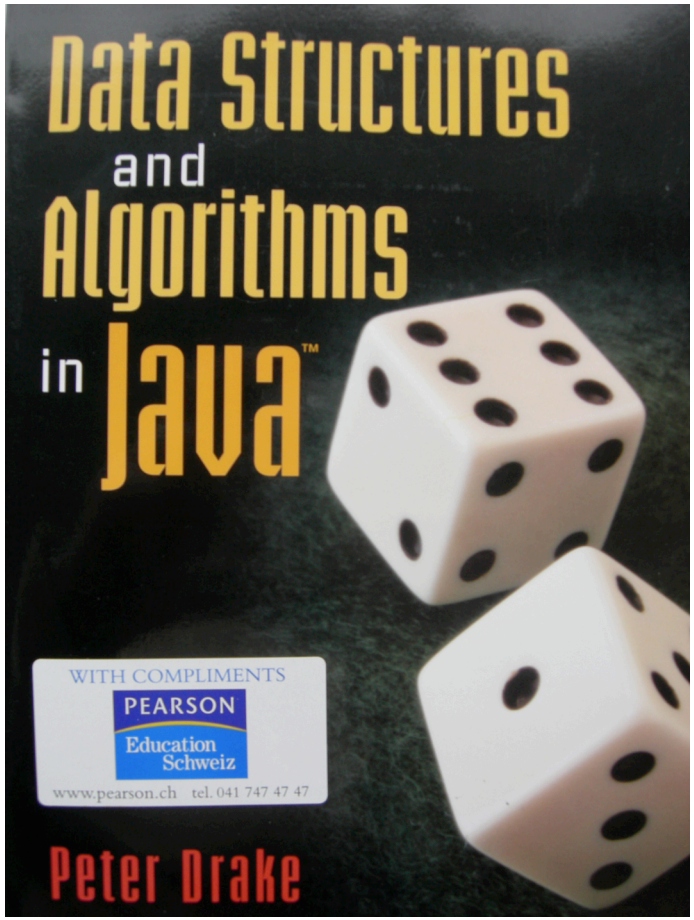


Literatur

Weiss, M.:
Data Structures and Algorithm Analysis in Java

Pearson , 2007

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

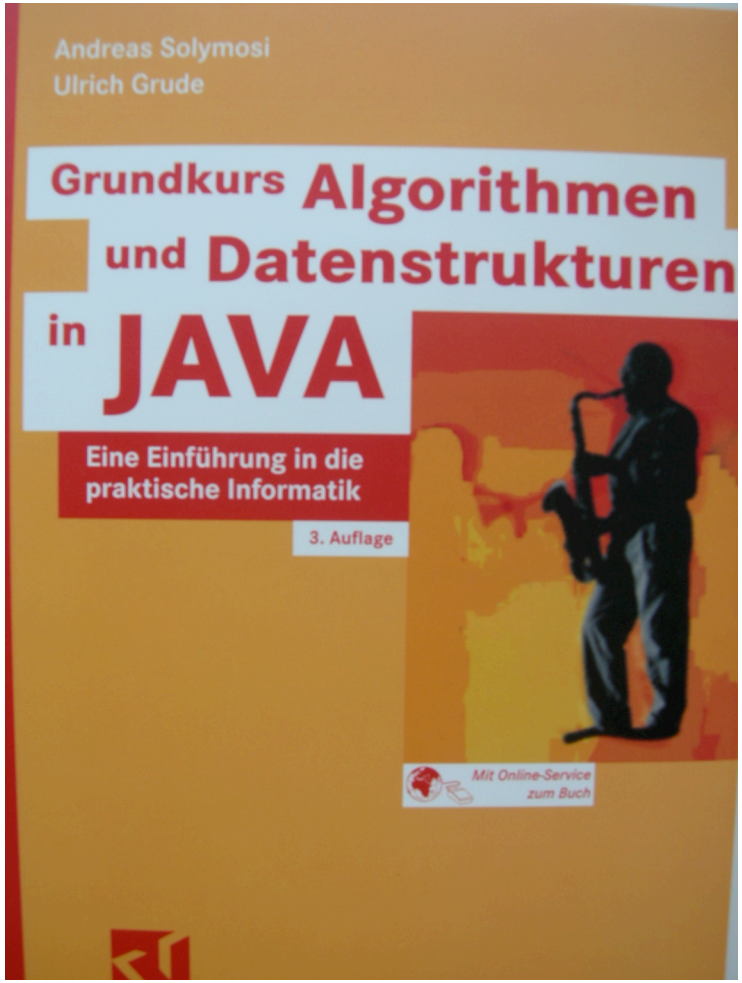


Literatur

Drake, P.:
Data Structures and Algorithms in Java

Pearson, 2006

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich



Literatur

Solymosi, A.; Grude, U.: Grundkurs Algorithmen und Datenstrukturen in Java

Vieweg, 2000

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

Insertion Sort (1)

```
for (i=1;i<N,i++) {
    {All j:1≤j<i:a[j-1]≤a[j] ⇔ ordered(a[0..i-1])}
    k=i; x=a[i];
    while ((k>0)&&(x<a[k-1]))
        a[k--]=a[k];
    a[k]=x;
    {All j:1≤j≤i:a[j-1]≤a[j] ⇔ ordered(a[0..i])}
}
{All j:1≤j<N:a[j-1]≤a[j] ⇔ ordered(a[0..N-1])}
```

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

Insertion Sort (2)

Vergleiche:

- **best case (sortiert): N**
- **worst case (verkehrt sortiert): $N^2/2$**
- **average case: $N^2/4$**

Bewegungen:

- **best case (sortiert): 0**
- **worst case (verkehrt sortiert): $N^2/2$**
- **average case: $N^2/4$**

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

Selection Sort (1)

```
for (i=0;i<N,i++) {
    {All j:1≤j<i:a[j-1]≤a[j] ⇔ ordered(a[0..i-1])}
    min=i;
    for (k=i+1;k<N,k++)
        if (a[k]<a[min]) min=k;
    swap(i,min);
    {All j:1≤j≤i:a[j-1]≤a[j] ⇔ ordered(a[0..i])}
}
{All j:1≤j<N:a[j-1]≤a[j] ⇔ ordered(a[0..N-1])}
```

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

Selection Sort (2)

$N^2/2$ Vergleiche
N Vertauschungen

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

Bubble Sort

$N^2/2$ Vergleiche
$N^2/2$ Vertauschungen

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

QuickSort

"divide et impera"

Worst Case: $A(N) = N(N-1)/2 \in O(N^2)$

Best Case: $A(N) = N \text{ ld } N \in O(N \log N)$

Average Case: $A(N) = 2N \ln N \in O(N \log N)$

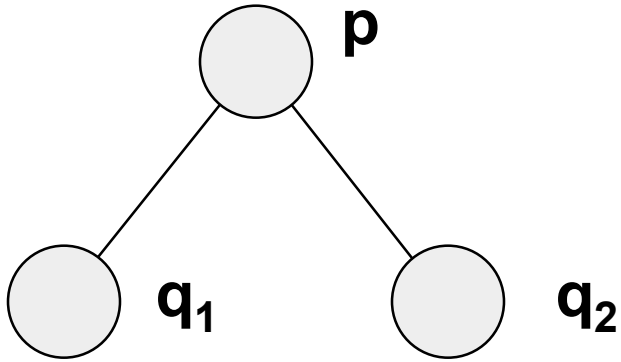
Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

MergeSort

$$A(N) = N \text{ ld } N \in O(N \log N)$$

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

HeapSort (1)

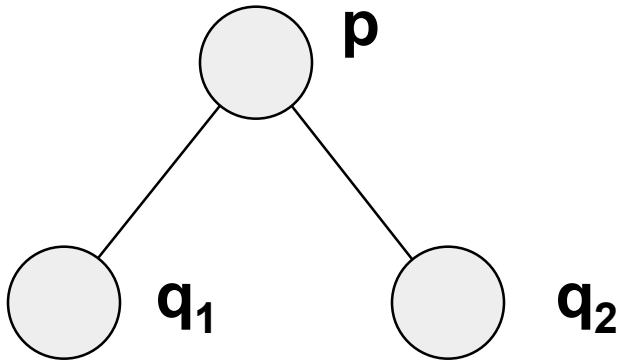


Heap-Bedingung für Maximum Heap:
All p,q: p ist predecessor von q: p≥q

Heap-Bedingung für Minimum Heap:
All p,q: p ist predecessor von q: p≤q

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

HeapSort (2)



Speicherung als Array a[0..N-1]:

p	...	a[i]
q_1	...	a[2i+1]
q_2	...	a[2i+2]

Heap-Bedingung für Maximum Heap:

heap(N) ⇔ All i: 1≤i<N: a[(i-1)/2] ≥ a[i]

Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

HeapSort (3)

1) heap(N) herstellen O(N log N)
2) for (i=N-1; i>0, i--) {
   swap(0,i); O(1)
   heap(i) wiederherstellen; O(log N)
   }

$$A(N) = 2N \text{ ld } N \in O(N \log N)$$