



# Informatik II

## Sortieren

G. Zachmann  
 Clausthal University, Germany  
[zach@in.tu-clausthal.de](mailto:zach@in.tu-clausthal.de)

## Sortieralgorithmen

- Preprocessing fürs Suchen
- sind für kommerzielle Anwendungen häufig die Programmteile, die die meiste Rechenzeit verbrauchen
- viele raffinierte Methoden wurden im Laufe der Zeit entwickelt, von denen wir ein paar kennenlernen wollen

G. Zachmann Informatik 2 - SS 06 Sortieren 2

## Die Sortieraufgabe

- Eingabe: Datensätze (records) aus einem File, der Form
 

Key und satellite/payload data	Sortierschlüssel	Inhalt
--------------------------------	------------------	--------
- Sortierschlüssel kann aus einem oder mehreren Feldern des Datensatzes bestehen (z.B. Nachname + Vorname)
- Ordnungsrelation: auf den Keys muß eine **totale Ordnungsrelation**  $\preceq$  definiert sein, d.h., es gilt
  - Trichotomie:** für alle Keys a,b gilt genau eine Relation  
 $a \prec b, \quad a = b, \quad a \succ b$
  - Transitivität:**  
 $\forall a, b : a \prec b \wedge b \prec c \Rightarrow a \prec c$
- Aufgabe: bestimme eine Permutation  $\Pi = (p_1, \dots, p_n)$  für die Records, so daß die Keys in nicht-fallender Ordnung sind:  

$$K_{p_1} \preceq \dots \preceq K_{p_n}$$

G. Zachmann Informatik 2 - SS 06 Sortieren 3

- Implementierung üblicherweise als Klasse (oder **struct** in C)

```

class MyData:
    def __init__( self, key, value ):
        self.key = key
        self.value = value
    def __cmp__( self, other ):
        if self.key < other.key:
            return 1
        elif self.key > other.key:
            return -1
        else:
            return 0
a = MyData(..)
b = MyData(..)
if a < b:
    ...
  
```

```

class Student
{
    String Name;
    String Vorname;
    long int MatrikelNr;
    short int Fachbereich;
};
  
```

G. Zachmann Informatik 2 - SS 06 Sortieren 4

## Klassifikation / Kriterien von Sortierverfahren

- interne Sortierverfahren
  - alle Datensätze befinden sich im Hauptspeicher
  - es besteht *random access* auf den gesamten Datenbestand
  - bekannte Verfahren: Bubblesort, Insertionsort, Selectionsort, Quicksort, Heapsort
- externe Sortierverfahren
  - die Datensätze befinden sich in einem Hintergrundspeicher (Festplatte, Magnetband, etc.) und können *nur sequentiell* verarbeitet werden
  - bekanntes Verfahren: Mergesort

G. Zachmann Informatik 2 - SS 06 Sortieren 5

- Vergleichsbasiert (*comparison sort*): zulässige Operationen auf den Daten sind nur Vergleich und Umkopieren
- Stabil (*stable*): Gleiche Keys behalten ihre relative Lage zueinander, d.h.,
 
$$K_{p_i} = K_{p_j} \wedge i < j \Rightarrow p_i < p_j$$
- Array-basiert vs. Listen-basiert: Können Datensätze beliebig im Speicher angeordnet sein (Liste), oder müssen sie hintereinander im Speicher liegen (Array)
- In-Place (in situ): Algorithmus braucht nur konstanten zusätzlichen Speicher (z.B. Zähler, keine Hilfsarrays)

G. Zachmann Informatik 2 - SS 06 Sortieren 6

## Erster Sortier-Algorithmus: Bubblesort

- Idee
  - vergleiche von links nach rechts jeweils zwei Nachbarelemente und vertausche deren Inhalt, falls sie in der falschen Reihenfolge stehen;
  - wiederhole dies, bis alle Elemente richtig sortiert sind;
  - die kleinsten Elemente steigen wie *Luftblasen* zu ihrer richtigen Position auf.
- Python-Code:
 

```
def bubblesort( a ):
    for k in range( len(a)-1, 0, -1 ):
        for i in range(0,k):
            if a[i] > a[i+1]:
                a[i], a[i+1] = a[i+1], a[i]
```

G. Zachmann Informatik 2 - SS 06 Sortieren 7

- Beispiel

G. Zachmann Informatik 2 - SS 06 Sortieren 8

### Korrektheits-„Beweis“

- Schleifeninvariante:
  - Nach dem  $i$ -ten Durchlauf befinden sich die  $i$  größten Elemente an der richtigen Position (und in der richtigen Reihenfolge)
  - nach dem 1. Durchlauf befindet sich das größte Element an der richtigen Stelle
  - nach dem 2. Durchlauf auch das 2.-größte, etc.
- nach spätestens  $N-1$  Durchgängen ist das Array sortiert
- da bei jedem Durchlauf auch andere Elemente ihre Position verbessern, ist häufig der Vorgang bereits nach weniger als  $N-1$  Durchgängen beendet

G. Zachmann Informatik 2 - SS 06 Sortieren 9

### Beispiel

0	S	O	R	T	I	E	R	B	E	I	S	P	I	E	L	14
	O	R	S	I	E	R	B	E	I	S	P	I	E	L		
	O	R	I	E	R	B	E	I	S	P	I	E	L	S	T	
	O	I	E	R	B	E	I	R	P	I	E	L	S	S	T	
	I	E	O	B	E	I	R	P	I	E	L	R	S	S	T	
	E	I	B	E	I	O	P	I	E	L	R	R	S	S	T	... etc. ...
	E	B	E	I	I	O	I	E	L	P	R	R	S	S	T	
	B	E	E	I	I	E	L	O	P	R	R	S	S	T		
	B	E	E	I	I	E	L	O	P	R	R	S	S	T		nach 10. BubbleUp
	B	E	E	E	I	I	L	O	P	R	R	S	S	T		Sortiert !

G. Zachmann Informatik 2 - SS 06 Sortieren 10

### Kleine Optimierung: Test auf vorzeitiges Ende

```
def bubblesort( a ):
    for k in range( len(a)-1, 0, -1 ):
        sorted = true
        for i in range( 0,k ):
            if a[i] > a[i+1]:
                a[i], a[i+1] = a[i+1], a[i]
                sorted = false
        if sorted:
            break
```

G. Zachmann Informatik 2 - SS 06 Sortieren 11

### Aufwand von Bubblesort

Laufzeitberechnung für den schlimmsten Fall

```
def bubblesort( a ):
    k = len(a)-1
    while k >= 0:
        for i in range( 0,k ):
            if a[i]>a[i+1]:
                a[i], a[i+1] = a[i+1], a[i]
```

Diagramm zur Laufzeitberechnung  $T(n)$  mit Kosten  $c_0, c_1, c_2, c_3$ .

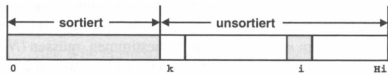
$$T(n) = \sum_{k=1}^n c_3' k = c_3' \sum_{k=1}^n k = c_3' \frac{1}{2} n(n+1) \approx cn^2 \in O(n^2)$$

- Für den besten Fall (für den Code mit "early exit"):  $T(n) = cn$
- Beweis: Übungsaufgabe
- Im durchschnittlichen Fall (o.Bew.):  $T(n) \approx cn^2$

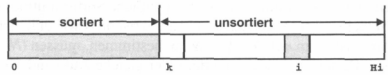
G. Zachmann Informatik 2 - SS 06 Sortieren 12

## Selectionsort

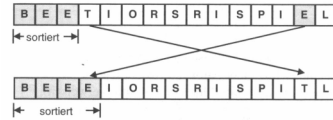
- Sortieren durch Auswahl
- suche das kleinste Element  $A[i]$  der noch nicht sortierten Elemente



- und hänge es an die Menge der sortierten Elemente hinten an (durch Vertauschung)



## Beispiel



## Python Code

```
def minPos(a, lo, hi):
    min = lo;
    for i in range(lo+1, hi):
        if a[i] < a[min]:
            min = i
    return min

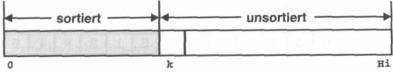
def selectionsort(a):
    for k in range(0, len(a)-1):
        min = minPos(a, k, len(a))
        if min != k:
            a[min], a[k] = a[k], a[min]
```

## Laufzeitbetrachtung

- Ebenfalls  $\approx N^2$ , da wieder zwei ineinander geschachtelte Schleifen (ganz ähnlich zu Bubblesort)
- aber Vorfaktor niedriger als bei Bubblesort, da nur einmal pro Durchgang getauscht wird
- allerdings braucht Selectionsort immer gleich viele Durchläufe, während Bubblesort häufig vorzeitig abbricht, insbesondere bei vorsortierten Arrays

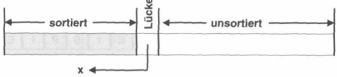
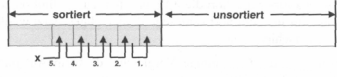
## Insertionsort

- Sortieren durch Einfügen
  - wie das Sortieren beim Kartenspielen
    - eine Karte nach der anderen wird aufgenommen und an die richtige Stelle einsortiert
  - sei ein Teil des Arrays schon sortiert



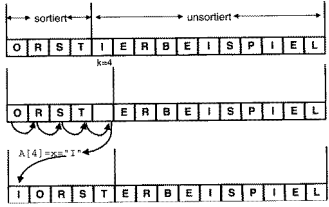
0                      k                      Hi

G. Zachmann    Informatik 2 - SS 06                      Sortieren    17

- dann wird das nächste Element  $A[k]$  herausgenommen
 
- die entstehende Lücke wird von links durch Verschieben der bereits sortierten Elemente aufgefüllt, bis die Stelle gefunden ist, an die  $A[k]$  gehört
 

G. Zachmann    Informatik 2 - SS 06                      Sortieren    18

## Beispiel



0                      k                      Hi

G. Zachmann    Informatik 2 - SS 06                      Sortieren    19

## Algo-Animation

### Insertion Sort an N Element Array

In  $i^{\text{th}}$  iteration:

- Read  $i^{\text{th}}$  value.
- Repeatedly swap  $i^{\text{th}}$  value with the one to its left if smaller.

Property: after  $i^{\text{th}}$  iteration, array positions 0 through  $i$  contain original elements 0 through  $i$  in increasing order.

Array index	0	1	2	3	4	5	6	7	8	9
Value	2.78	7.42	0.56	1.12	1.17	0.32	6.21	4.42	3.14	7.71

Iteration 0: step 0

Courtesy: Koehn, Wayne & Robert Sedgwick

G. Zachmann    Informatik 2 - SS 06                      Sortieren    20

Python Code

```
def insertionsort(a):
    for k in range(1, len(a)):
        x = a[k]
        i = k
        while i > 0 && a[i-1] > x:
            a[i] = a[i-1]
            i -= 1
        a[i] = x
```

Geht dieser Test gut?  
(ist a[i-1] immer wohldefiniert??)

The diagram shows an array with a shaded region on the left labeled 'sortiert' (sorted) and an unshaded region on the right labeled 'unsortiert' (unsorted). The sorted region contains elements x, 4, 2, 1. Arrows point from the elements 4, 2, and 1 to the position of x, indicating they are being shifted to the right to make space for x. The unsorted region is currently empty.

G. Zachmann Informatik 2 - SS 06 Sortieren 21