

Einführung in die Informatik I

Kapitel II.3: Sortieren

Prof. Dr. Marcin Grzegorzek¹

Research Group for Pattern Recognition
www.pr.informatik.uni-siegen.de

Institute for Vision and Graphics
University of Siegen, Germany



¹Die im Rahmen dieser Lehrveranstaltung verwendeten Lernmaterialien wurden uns zum Großteil von Herrn Prof. Dr. Wolfgang Wiechert und Herrn Prof. Dr. Roland Reichardt zur Verfügung gestellt.

Testlauf für den neuen Übungsbetrieb

- PC Pools werden künftig mit Zero Clients ausgestattet.
- Ein Testlauf dieser Lösung wird benötigt.
- Die Übung am 30.01 von 12 bis 14 Uhr (Gruppen 23 und 24, Übungsblatt 5) findet im Raum H-D 2202 statt.m

Inhaltsverzeichnis

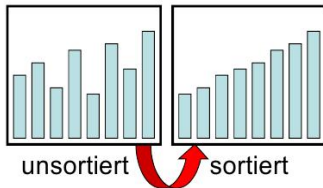
I. MATLAB-Einführung

1. Voraussetzungen und Konventionen
2. Variablen und arithmetische Ausdrücke
3. Automatisierungen von Berechnungen
4. Logische Ausdrücke
5. Verzweigungen
6. Schleifen
7. Fehlersuche in Programmen
8. Funktionen
9. Arbeitsweise von Funktionen
10. Vektoren
11. Matrizen

II. Algorithmen

1. Suchen
2. Spezielle Suchalgorithmen
- ▶ 3. Sortieren
4. Rekursion und Quicksort

Sortieren



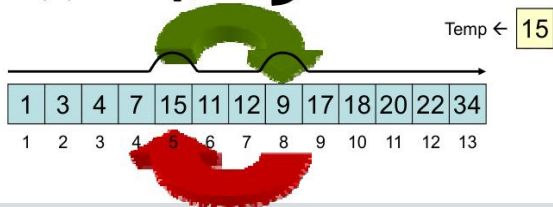
- Beim Sortieren wird ein Feld mit Elementen nach einem **Schlüssel** mit Hilfe eines **Sortierkriteriums** geordnet.
- Beispiel Telefonbuch:
 - **Schlüssel:** Nachname, Vorname
 - **Sortierkriterium:** Vorgänger < Nachfolger
- In der Vorlesung werden verschiedene Sortierverfahren für **Zahlen in einem Vektor** beschrieben, um sich auf algorithmische Fragen zu konzentrieren.

Vertauschen von Inhalten

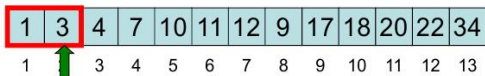
- Grundlegende Funktion bei Sortierverfahren.
- Beispiel: „fast“ sortierte Zahlenfolge
 - 15 und 9 sind nicht in der richtigen Reihenfolge
 - Die Inhalte Speicherplätze 5 und 8 müssen getauscht werden.

```
□ Temp      = Feld(5) ;  
□ Feld(5)   = Feld(8) ;  
□ Feld(8)   = Temp ;
```

} **Tausch!**

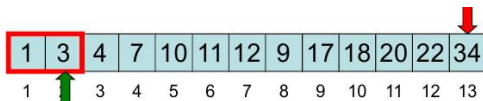


Bubble Sort (erste Version)



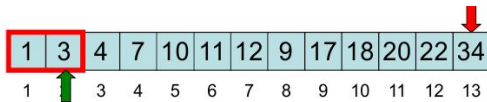
- Bubble Sort beruht auf dem wiederholten Durchlaufen aller Daten von links nach rechts (rotes Fenster). Stehen zwei Werte in falscher Reihenfolge, so werden sie getauscht.
- Dazu laufen zwei geschachtelte Schleifen über die Daten
 - Die erste Schleife ruft lediglich die zweite Schleife n -Mal auf (n = Anzahl der Elemente im Feld).
 - Die zweite Schleife (grüner Zeiger) läuft jeweils von Position 2 bis n . Das rote Fenster wird dabei über die Daten gezogen.
 - Im Fenster werden jeweils die linke und rechte Position verglichen.
 - Bei Fehlordnung Tauschen der beiden Werte im Fenster.
- Das kleinste Element wandert dabei pro Durchlauf um eine Position nach links. Dies muss maximal n Mal passieren, damit ein Element im Worst Case von ganz rechts nach ganz links wandern kann.

Quelltext: Bubble Sort (1)



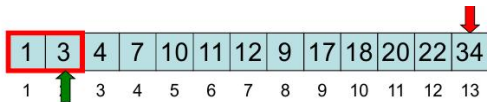
```
function x = bubble_sort(x)
n=length(x);           % Anzahl
for i = 1:n           % Durchlaufzähler
    for j =2:n        % Fenster-Durchlauf
        if x(j-1) > x(j)
            temp =x(j-1); % Tauschen
            x(j-1)=x(j ); % ...
            x(j )=temp ; % ...
        end
    end
end
```


Bubble Sort (zweite Version)



- Feststellung: Das größte Element steht nach dem ersten Durchlauf immer ganz rechts. Daher muss das Fenster beim zweiten Durchlauf nur noch bis $n-1$ gezogen werden. Beim i -ten Durchlauf nur noch bis Position $n-i+1$.
- Dies erklärt auch warum der Algorithmus funktioniert.
- Realisierung mit einem weiteren Zeiger (rot):
 - Das Ende des aktuellen Fenster-Durchlaufs wird durch den roten Zeiger vorgegeben.
 - Dieser läuft jetzt über die Werte $n, n-1, n-2, \dots, 2$.

Quelltext: Bubble Sort (2)



```
function x = bubble_sort(x)
n=length(x);           % Anzahl
for i = n:-1:1        % von rechts nach links
    for j =2:i        % von links bis i
        if x(j-1) > x(j)
            temp =x(j-1); % Tauschen
            x(j-1)=x(j ); % ...
            x(j )=temp ; % ...
        end
    end
end
```

Bubble Sort (dritte Version)



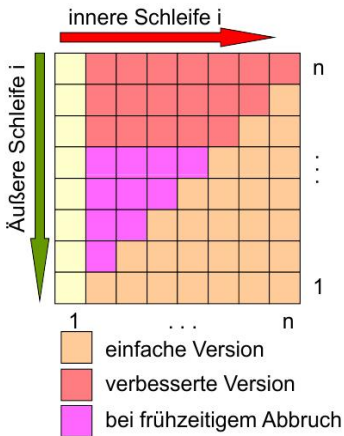
- Situation von eben:
 - „9“ ist von Position 8 nach 5 gewandert
 - zweiter Durchlauf.
- Diesmal nur bis Feld Nr. 2
- Wenn in einem Durchlauf **nicht getauscht** wurde, dann ist das Feld sortiert.
- Das Programm muss sich also **merken, ob getauscht wurde**, um die Sortierung abubrechen.

Quelltext: Bubble Sort (3)

```
function x = bubble_sort(x)
n=length(x);           % Anzahl
for i = n:-1:1        % von rechts nach links
    getauscht = 0;    % Merker für Tausch
    for j =2:i        % von links bis i
        if x(j-1) > x(j)
            temp=x(j-1); % Tauschen
            x(j-1)=x(j); % ...
            x(j)=temp;   % ...
            getauscht=1; % Merker für Tausch
        end
    end
    if ~getauscht % Wenn nicht getauscht...
        break    % Sortierung beendet!
    end
end
end
```

Laufzeitanalyse: Bubble Sort

- Bubble Sort durchläuft zwei geschachtelte Schleifen



- Anzahl der Schleifendurchläufe

- einfach

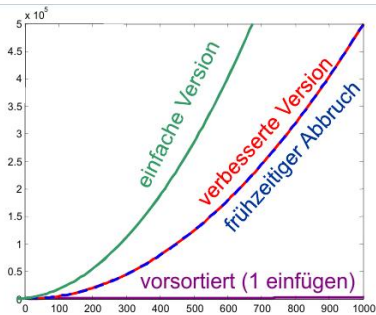
$$\begin{aligned}n \cdot (n-1) \\ = n^2 + n \approx n^2\end{aligned}$$

- verbessert

$$\begin{aligned}(n-1) + (n-2) + \dots + 2 + 1 \\ = 1/2 \cdot n \cdot (n-1) = \\ = 1/2 \cdot n^2 - 1/2 \cdot n \approx 1/2 \cdot n^2\end{aligned}$$

- mit Abbruch im worst case etwas besser, aber im Durchschnitt immer noch $\approx \frac{1}{2}n^2$

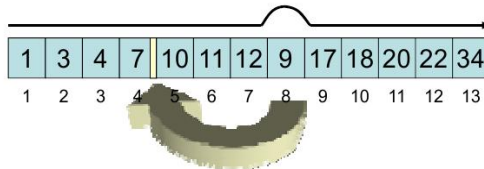
Empirische Laufzeitergebnisse



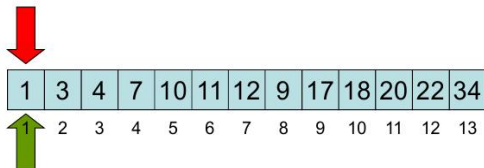
- Bei völlig unsortierten Listen wächst die Rechenzeit mit dem Quadrat der Listenlänge
- Doppelte Listenlänge bedeutet 4-fache Rechenzeit
10-fache Listenlänge bedeutet 100-fache Rechenzeit
- Nur bei vorsortierten Listen ist der frühzeitige Abbruch von Vorteil

Insertion Sort

- Sortieren durch direktes Einfügen
- Kartenspieler sortiert seine Karten:
 - Betrachte ein Element nach dem anderen und füge jedes an die richtige Stelle zwischen den bereits betrachteten ein.
 - Einfügen, indem die größeren Elemente um eins nach rechts geschoben werden, während die Einfügeposition gesucht wird.

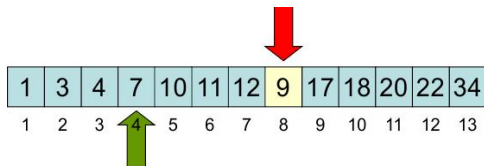


Insertion Sort



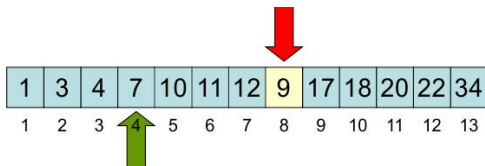
- Zwei Schleifen (Zeiger)
- Die erste Schleife schaut sich die Zahlen nacheinander an.
- Die zweite Schleife sucht **nach jedem Verschieben** des Zeigers der ersten Schleife eine Einfügeposition für die Zahl der ersten Schleife.

Insertion Sort (erste Version)



- Einfügen: „9“ muss zwischen Position 4 und 5 (also nach 4)
- *Mensch (Kartenspiel)*:
 - „9“ rausnehmen,
 - „10, 11, 12“ nach rechts schieben,
 - „9“ in die Lücke einfügen
- Ein Programm kann aus Feldern keine „rausnehmen“ und Felder haben auch keine „Lücken“ zum Einfügen.
- Einfache Lösung: Neuen Wert wie bei Bubble Sort von rechts nach links tauschen bis er an der richtigen Stelle steht.

Implementierung von Insertion Sort



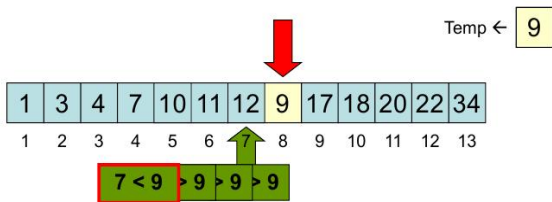
- Die Äußere Schleife braucht **nicht bei 1** loszulaufen, sondern **erst bei 2**, weil die innere Schleife sich immer um den Bereich links von der Äußeren kümmert.
- Äußere Schleife ist eine **for-Schleife**, weil die Schleifengrenzen von Anfang an fest stehen.
- Innere Schleife ist ein **while-Schleife**, weil diese abgebrochen werden kann, wenn die Einfügeposition gefunden wurde.
- Suchen der Einfügestelle und Verschieben der Elemente rechts davon erfolgt simultan in nur einer Schleife

Quelltext: Insertion Sort (1)

```
function x = insertion_sort_bubble(x)
n=length(x);

for i = 2:n
%   temp = x(i); % Zahl „merken“
    j=i; % zweite Schleife beginnt bei i
    while (j>1) && (x(j-1) > x(i))
        temp =x(j-1); % Tauschen wie BUBBLE
        x(j-1)=x(i);   % ...
        x(i)  =temp;   % ...
        j=j-1;        % eine Position runter
    end
end
end
```

Insertion Sort (2)

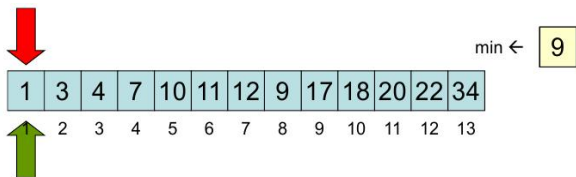


- Im Programm:
 - Zahl der ersten Schleife in temporärer Variable „merken“.
 - Zweite Schleife kopiert die Zahlen nach oben, so lange die aktuelle Zahl größer als die „gemarkte“ Zahl ist.
 - Temporärer Variable wird in letzte Position der zweiten Schleife kopiert.
- Die Verbesserung spart eine Reihe von Zuweisungen.

Quelltext: Insertion Sort (2)

```
function x = insertion_sort(x)
n=length(x);
for i = 2:n
    temp = x(i); % Zahl „merken“
    j=i; % zweite Schleife beginnt bei i
    while (j>1) & (x(j-1) > temp)
        x(j)=x(j-1); % hoch kopieren
        j=j-1; % eine Position runter
    end
    x(j)=temp; % „gemerkte“ Zahl von i kopieren
end
```

Selection Sort



- Sortieren durch Auswahl sucht in jedem Schritt aus einem Stapel noch nicht sortierter Karten das kleinste Element. Dieses wird an die schon sortierten Werte angehängt.
- Zwei Schleifen (Zeiger)
- Die erste Schleife markiert eine Tauschposition am Ende des schon sortierten Teils (links nach rechts).
- Die zweite Schleife sucht **nach jedem Verschieben** des ersten Zeigers die **kleinste Zahl im Rest des Feldes**.
- Diese kleinste Zahl wird mit der Position der ersten Scheife getauscht.

Implementierung von Selection Sort

- Äußere Schleife muss **nicht bis ganz ans Ende** laufen, sondern **nur bis zum Vorletzten** Eintrag. Die innere Schleife kümmert sich immer um den Bereich rechts von der Äußeren.
- Äußere Schleife ist eine **for-Schleife**, weil die Schleifengrenzen von Anfang an fest stehen.
- Die Innere Schleife sucht das kleinste Element im noch nicht sortierten Teil des Vektors: Siehe Suchalgorithmen
- Innere Schleife ist auch eine **for-Schleife**, weil die Schleifengrenzen ebenfalls von Anfang an fest stehen.
- Links von der Äußeren Schleife ist alles sortiert. Die Innere Schleife braucht also nicht vorne zu beginnen.

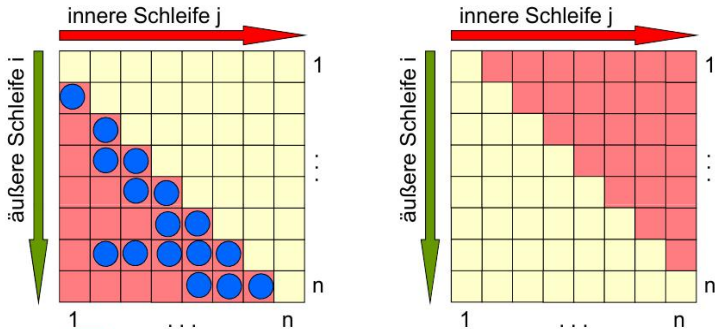
Quelltext: Selection Sort

```
function x = selection_sort(x)
n=length(x); % Anzahl

for i = 1:n-1 % bis zum Vorletzten
    % Minimumssuche von i bis n
    min_i=i; % min initialisieren
    for j = i+1:n % Beginn nach i
        if x(j) < x(min_i) % kleiner als akt. min?
            min_i = j; % neues min
        end
    end
    % Tauschen min <-> aktuelles i
    temp = x(i);
    x(i) = x(min_i);
    x(min_i) = temp;
end
```


Laufzeitanalyse: Insertion/Selection Sort

- Beide Algorithmen durchlaufen zwei geschachtelte Schleifen



 worst case

$$1/2 \cdot n^2 - 1/2 \cdot n \approx 1/2 \cdot n^2$$

 Bei frühzeitigem Abbruch (Mittel)

$$1/4 \cdot n^2 - 1/4 \cdot n \approx 1/4 \cdot n^2$$

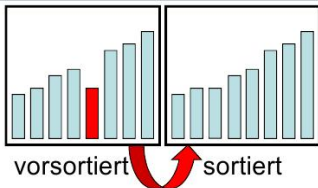
$$1/2 \cdot n^2 - 1/2 \cdot n \approx 1/2 \cdot n^2$$

Zusammenfassung: Sortieren

Verfahren	Bubble	Selection	Insertion
Durchschnittliche Laufzeit	$1/2 \cdot n^2$	$1/4 \cdot n^2$	$1/2 \cdot n^2$

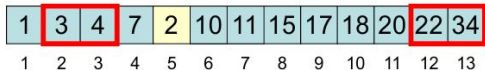
- Selection Sort schneidet am besten ab.
- In der Praxis ist dieser Vorteil marginal, weil eine quadratische Laufzeit bei großen Datenmengen inakzeptabel ist.

Sortieren vorsortierter Listen



- In eine vorsortierte Liste wurde ein neues Element ungeachtet des Sortierschlüssels eingefügt.
- Die Liste muss neu sortiert werden.
- ...

Bubble Sort (vorsortierte Liste)



- Es liegt die vorsortierte Liste vor, lediglich das 5. Element wurde neu, an der falschen Position eingefügt.
- Die erste Schleife (**rechts nach links**) vergleicht den Nachfolger und die aktuelle Position.
- Tauschen der Werte bei Bedarf.
- Das neue Element steht **nach dem ersten** Durchlauf auf jeden Fall auf der **richtigen Position**.
- Der Sortiervorgang nach einem Durchlauf fertig!

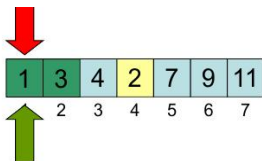
Sortieren vorsortierter Listen

Bubble Sort

1	3	4	2	7	9	11
1	2	3	4	5	6	7

- Die erste Schleife (**jetzt: rechts nach links**) vergleicht den Nachfolger und die aktuelle Position.
- Tauschen der Werte bei Bedarf.
- Das **neue** Element steht **nach dem ersten** Durchlauf auf jeden Fall auf der **richtigen Position**.
- Bei vorsortierten Listen ist der frühzeitige Abbruch von Vorteil.

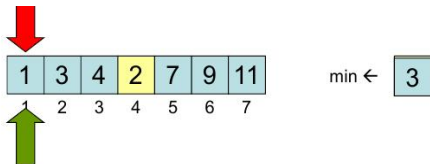
Sortieren vorsortierter Listen



Insertion Sort

- Die erste Schleife schaut sich die Zahlen nacheinander an.
- Die zweite Schleife sucht **nach jedem Verschieben** des Zeigers der ersten Schleife eine Einfügeposition für die Zahl der ersten Schleife.
- Das neue Element wird einsortiert, nachdem die erste Schleife bis an dessen Einfügeposition (4) gelaufen ist.
- Die erste Schleife läuft dann weiter bis ans Ende der Liste weiter.

Sortieren vorsortierter Listen



Selection Sort

- Die erste Schleife dient als Tauschposition (links nach rechts).
- Die zweite Schleife sucht **nach jedem Verschieben** des ersten Zeigers die **kleinste Zahl im Rest des Feldes**.

Logistik: Hochregallager

- Sortieren betrifft nicht nur Dateien oder Datensätze, sondern kann auch wesentlich komplexer sein!
- In einem Hochregallager dauert der Zugriff auf die einzelnen Lagerplätze unterschiedlich lange.
- Um die Wartezeit zu verkürzen, werden häufig angefragte Artikel näher am Eingang/Ausgang einsortiert.
- Diese Art der Lagerhaltung heißt **chaotisch**

