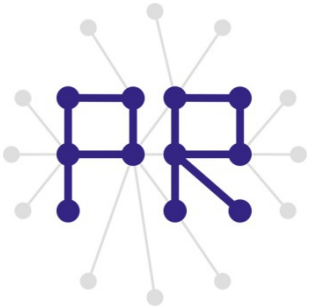
UNIVERSITÄT SIEGEN



Einführung in die Informatik I

Kapitel II.1: Suchen

Prof. Dr.-Ing. Marcin Grzegorzek
Juniorprofessur für Mustererkennung im Institut für Bildinformatik
Department Elektrotechnik und Informatik
Fakultät IV der Universität Siegen

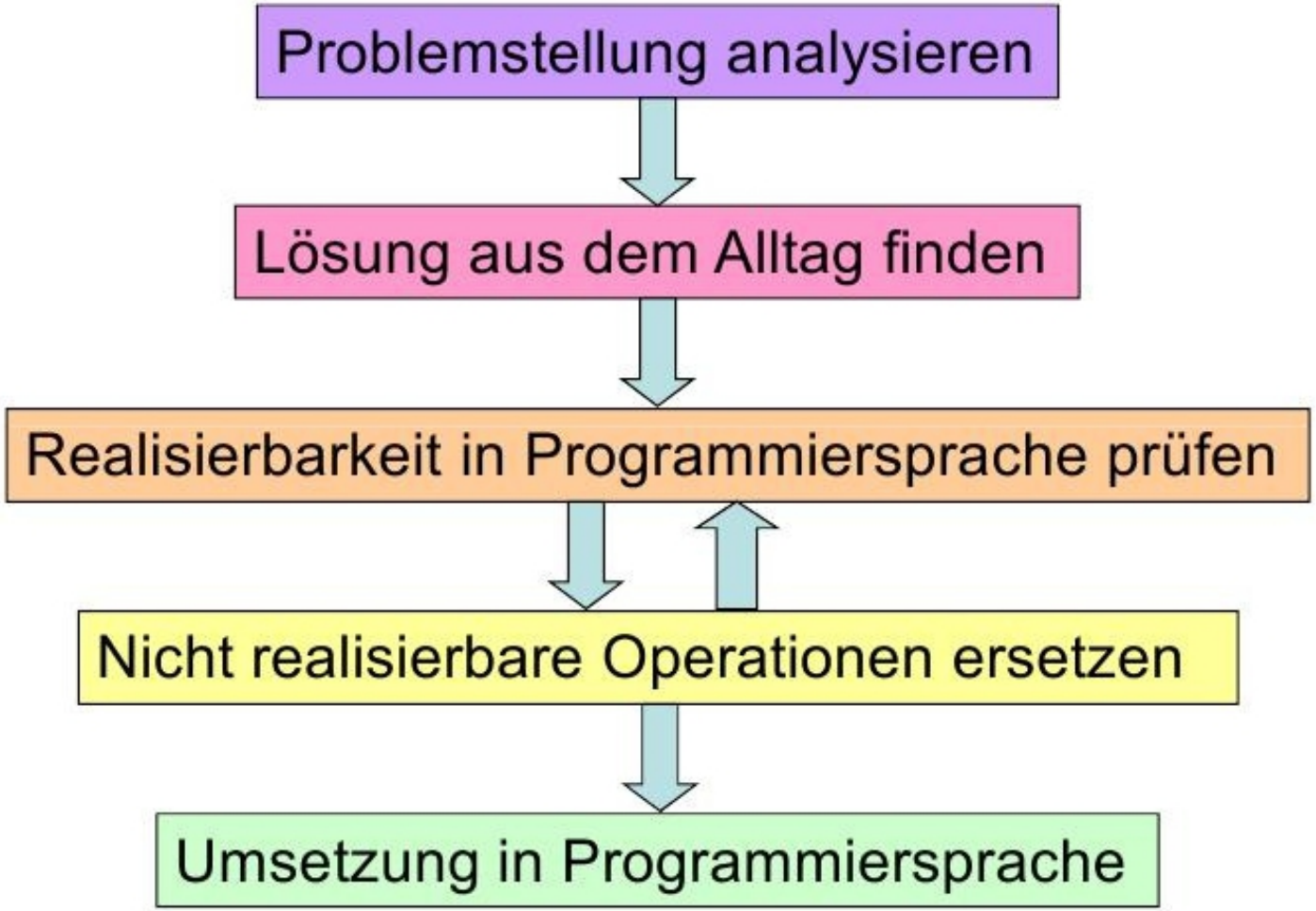
16.01.2013

Inhaltsverzeichnis

Algorithmen

- Der Name stammt aus der mittelalterlichen Übersetzung *Algoritmi de numero Indorum,* eines Werkes des arabischen Mathematikers al-Khwarizmi
- Definition: **Systematische Prozedur**, die in einer **endlichen Zahl** von **Schritten** die Antwort auf einer Frage oder die Lösung eines Problems produziert
- Algorithmen bestehen aus **einzelnen Befehlen**
- Entscheidend ist, dass alle **Befehle unmissverständlich** sind, d.h. sie haben eine klar festgelegte Bedeutung
- Das hängt vom Betrachter ab:
  - Kochrezepte sind klar verständlich für Köche
  - Pseudocode ist klar verständlich für Menschen
  - MATLAB-Programme sind eindeutig für MATLAB

Heuristik zur Algorithmenentwicklung



Suchen

- Eine grundlegende **Operation**, die Bestandteil vieler Berechnungsaufgaben ist
- Das **Wiederauffinden** eines bestimmten **Elements** oder bestimmter **Informationsteile** aus einer großen Menge früher gespeicherter Information
- Ziel: Alle **Datensätze finden**, deren **Schlüssel** mit einem bestimmten **Suchschlüssel übereinstimmen**
- Fertige **Funktionen** sind in MATLAB-System bereits für das Suchen (`find, min, max`) und Sortieren (`sort`) vorhanden

Beispiel: Karteikasten

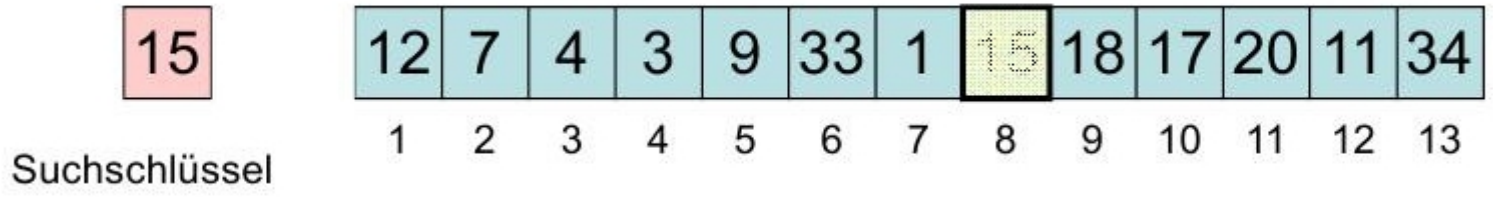
- Die **Namen** in der **Personaldatei** sind alphabetisch **sortiert**
- Jeder **Eintrag** ist ein **Datensatz**
- Der **Name** oder die **Personalnummer** ist der **Schlüssel**
- Der **gesuchte** Name oder die **gesuchte** Personalnummer ist der **Suchschlüssel**

Abstaktion:

- Die Suche nach dem **Namen** entspricht der Suche in einem **sortierten Vektor**
- Die Suche nach der **Personalnummer** entspricht der Suche in einem **unsortierten Vektor** – weil nach Name sortiert

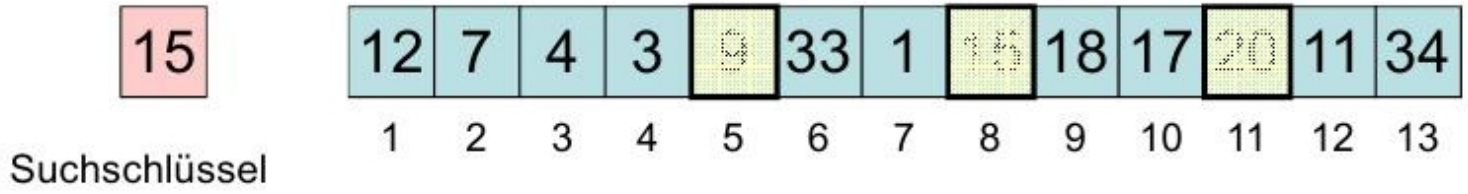
Vektor mit Zahlen

- Als Beispiel wird folgender **Vektor** mit 13 Zahlen verwendet
- Der **Suchschlüssel** ist gegeben: 15
- Der Algorithmus soll den **Suchschlüssel** im Vektor finden
- Ggfs. soll auch ermittelt werden, ob der **gesuchte Schlüssel** überhaupt im Vektor **vorkommt**



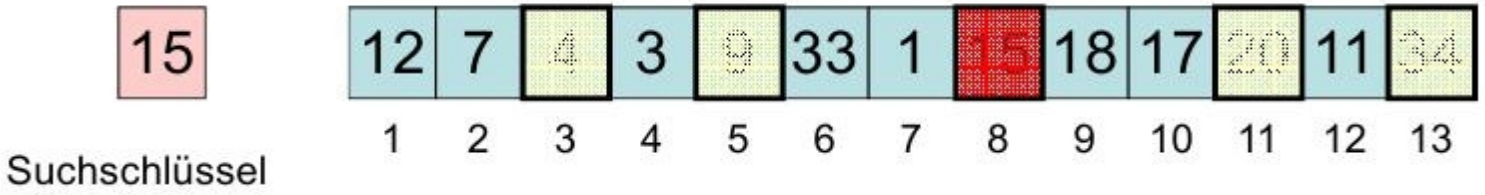
Vergleichsoperationen

- Bei der Suche muss der **Suchschlüssel** mit dem **Inhalt** eines **Elements** des Vektors **verglichen** werden
- Zugelassene Vergleichsoperation
  - < kleiner (9<15) → Vektor(5)<Suchschlüssel
  - > größer (20>15) → Vektor(11)>Suchschlüssel
  - == gleich (15 == 15 , Ende der Suche)



Zufälliges Suchen

- Um die Lösung zu finden, **rät** der Algorithmus eine **Zahl** zwischen 1 und der Länge des Vektors
- Der **Inhalt** des zufällig gefundenen Elements wird mit dem **Suchschlüssel verglichen**
- Dieses Verfahren wird so lange **wiederholt**, bis das Element gefunden wurde
- Beispiel Karteikasten: Irgendeine Karte ziehen, wenn diese falsch war, Karte wieder rein und wieder von Vorne



Beispiel: Zufallssuche

- Der Funktion wird das zu durchsuchende **Feld** (`x`) und der **Suchschlüssel** (`Key`) **übergeben**. Es wird die **Position** des Suchschlüssels im Feld **zurückgegeben**

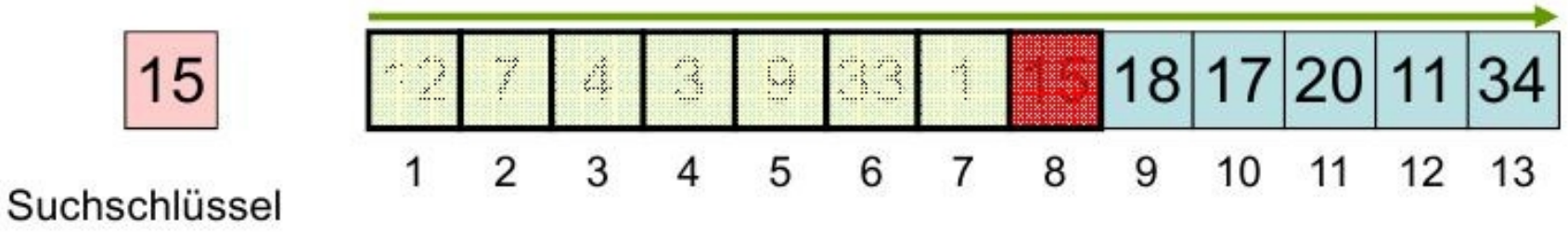
```
function Position=Suche_Zufall(x,Key)
Position=ZufallPosi(length(x));
while x(Position)~=Key
    Position=ZufallPosi(length(x));
end
end

function Posi=ZufallPosi(n)
      Posi=ceil(n*rand);
end
```

- Das zufällige Suchen hat den **Nachteil**, dass die Zufallsfunktion u.U. mehrmals die **selbe Zahl** liefert
- Falls der Suchschlüssel im Feld **nicht vorkommt**, dann **bricht** der Algorithmus **nicht ab** (Endlosschleife)

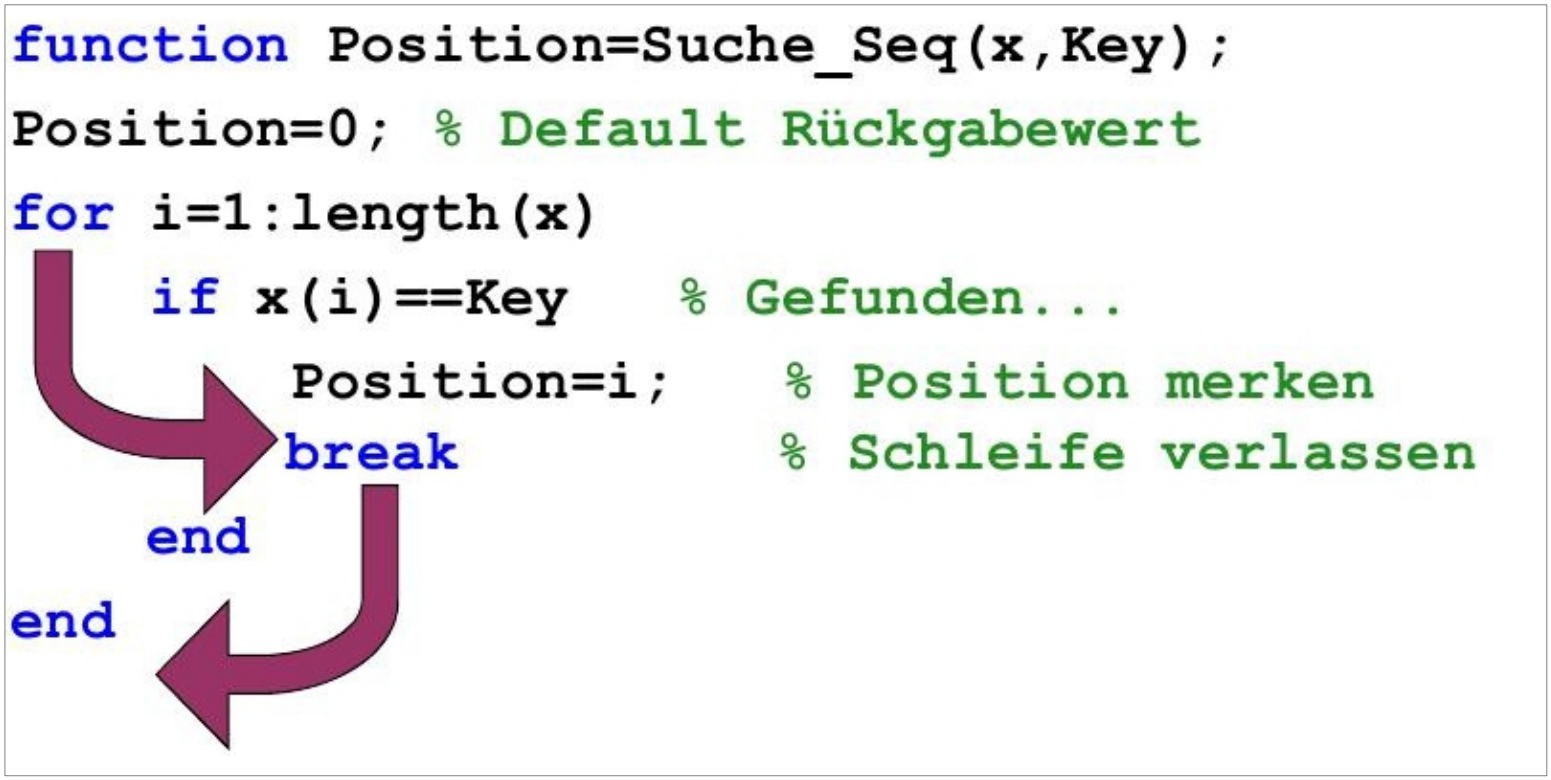
Sequentielle Suche

- Die Elemente werden von **Anfang** beginnend **nacheinander** geprüft
- Die Schleife wird **beendet**, sobald das Element **gefunden** ist
- Die Funktion liefert bei **erfolgloser** Suche die Position **0** zurück
- Andere Namen für dieses Verfahren:
  - Erschöpfendes Durchsuchen
  - Brute Force



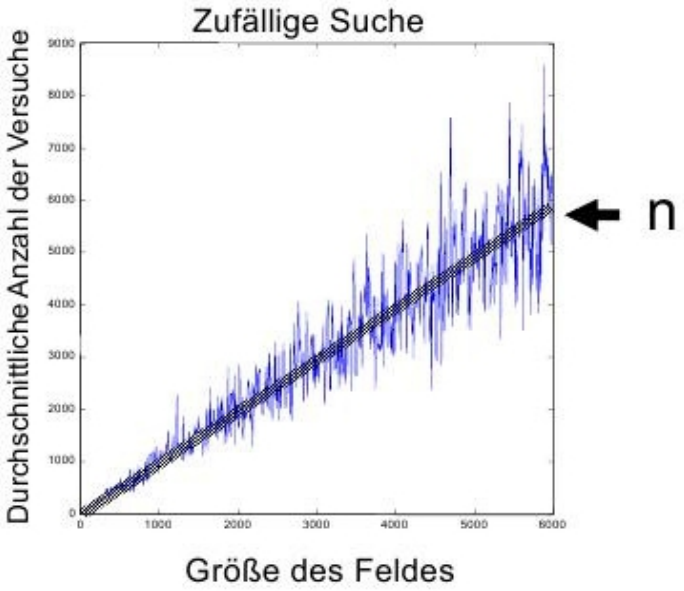
Beispiel: Sequentielle Suche

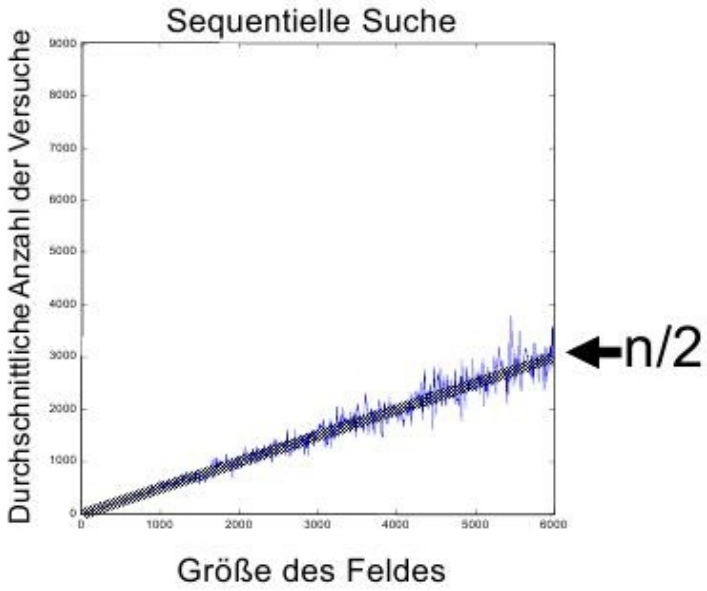
- Mit Hilfe der **`break`**-Anweisung kann man eine Schleife **vorzeitig beenden**
- Die **`break`**-Anweisung bezieht sich immer nur auf die innerste **`for`**- oder **`while`**-Schleife, die den Befehl umgibt. Sie springt hinter das Ende dieser Schleife



Analyse des Laufzeitverhaltens

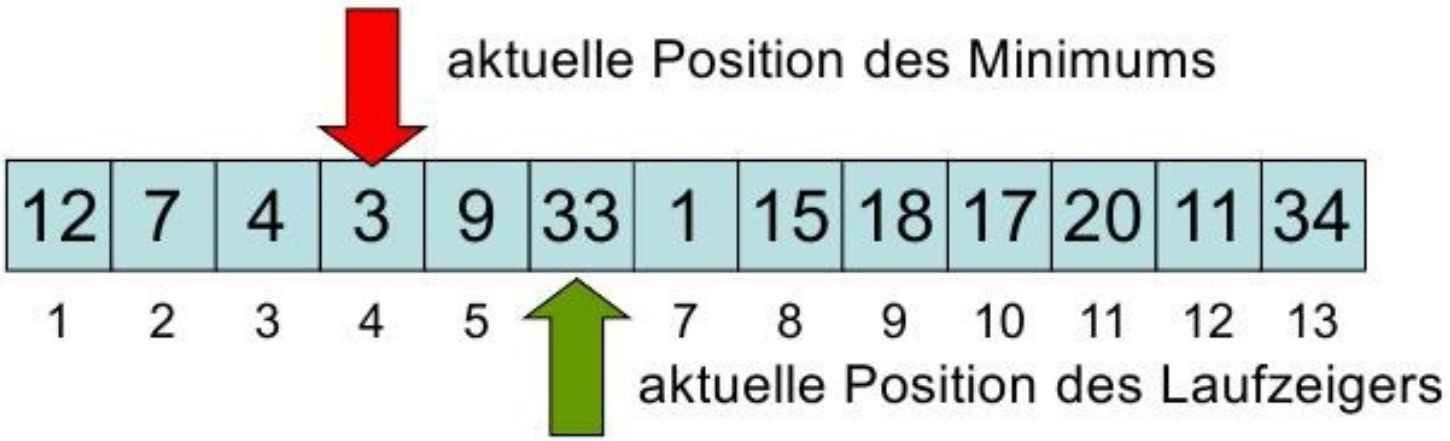
- Je 30 mal eine Zufallszahl in einem Intervall 1:n finden (n=1,2,3...)
- Der Mittelwert der 30 Versuche wird gebildet





Minimumssuche

- Gesucht wird die **Position** des **Minimums** in einer unsortierten Liste
- Im Gegensatz zum einfachen Suchen:
  - ergibt sich erst **während** der Suche, welcher Eintrag überhaupt gesucht ist
  - kann das Minimum **immer** gefunden werden
- Für die Minimumssuche werden alle Werte **der Reihe nach** besucht. Der bisher **kleinste** Wert wird **markiert**



Beispiel: Minimumssuche

```
function pos = minPos(x)
% Position des Minimums finden
n=length(x);          % Anzahl der Werte
pos=1;                % Bisher kleinstes Element
for i = 2:n           % Alle anderen Werte besuchen
   if x(i)<x(pos)     % Neues Minimum gefunden
      pos = i ;
end
```