Universität zu Köln

Universität zu Köln

# Informatik I

Vorlesung 1 – Teil 2 (Grundbegriffe und Lernziele)

# Was ist Informatik?

**Aus Positionspapier der Gesellschaft für Informatik e.V.**

- Informatik ist die Disziplin der automatischen Verarbeitung von Information
- Der Name ist eine Kurzform für die Kombination aus Information und Automatik
- Informatik ist Ingenieurs-, Grundlagen- und Systemwissenschaft
- Informatik ist eine Querschnittsdisziplin, die alle Lebens- und Wissenschaftsbereiche berührt

Quelle: https://gi.de/fileadmin/GI/Hauptseite/Themen/was-ist-informatik-kurz.pdf

# Was ist Informatik?

## Warum finde ich persönlich Informatik spannend?

- Informatik verlangt eine Kombination aus technischem Verständnis, mathematischen Methoden und Anwendungsdisziplinen
- Extrem dynamische Entwicklung
- Informatik beschäftigt sich mit fundamentalen Fragestellungen:
  - P vs. NP: Ist es einfacher einen Beweis nachzuvollziehen als einen Beweis zu finden?
  - Eingebettete Systeme: Wie können Autos automatisch fahren?
  - Algorithmische Spieltheorie: Gibt es Situation, in denen Marktgleichgewichte nicht erreicht werden?
  - Maschinelles Lernen/KI: Können Computer denken?
  - Quantum-Computing: Wie sehen die Rechner der Zukunft aus?

# Was sind Algorithmen?

## Informal

- Ein *Algorithmus* ist eine wohldefinierte Handlungsvorschrift, die einen Wert oder eine Menge von Werten als Eingabe erhält und als Ausgabe einen Wert oder eine Menge von Werten liefert
- Ein Algorithmus ist damit eine (endliche) Sequenz von Rechenschritten, die eine Eingabe in eine Ausgabe umwandelt
- Quelle: T. Cormen, C. Leisserson, R. Rivest, C. Stein. Introduction to Algorithms. Second Edition. MIT Press.

## Ziel dieser Vorlesung

- Grundlegendes Verständnis der Entwicklung, Bewertung und Analyse von Algorithmen

# Lernziele

**Die wichtigsten Lernziele**

- Methoden zur Entwicklung von Algorithmen für neue Problemstellungen
- Bewertung der Qualität von Algorithmen:
  - Korrektheit
  - Laufzeit
- Lernen grundlegender Algorithmen und Datenstrukturen

# Lernziele
# - Bewertung von Algorithmen

## Beispiel

- Berechnung der Fibonacci Zahlen
- Die erste Fibonacci-Zahl ist 1
- Die zweite Fibonacci-Zahl ist ebenfalls 1
- Die jeweils nächste Fibonacci-Zahl ergibt sich durch Addition der beiden vorherigen Zahlen

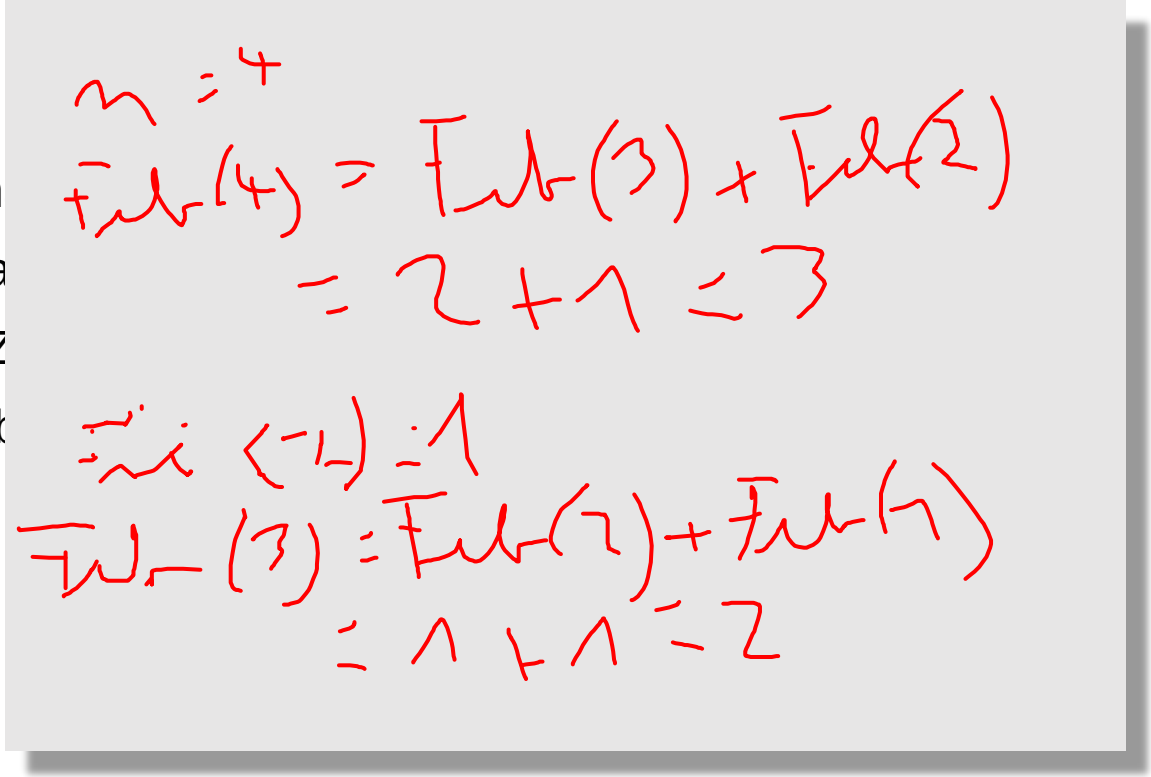
## Als Formel

- Fib(1)=1
- Fib(2)=1
- Für jedes n>2: Fib(n) = Fib(n-1) + Fib(n-2)

# Lernziele
# - Bewertung von Algorithmen

## Beispiel

- Berechnung der Fibon
- Die erste Fibonacci-Za
- Die zweite Fibonacci-Z
- Die jeweils nächste Fib
  vorherigen Zahlen




## Als Formel

- Fib(1)=1
- Fib(2)=1
- Für jedes n>2: Fib(n) = Fib(n-1) + Fib(n-2)

# Lernziele
# - Bewertung von Algorithmen

```
Fib1(n)
1.  F = new array[1…n]
2.  F[1] = 1
3.  F[2] = 1
4.  for i=3 to n do
5.     F[i] = F[i-1] + F[i-2]
6.  return F[n]
```

```
Fib2(n)
1.  if n=1 then return 1
2.  if n=2 then return 1
3.  return Fib2(n-1) + Fib2(n-2)
```

## Aufgabe

- Sie haben die Auswahl zwischen den beiden Algorithmen auf der linken Seite
- Welche verwenden Sie und warum?

# Lernziele
# - Bewertung von Algorithmen

```
Fib1(n)
1.  F = new array[1…n]
2.  F[1] = 1
3.  F[2] = 1
4.  for i=3 to n do
5.     F[i] = F[i-1] + F[i-2]
6.  return F[n]
```

```
Fib2(n)
1.  if n=1 then return 1
2.  if n=2 then return 1
3.  return Fib2(n-1) + Fib2(n-2)
```

## Aufgabe

- Sie haben die Auswahl zwischen den beiden Algorithmen auf der linken Seite
- Welche verwenden Sie und warum?

## Lösung

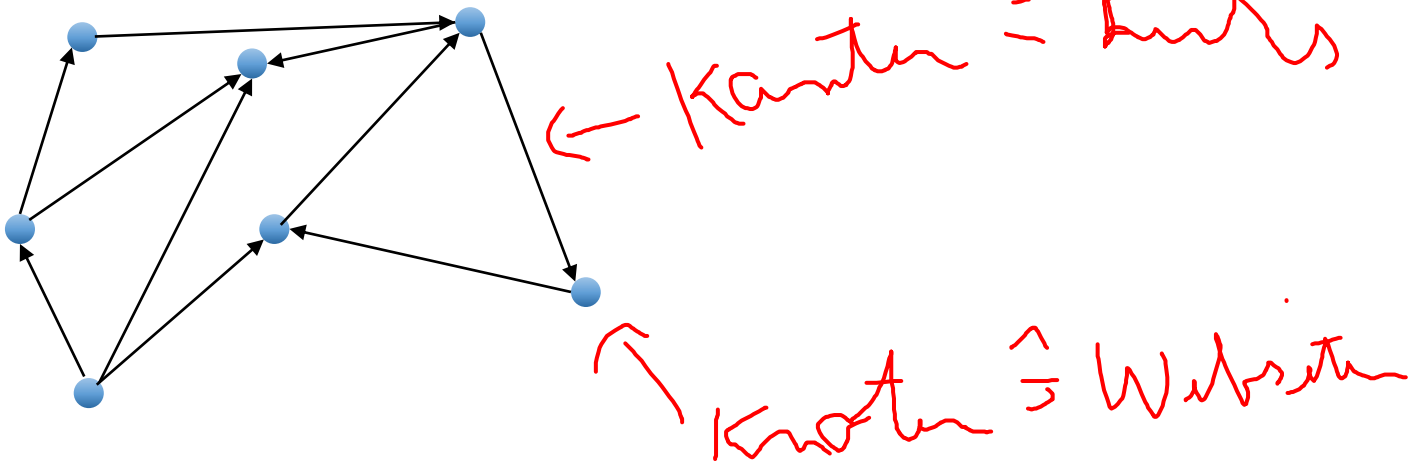
- Der erst Algorithmus ist besser, weil er viel schneller ist

Fib(50) = 12 586 269 025

50

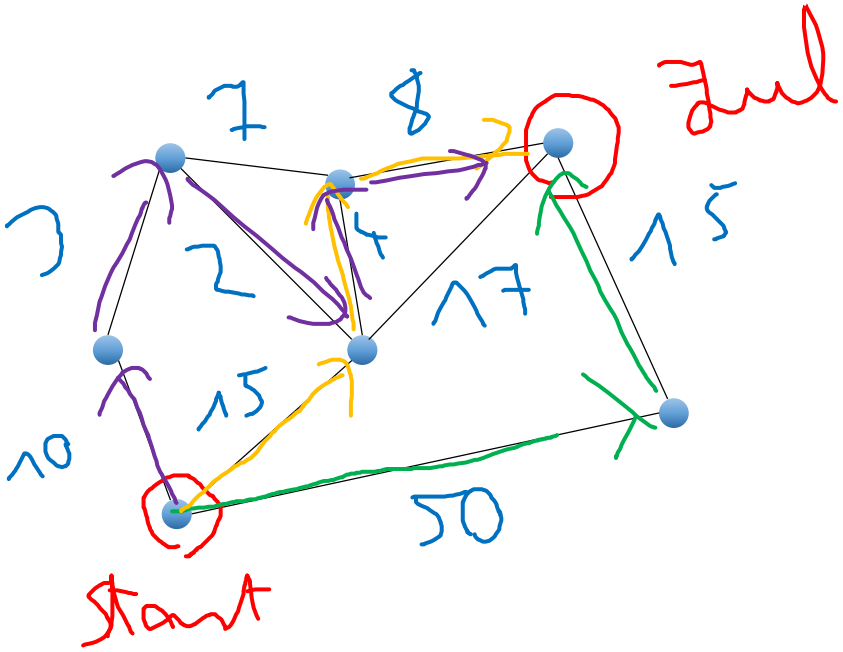
# Lernziele
# - Grundlegende Algorithmen und Datenstrukturen

- Graphen und Graphalgorithmen



# Lernziele
# - Grundlegende Algorithmen und Datenstrukturen

- Graphen und Graphalgorithmen




50 + 15 = 65

15 + 4 + 8 = 27

10 + 3 + 2 + 4 + 6 = 27

# Lernziele
# - Grundlegene Algorithmen und Datenstrukturen

**Algorithmen, die wir im Laufe der Vorlesung kennenlernen**

- Sortierverfahren
- Matrixmultiplikation
- Geometrische Algorithmen
- Graphalgorithmen
  - Graphtraversierung
  - Kürzeste Wege
  - Minimale Spannbäume
- Approximationsalgorithmen
- Datenstromalgorithmen

# Lernziele
# - Grundlegene Algorithmen und Datenstrukturen

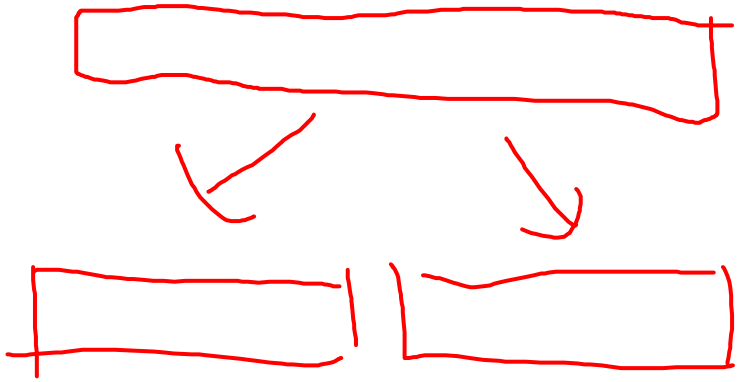
**Datenstrukturen, die wir im Laufe der Vorlesung kennenlernen**

- Felder
- Listen
- Heaps
- Balanzierte Suchbäume
- Hashing
- Graphen
- Union-Find Datenstruktur

# Lernziele
# - Methoden zur Algorithmenentwicklung

## Methoden

- Teile & Herrsche
- Dynamische Programmierung
- Gierige Algorithmen



## Wichtige Herangehensweise

- Rekursion

Universität zu Köln

# Informatik I

## Vorlesung 1 – Teil 2 (Erste Überlegungen zum Algorithmenentwurf)

# Rekursion

Rückwärtszähler(n)
1. **output** << n
2. Rückwärtszähler(n-1)

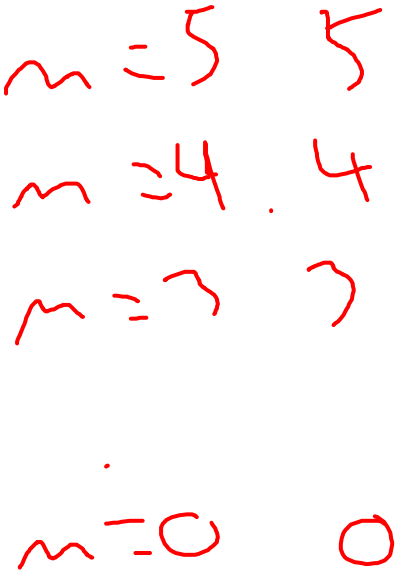
## Was ist Rekursion?

- Rekursion bezeichnet den Selbstaufruf von Algorithmen

n = 5. 5, 4, 3, 2, 1, 0, -1, ..

# Rekursion

Rückwärtszähler(n)
1. **output** << n
2. **if** n=0 **then return**
3. Rückwärtszähler(n-1)




## Was ist Rekursion?

- Rekursion bezeichnet den Selbstaufruf von Algorithmen

# Rekursion

## Rekursion in der Algorithmenentwicklung

- Rekursion führt die Lösung eines Problems auf die Lösung eines einfacheren (typischerweise kleineren) Problems zurück
- Es gibt ein grundlegendes Problem, dass einfach ohne Rekursion gelöst werden kann (Rekursionsabbruch)

# Weitere Vorgehensweise

**Konzepte für die Algorithmenanalyse am Beispiel „Sortieren von Zahlen“**

- Entwicklung eines Sortieralgorithmus mit Hilfe von Rekursion
- Informatische Grundlagen
  - Pseudocode
  - Rechenmodell
  - Laufzeitanalyse
- Mathematische Grundlagen
  - Induktion und Korrektheitsbeweise
  - Landau Notation

# Berechnungsproblem

## Berechnungsproblem

- Beschreibt eindeutig eine gewünschte Relation zwischen Eingabe und Ausgabe
- Ein Algorithmus kann als Lösungsverfahren für ein Berechnungsproblem angesehen werden

# Beispiel: Sortieren

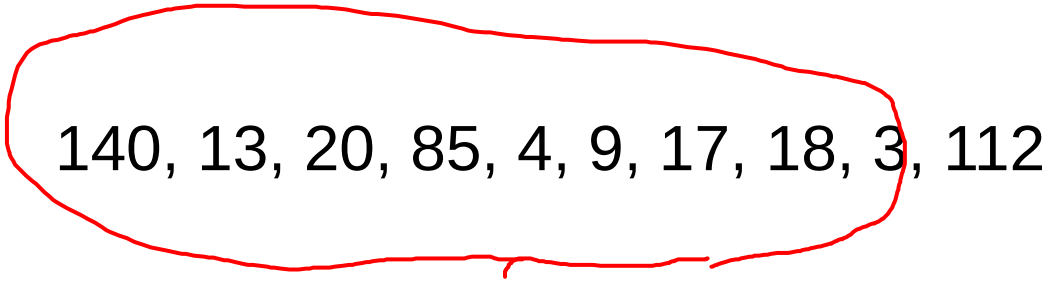
**Berechnungsproblem: Sortieren**

- Eingabe: Folge von n Zahlen: $(x_1,\ldots, x_n)$
- Ausgabe: Permutation $(x_1',\ldots,x_n')$ von $(x_1,\ldots, x_n)$ mit $x_1' \leq x_2' \leq \ldots \leq x_n'$

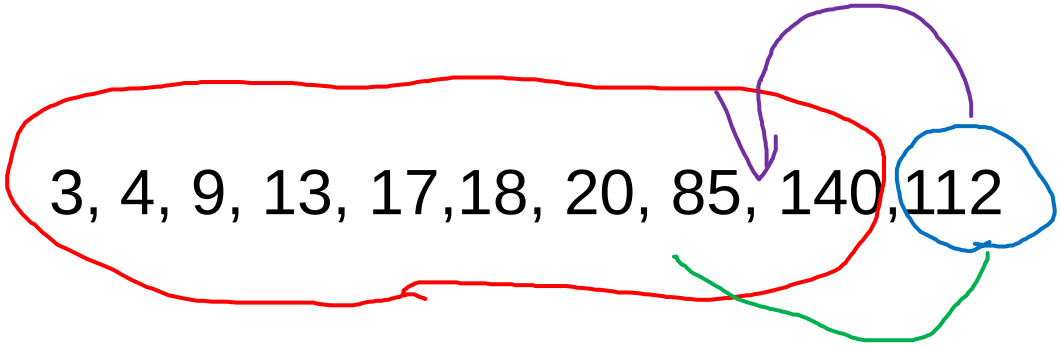
**Beispiel:**

- Eingabe: 15, 7, 3, 18, 8, 4
- Ausgabe: 3, 4, 7, 8, 15, 18

# Entwicklung eines Sortieralgorithmus



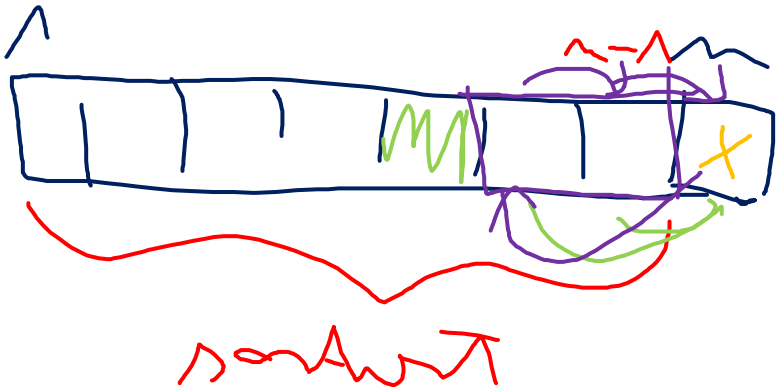
# Entwicklung eines Sortieralgorithmus



# Entwicklung eines Sortieralgorithmus

OurSort(A, n)  A ist Feld (Array) mit n Zahlen

1. x=A[n]
2. ~~Magic~~Sort(A,n-1)
3. Füge x an die korrekte Stelle in A ein



# Entwicklung eines Sortieralgorithmus

OurSort(A, n)  A ist Feld (Array) mit n Zahlen

1. x=A[n]
2. OurSort(A,n-1)
3. Füge x an die korrekte Stelle in A ein

# Entwicklung eines Sortieralgorithmus

OurSort(A, n)  A ist Feld (Array) mit n Zahlen

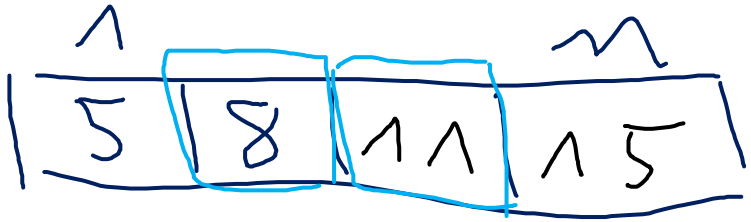
1. **if** n=1 **then return**
2. x=A[n]
2. OurSort(A,n-1)
3. Füge x an die korrekte Stelle in A ein

# Entwicklung eines Sortieralgorithmus

OurSort(A, n)

A ist Feld (Array) mit n Zahlen

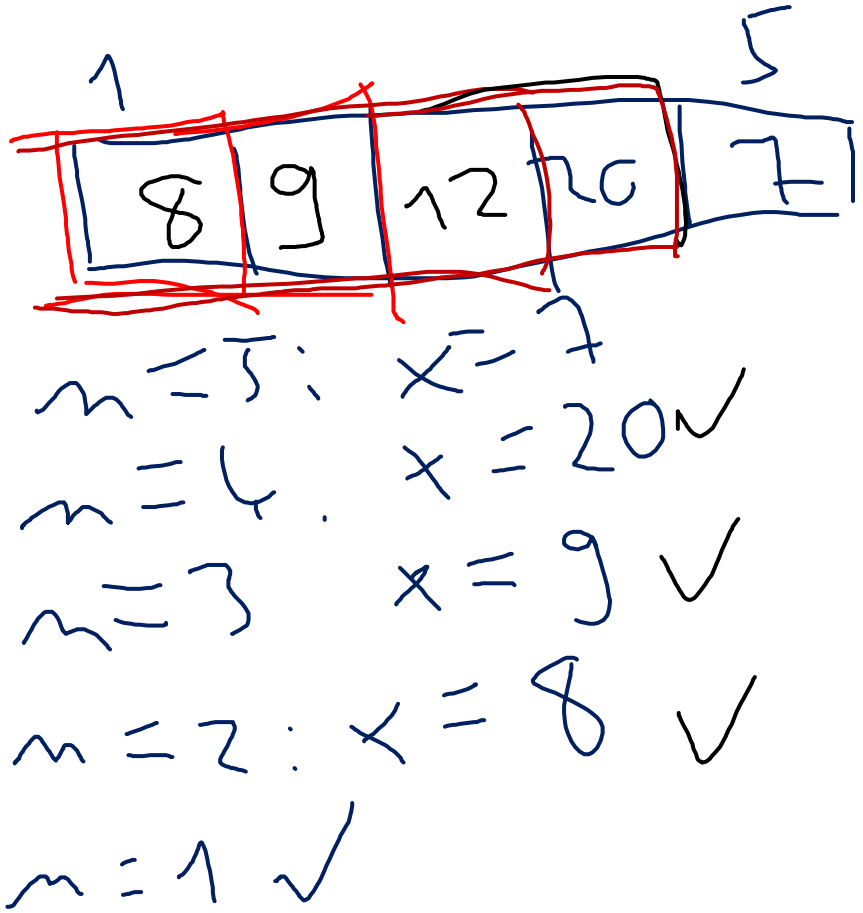
1. **if** n=1 **then return**
2. x=A[n]
2. OurSort(A,n-1)
3. j=n-1
4. **while** j>0 and A[j]>x **do**
5. A[j+1] = A[j]
6. j = j-1
7. A[j+1]=x



# Entwicklung eines Sortieralgorithmus

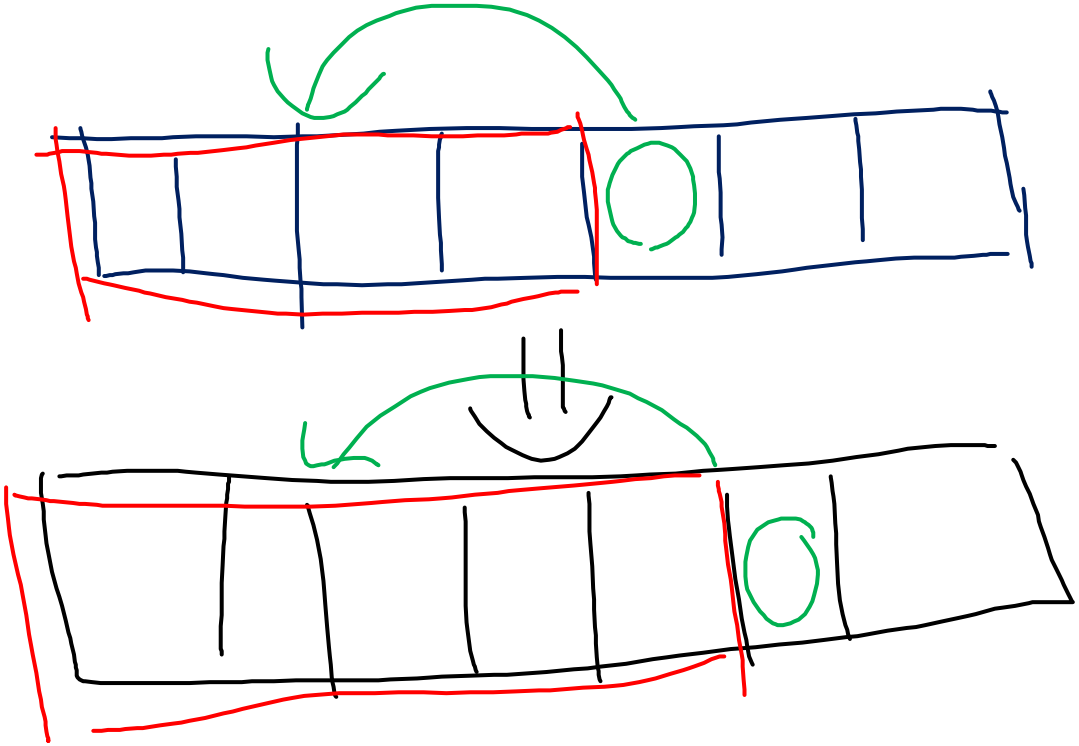
OurSort(A, n)

1. **if** n=1 **then return**
2. x=A[n]
2. OurSort(A,n-1)
3. j=n-1
4. **while** j>0 and A[j]>x **do**
5. A[j+1] = A[j]
6. j = j-1
7. A[j+1]=x

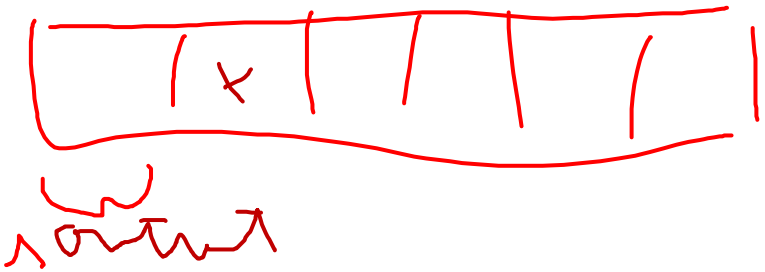


# InsertionSort

**Idee**

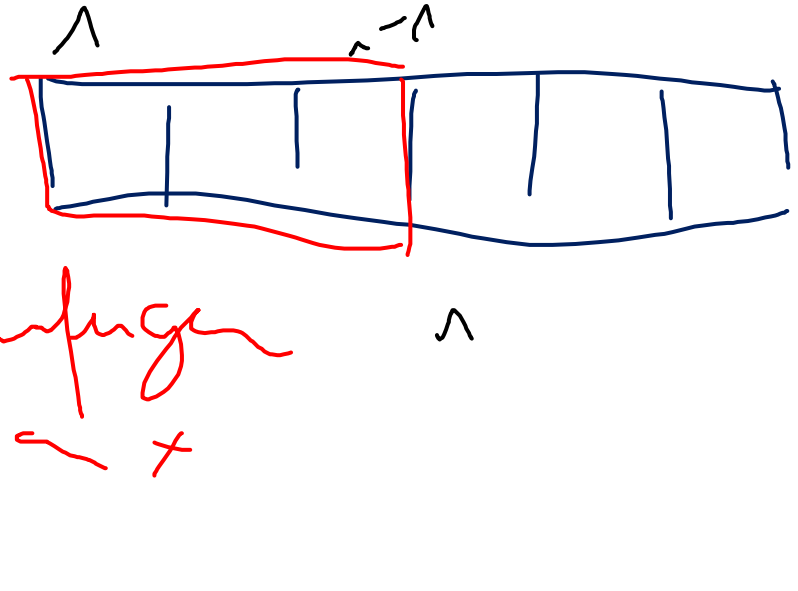


# InsertionSort



InsertionSort(A, n)  Feld A der Länge n wird übergeben

1. **for** i=2 **to** n **do**
2. x = A[i]
3. j = i -1
4. **while** j>0 and A[j]>x **do**
5. A[j+1] = A[j]
6. j = j-1
7. A[j+1]=x



# Zusammenfassung

**Entwicklung von Algorithmen mit Hilfe von Rekursion**

- Rekursion
- Entwicklung eines Sortieralgorithmus mit Hilfe von Rekursion
- Umwandlung in einen iterativen Algorithmus (InsertionSort)