

Programmieren und Problemlösen

Daten einlesen und Sortieren 1

Dennis Komm



Listen

Weiterführende Konzepte

2-dimensionale Listen

Bislang beinhalten Listen Zahlen oder Zeichen

2-dimensionale Listen

Bislang beinhalten Listen Zahlen oder Zeichen

- Liste kann auch Listen beinhalten
- Solche **2-dimensionalen Listen** speichern z. B. Tabellen und Matrizen

2-dimensionale Listen

Bislang beinhalten Listen Zahlen oder Zeichen

- Liste kann auch Listen beinhalten
- Solche **2-dimensionalen Listen** speichern z. B. Tabellen und Matrizen

$$M = \begin{pmatrix} 2 & 0 & 3 & 0 & 6 \\ 3 & 9 & 5 & 1 & 1 \\ 0 & 0 & 7 & 2 & 7 \\ 3 & 9 & 5 & 8 & 0 \\ 8 & 2 & 0 & 3 & 2 \\ 1 & 6 & 5 & 9 & 6 \end{pmatrix}$$

```
M = [ [2, 0, 3, 0, 6],  
      [3, 9, 5, 1, 1],  
      [0, 0, 7, 2, 7],  
      [3, 9, 5, 8, 0],  
      [8, 2, 0, 3, 2],  
      [1, 6, 5, 9, 6] ]
```

2-dimensionale Listen

Bislang beinhalten Listen Zahlen oder Zeichen

- Liste kann auch Listen beinhalten
- Solche **2-dimensionalen Listen** speichern z. B. Tabellen und Matrizen

$$M = \begin{pmatrix} 2 & 0 & 3 & 0 & 6 \\ 3 & 9 & 5 & 1 & 1 \\ 0 & 0 & 7 & 2 & 7 \\ 3 & 9 & 5 & 8 & 0 \\ 8 & 2 & 0 & 3 & 2 \\ 1 & 6 & 5 & 9 & 6 \end{pmatrix}$$

```
M = [ [2, 0, 3, 0, 6],  
      [3, 9, 5, 1, 1],  
      [0, 0, 7, 2, 7],  
      [3, 9, 5, 8, 0],  
      [8, 2, 0, 3, 2],  
      [1, 6, 5, 9, 6] ]
```

- Zugriff auf *i*-te Zeile und *j*-te Spalte mit `M[i][j]`

Listen

Daten einlesen und in Listen speichern

Beispiel: Matrix in Datei gegeben

- Inhalt der Datei ist ein Text
- Matrix zeilenweise abgespeichert
- Einträge in jeder Zeile mit Kommata getrennt
- Einträge sollen als Zahlen interpretiert werden

Beispiel: Matrix in Datei gegeben

- Inhalt der Datei ist ein Text
- Matrix zeilenweise abgespeichert
- Einträge in jeder Zeile mit Kommata getrennt
- Einträge sollen als Zahlen interpretiert werden

Drei Schritte

1. Datei Zeile für Zeile einlesen
2. Einträge aus den Zeilen extrahieren (Trennsymbol: Komma)
3. Jeden Eintrag in Zahl konvertieren

Daten einlesen

1. Datei Zeile für Zeile einlesen

```
with open("daten.txt") as file:
```

1. Datei Zeile für Zeile einlesen

```
with open("daten.txt") as file:
```

- Datei `daten.txt` ist für den nachfolgenden Anweisungsblock geöffnet
- Zugreifbar unter dem Namen `file`

1. Datei Zeile für Zeile einlesen

```
with open("daten.txt") as file:  
    lines = file.read().splitlines()
```

- Datei `daten.txt` ist für den nachfolgenden Anweisungsblock geöffnet
- Zugreifbar unter dem Namen `file`
- `lines = file.read()` speichert den gesamten Text aus `daten.txt` in der Variablen `lines`

1. Datei Zeile für Zeile einlesen

```
with open("daten.txt") as file:  
    lines = file.read().splitlines()
```

- Datei `daten.txt` ist für den nachfolgenden Anweisungsblock geöffnet
- Zugreifbar unter dem Namen `file`
- `lines = file.read()` speichert den gesamten Text aus `daten.txt` in der Variablen `lines`
- `lines = file.read().splitlines()` speichert die einzelnen Zeilen von `daten.txt` als Einträge der Liste `lines`

Daten einlesen: Beispiel

1. Datei Zeile für Zeile einlesen

```
with open("daten.txt") as file:  
    lines = file.read().splitlines()
```

daten.txt

```
2, 0, 3, 0, 6  
3, 9, 5, 1, 1  
0, 0, 7, 2, 7  
3, 9, 5, 8, 0  
8, 2, 0, 3, 2  
1, 6, 5, 9, 6
```

```
lines = [ "2, 0, 3, 0, 6",  
          "3, 9, 5, 1, 1",  
          "0, 0, 7, 2, 7",  
          "3, 9, 5, 8, 0",  
          "8, 2, 0, 3, 2",  
          "1, 6, 5, 9, 6" ]
```

2. Einträge aus erster Zeile extrahieren (Trennsymbol: Komma)

```
tmp = lines[0].split(",")
```

2. Einträge aus erster Zeile extrahieren (Trennsymbol: Komma)

```
tmp = lines[0].split(",")
```

```
lines = [ "2, 0, 3, 0, 6",  
          "3, 9, 5, 1, 1",  
          "0, 0, 7, 2, 7",  
          "3, 9, 5, 8, 0",  
          "8, 2, 0, 3, 2",  
          "1, 6, 5, 9, 6" ]
```

```
tmp = ["2", "0", "3", "0", "6"]
```


3. Jeden Eintrag in Zahl konvertieren

```
data = [0] * len(tmp)
for i in range(0, len(tmp)):
    data[i] = int(tmp[i])
```

3. Jeden Eintrag in Zahl konvertieren

```
data = [0] * len(tmp)
for i in range(0, len(tmp)):
    data[i] = int(tmp[i])
```

```
tmp = ["2", "0", "3", "0", "6"]
```

```
data = [2, 0, 3, 0, 6]
```

Daten einlesen: Zusammenfassung

```
def readfile(filename):  
  
    # Datei Zeile für Zeile einlesen  
    with open(filename) as file:  
        lines = file.read().splitlines()  
  
    # Einträge aus erster Zeile extrahieren (Trennsymbol: Komma)  
    tmp = lines[0].split(",")  
  
    # Jeden Eintrag in Zahl konvertieren  
    data = [0] * len(tmp)  
    for i in range(0, len(tmp)):  
        data[i] = int(tmp[i])  
  
    return data
```

Aufgabe – Daten einlesen

Erweitern Sie die Funktion, sodass

- alle Zeilen der Datei gelesen und konvertiert werden
- der Inhalt in einer 2-dimensionalen Liste gespeichert wird

```
def readfile(filename):  
    with open(filename) as file:  
        lines = file.read().splitlines()  
        tmp = lines[0].split(",")  
        data = [0] * len(tmp)  
        for i in range(0, len(tmp)):  
            data[i] = int(tmp[i])  
    return data
```



Daten einlesen

```
def readfile2(filename):
    # Datei Zeile für Zeile einlesen
    with open(filename) as file:
        lines = file.read().splitlines()
    data = []

    # Alle Zeilen nacheinander verarbeiten
    for i in range(0, len(lines)):
        tmp = lines[i].split(",")
        dataline = [0] * len(tmp)
        for j in range(0, len(tmp)):
            dataline[j] = int(tmp[j])
        data.append(dataline)
    return data
```

Daten einlesen

```
def readfile2(filename):
    # Datei Zeile für Zeile einlesen
    with open(filename) as file:
        lines = file.read().splitlines()
    data = []

    # Alle Zeilen nacheinander verarbeiten
    for i in range(0, len(lines)):
        tmp = lines[i].split(",")
        dataline = [0] * len(tmp)
        for j in range(0, len(tmp)):
            dataline[j] = int(tmp[j])
        data.append(dataline)
    return data
```

Oft sind Daten als solche `csv`-Dateien (Comma Separated Values) gegeben

Sortieren 1

Sortieren und Suchen

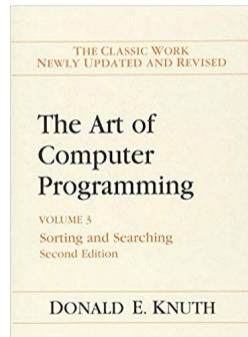
Sortieren und Suchen

Daten suchen und **sortieren** sind zwei der grundlegendsten Aufgaben von Informatikerinnen und Informatikern

Sortieren und Suchen

Daten suchen und **sortieren** sind zwei der grundlegendsten Aufgaben von Informatikerinnen und Informatikern

Standardwerk dreht sich nur um diese Themen

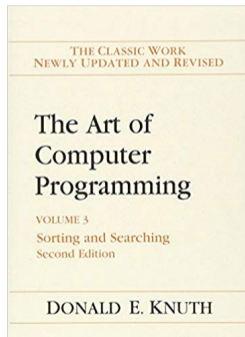


Sortieren und Suchen

Daten suchen und **sortieren** sind zwei der grundlegendsten Aufgaben von Informatikerinnen und Informatikern

Standardwerk dreht sich nur um diese Themen

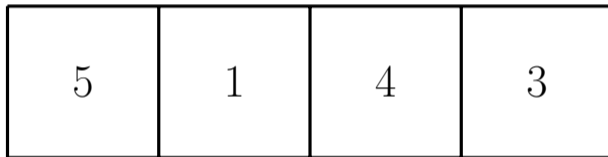
- Gegeben seien n positive ganze Zahlen
- Genauer **unsortierte** Liste `data` mit $n = \text{len}(\text{data})$
- Wir betrachten n als Eingabelänge
- Zahlen können mehrfach auftauchen
- **Sortiere** mit möglichst kleiner Zeitkomplexität



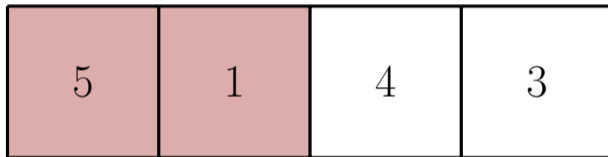
Sortieren 1

Bubblesort

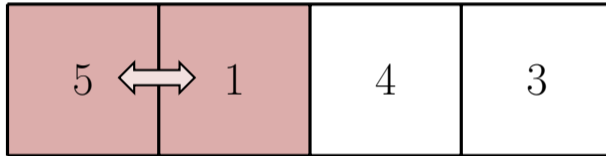
Bubblesort



Bubblesort



Bubblesort



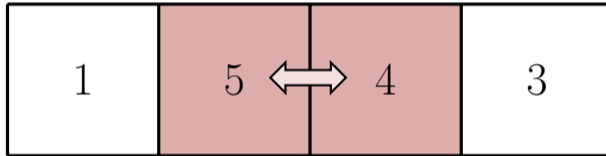
Bubblesort



Bubblesort



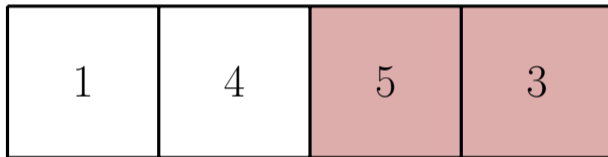
Bubblesort



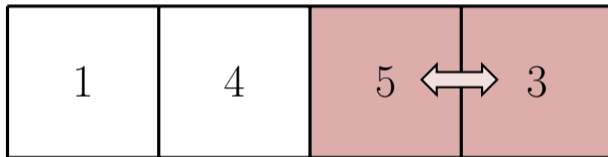
Bubblesort



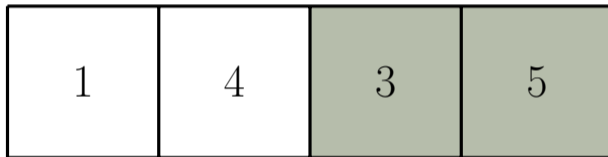
Bubblesort



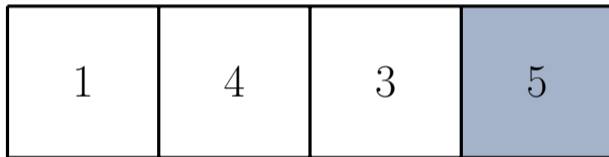
Bubblesort



Bubblesort



Bubblesort



Bubblesort



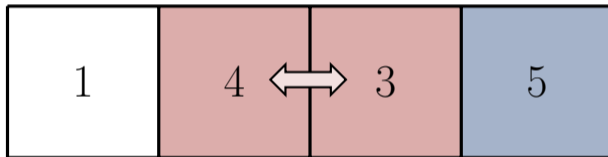
Bubblesort



Bubblesort



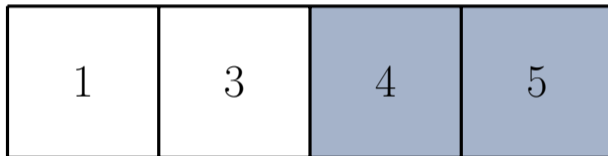
Bubblesort



Bubblesort



Bubblesort



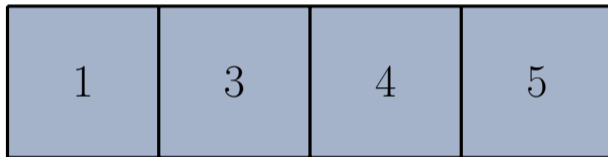
Bubblesort



Bubblesort



Bubblesort



Bubblesort

Idee

Sortieren durch wiederholtes Finden des Maximums

Bubblesort

Idee

Sortieren durch wiederholtes Finden des Maximums

Ziel

Liste `data` mit n Elementen sortieren, also Bereich $0, \dots, n - 1$

Bubblesort

Idee

Sortieren durch wiederholtes Finden des Maximums

Ziel

Liste `data` mit n Elementen sortieren, also Bereich $0, \dots, n - 1$

- Finde Maximum und schiebe es an die letzte Position
- Vergleiche hierzu iterativ alle benachbarten Elemente

Bubblesort

Idee

Sortieren durch wiederholtes Finden des Maximums

Ziel

Liste `data` mit n Elementen sortieren, also Bereich $0, \dots, n - 1$

- Finde Maximum und schiebe es an die letzte Position
- Vergleiche hierzu iterativ alle benachbarten Elemente
- Maximum wandert wie Luftblase durch Liste an letzte Position

Bubblesort

Idee

Sortieren durch wiederholtes Finden des Maximums

Ziel

Liste `data` mit n Elementen sortieren, also Bereich $0, \dots, n - 1$

- Finde Maximum und schiebe es an die letzte Position
- Vergleiche hierzu iterativ alle benachbarten Elemente
- Maximum wandert wie Luftblase durch Liste an letzte Position
- Wiederholen mit Bereich $0, \dots, n - 2$

Bubblesort

Idee

Sortieren durch wiederholtes Finden des Maximums

Ziel

Liste `data` mit n Elementen sortieren, also Bereich $0, \dots, n - 1$

- Finde Maximum und schiebe es an die letzte Position
- Vergleiche hierzu iterativ alle benachbarten Elemente
- Maximum wandert wie Luftblase durch Liste an letzte Position
- Wiederholen mit Bereich $0, \dots, n - 2$
- Fahre fort, bis `data` sortiert ist

Aufgabe – Eine Bubble-Sequenz

Implementieren Sie eine Bubble-Sequenz

- Laufe einmal durch data
- Vergleiche jeweils benachbarte Paare
- Vertausche, wenn das erste Element grösser ist
- Maximum „bubblet“ nach rechts



Eine Bubble-Sequenz

Eine Bubble-Sequenz in Python

```
data = [6, 22, 61, 1, 89, 31, 9, 10, 76]
n = len(data)

for i in range(0, n-1):
    if data[i] > data[i+1]:
        tmp = data[i]
        data[i] = data[i+1]
        data[i+1] = tmp
```

Aufgabe – Bubblesort

Implementieren Sie den vollständigen Algorithmus

- Iteriere Bubble-Sequenzen
- Nach k -ter Sequenz sind die letzten k Elemente von `data` sortiert
- Bubble-Sequenzen werden mit jeder Iteration kürzer
- Verwende hierzu eine äussere Schleife



Bubblesort

```
def bubblesort(data):  
    n = len(data)  
    for d in range(n, 1, -1):  
        for i in range(0, d-1):  
            if data[i] > data[i+1]:  
                tmp = data[i]  
                data[i] = data[i+1]  
                data[i+1] = tmp  
    return data  
  
print(bubblesort([6, 22, 61, 1, 89, 31, 9, 10, 76]))
```

Sortieren 1

Minsort

Minsort

Idee

Sortieren durch wiederholtes Finden des Minimums

Minsort

Idee

Sortieren durch wiederholtes Finden des Minimums

- Anders als bei Bubblesort werden nicht benachbarte Elemente verglichen
- Aktuelles Minimum wird gespeichert
- Jedes Element wird mit diesem verglichen
- Falls es kleiner ist, werden beide miteinander vertauscht

Idee

Sortieren durch wiederholtes Finden des Minimums

- Anders als bei Bubblesort werden nicht benachbarte Elemente verglichen
- Aktuelles Minimum wird gespeichert
- Jedes Element wird mit diesem verglichen
- Falls es kleiner ist, werden beide miteinander vertauscht
- Nach einer Iteration wird Minimum an (aktuellen) Anfang kopiert
- Fahre fort, bis data sortiert ist

Minsort

```
def minsort(data):
    n = len(data)
    for current in range(0, n-1):
        minimum = data[current]
        for i in range(current+1, n):
            if data[i] < minimum:
                tmp = data[i]
                data[i] = minimum
                minimum = tmp
        data[current] = minimum
    return data

print(minsort([6, 22, 61, 1, 89, 31, 9, 10, 76]))
```

Sortieren 1

Zeitkomplexität von Bubblesort und Minsort

Zeitkomplexität von Bubblesort und Minsort

Zähle die Vergleiche von jeweils zwei Zahlen

Zeitkomplexität von Bubblesort und Minsort

Zähle die Vergleiche von jeweils zwei Zahlen

- $n - 1$ Vergleiche, um das Maximum zu finden

Zeitkomplexität von Bubblesort und Minsort

Zähle die Vergleiche von jeweils zwei Zahlen

- $n - 1$ Vergleiche, um das Maximum zu finden
- $n - 2$ Vergleiche, um das zweitgrösste Element zu finden

Zeitkomplexität von Bubblesort und Minsort

Zähle die Vergleiche von jeweils zwei Zahlen

- $n - 1$ Vergleiche, um das Maximum zu finden
- $n - 2$ Vergleiche, um das zweitgrösste Element zu finden
- ...

Zeitkomplexität von Bubblesort und Minsort

Zähle die Vergleiche von jeweils zwei Zahlen

- $n - 1$ Vergleiche, um das Maximum zu finden
- $n - 2$ Vergleiche, um das zweitgrösste Element zu finden
- ...
- 1 Vergleich, um die beiden kleinsten Elemente zu sortieren

Zeitkomplexität von Bubblesort und Minsort

Zähle die Vergleiche von jeweils zwei Zahlen

- $n - 1$ Vergleiche, um das Maximum zu finden
 - $n - 2$ Vergleiche, um das zweitgrösste Element zu finden
 - ...
 - 1 Vergleich, um die beiden kleinsten Elemente zu sortieren
- ⇒ Insgesamt $\sum_{i=1}^{n-1} i = (n - 1) \cdot n / 2 = (n^2 - n) / 2$ Vergleiche
- ⇒ Quadratisch viele Vergleiche

Zeitkomplexität von Bubblesort und Minsort

Zähle die Vergleiche von jeweils zwei Zahlen

- $n - 1$ Vergleiche, um das Maximum zu finden
 - $n - 2$ Vergleiche, um das zweitgrösste Element zu finden
 - ...
 - 1 Vergleich, um die beiden kleinsten Elemente zu sortieren
- ⇒ Insgesamt $\sum_{i=1}^{n-1} i = (n - 1) \cdot n / 2 = (n^2 - n) / 2$ Vergleiche
- ⇒ Quadratisch viele Vergleiche

Die Laufzeit von Bubblesort ist in $\mathcal{O}(n^2)$

Zeitkomplexität von Bubblesort und Minsort

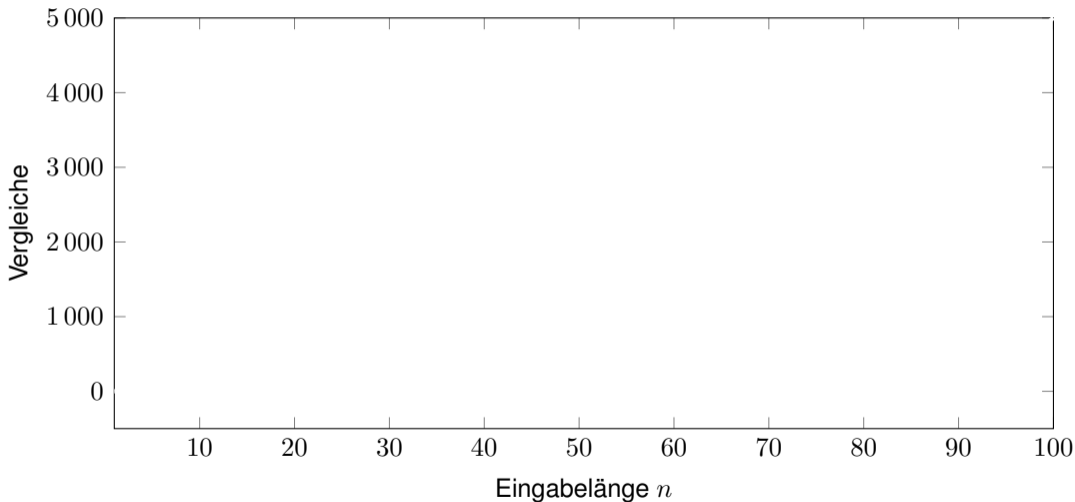
Zähle die Vergleiche von jeweils zwei Zahlen

- $n - 1$ Vergleiche, um das Maximum zu finden
 - $n - 2$ Vergleiche, um das zweitgrösste Element zu finden
 - ...
 - 1 Vergleich, um die beiden kleinsten Elemente zu sortieren
- ⇒ Insgesamt $\sum_{i=1}^{n-1} i = (n - 1) \cdot n/2 = (n^2 - n)/2$ Vergleiche
- ⇒ Quadratisch viele Vergleiche

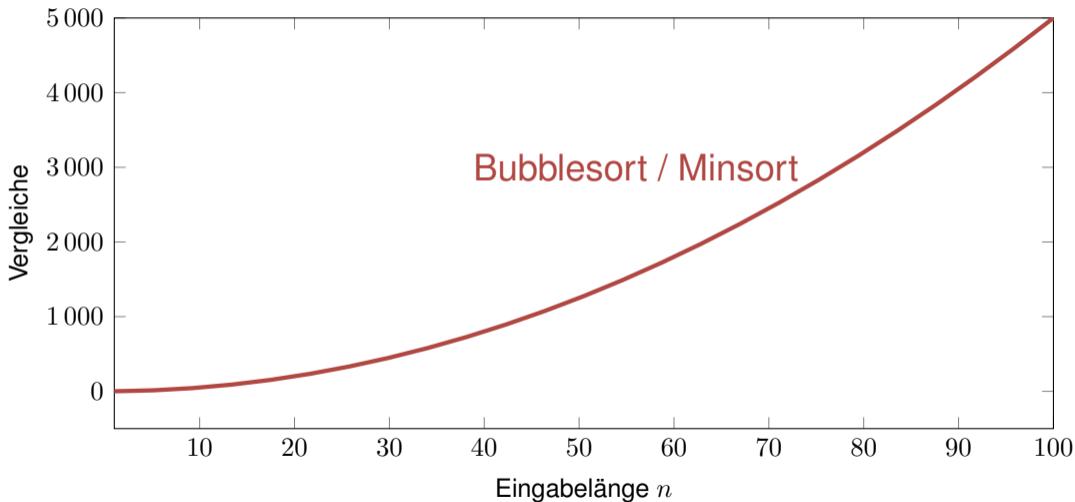
Die Laufzeit von Bubblesort ist in $\mathcal{O}(n^2)$

Die Laufzeit von Minsort ist (mit ähnlichen Argumenten) in $\mathcal{O}(n^2)$

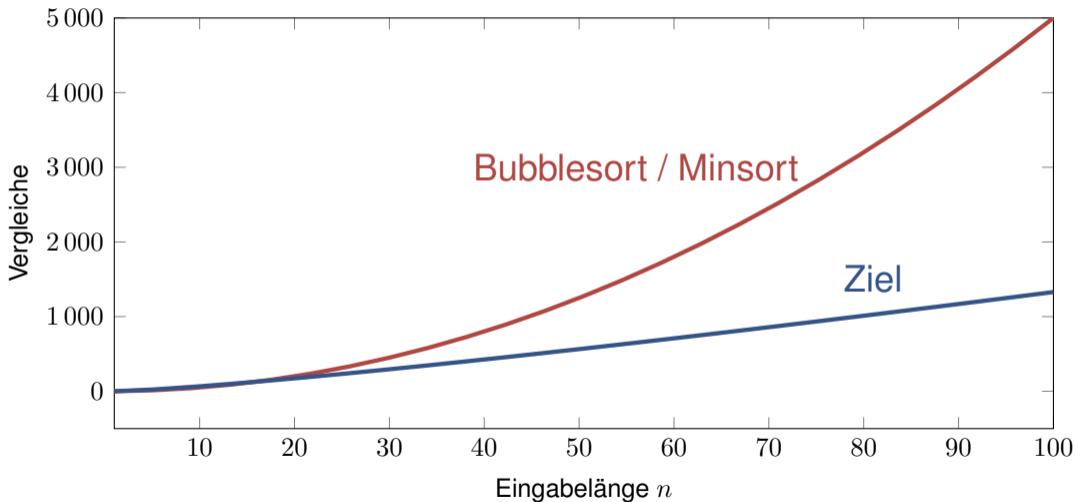
Zeitkomplexität von Bubblesort und Minsort



Zeitkomplexität von Bubblesort und Minsort



Zeitkomplexität von Bubblesort und Minsort



Danke für die
Aufmerksamkeit