

Algorithmen und Datenstrukturen

- Sortieralgorithmen -

Prof. Dr. Klaus Volbert

Wintersemester 2018/19
Regensburg, 18. Oktober 2018

- Einführung und Organisation
- Grundlagen
 - Begriffe
 - Algorithmus, Datentyp, Datenstruktur, Datenstruktur Stapel
 - Korrektheit und Komplexität
 - Totale Korrektheit, Iterationen, Rekursionen
 - RAM, Church'sche These, O-Notation (O , Ω , Θ)
 - Rekursionsgleichungen
 - Iterationsmethode, Substitutionsmethode, Master-Methode
 - Entwurfsmethode Divide & Conquer
- Sortieralgorithmen

- Standardproblem: Sortieren
 - Schnelles Finden von Informationen durch vorheriges Sortieren
 - Anfang der 70er: ca. 25 % der weltweit verbrauchten Rechenzeit wird fürs Sortieren verwendet (heute nicht mehr so)
 - Heutige Anwendungen: DNA-Analyse, Internet, Computergrafik, ...
- Sortierproblem
 - Eingabe: a_0, \dots, a_{n-1} (n Zahlen einer total geordneten Menge, $n \in \mathbb{N}$)
 - Ausgabe: $a_{\pi(0)}, \dots, a_{\pi(n-1)}$ mit $a_{\pi(0)} \leq \dots \leq a_{\pi(n-1)}$ und π ist Permutation, d.h. $\pi: \{0, \dots, n-1\} \rightarrow \{0, \dots, n-1\}$
- Beispiel
 - Eingabe: 31, 41, 59, 26, 41, 58
 - Ausgabe: 26, 31, 41, 41, 58, 59

i	0	1	2	3	4	5
$\pi(i)$	1	2	5	0	3	4

Überblick Sortieralgorithmen

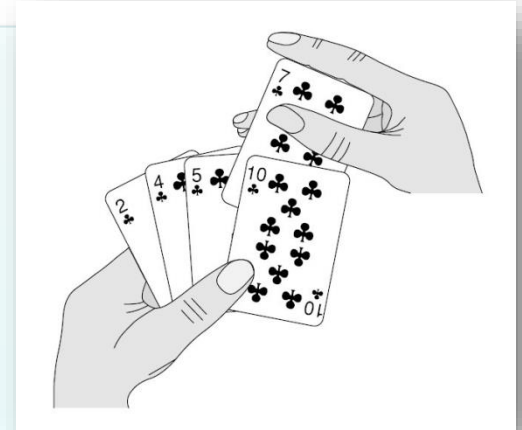
- Nichts ist besser verstanden als das Sortieren...
- Einfache Sortieralgorithmen
 - Sortieren durch Einfügen (Insertion Sort)
 - Sortieren durch Vertauschen (Bubble Sort)
 - Sortieren durch Auswählen (Selection Sort)
- Fortgeschrittene Sortieralgorithmen
 - Sortieren durch Divide & Conquer (Quicksort)
 - Sortieren durch Mischen (Merge Sort)
 - Sortieren durch Bäume (Heap Sort)
- Spezielle Sortieralgorithmen
 - Sortieren durch Zählen (Count Sort)
 - Sortieren durch Abbilden (Map Sort)
- Untere Schranke für vergleichsbasierte Sortieralgorithmen

Sortieren durch Einfügen (Insertion Sort)

```
void InsertionSort(int a[], int n)
{
    int i, j, key;

    for (j = 1; j < n; j++)
    {
        key = a[j];
        i = j-1;
        while (i >= 0 && a[i] > key)
        {
            a[i+1] = a[i];
            i = i-1;
        }

        a[i+1] = key;
    }
}
```



Beispiel Insertion Sort

34	45	12	34	23	18	38	17	43	51
<u>34</u>	<u>45</u>	12	34	23	18	38	17	43	51
<u>12</u>	<u>34</u>	<u>45</u>	34	23	18	38	17	43	51
<u>12</u>	<u>34</u>	<u>34</u>	<u>45</u>	23	18	38	17	43	51
<u>12</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>45</u>	18	38	17	43	51
<u>12</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>45</u>	38	17	43	51
<u>12</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>38</u>	<u>45</u>	17	43	51
<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>38</u>	<u>45</u>	43	51
<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>38</u>	<u>43</u>	<u>45</u>	51
<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>38</u>	<u>43</u>	<u>45</u>	<u>51</u>

Korrektheit InsertionSort I

- Behauptung:
 - Nach dem j .ten Schleifendurchlauf gilt (Schleifeninvariante):

$$a_0 \leq \dots \leq a_j$$

- IA: $j = 1$:
 - In der Schleife wird $\text{key} = a[j] = a[1]$ und $i = j - 1 = 0$
 - 1. Fall: $a[i] \leq \text{key}$, dann wird die while-Schleife nicht betreten, $a[j]$ bleibt $a[j]$ und es gilt: $a[0] = a[i] \leq a[j] = a[1]$
 - 2. Fall: $a[i] > \text{key}$, dann wird die while-Schleife betreten, und es werden die Inhalte von $a[0]$ und $a[1]$ getauscht.
Es gilt: $a[0] = a[j] \leq a[i] = a[1]$
- IV: Behauptung gilt für $j-1$

Korrektheit InsertionSort II

- IS: $j - 1 \rightarrow j$:
 - Nach IV gilt nach dem $(j - 1)$.ten Schleifendurchlauf (Schleifeninvariante):

$$a_0 \leq \dots \leq a_{j-1}$$
 - Im j .ten Schleifendurchlauf wird $\text{key} = a[j]$ (Sicherung aktueller Wert) und $i = j - 1$ (Index letztes Element in der bereits sortierten Folge)
 - In der while-Schleife wird nun mittels Iteration über die bereits sortierte Folge die Position gesucht, an der das neue Element eingeordnet werden muss
 - Je Iteration wird der Laufindex um 1 verringert und nicht ungültig
 - Da der aktuelle Wert gesichert ist, ist eine Tauschlücke entstanden, die genutzt wird, um die Werte bei Iteration jeweils um eine Position nach rechts zu versetzen (solange bis die Zielstelle gefunden wurde)
 - Zuletzt wird die Zielposition mit dem gesicherten Wert gefüllt, so dass:

$$a_0 \leq \dots \leq \text{key} \leq \dots \leq a_{j-1} \text{ , d.h. } a_0 \leq \dots \leq a_j \text{ (Schleifenende bei } j = n - 1 \text{)}$$

Laufzeit Insertion Sort

- Best Case

$$T(n) = \sum_{i=1}^{n-1} 4 = \Theta(n)$$

- Average Case

$$T(n) = \sum_{i=1}^{n-1} \frac{i}{2} = \frac{(n-1)n}{4} = \Theta(n^2)$$

- Worst Case

$$T(n) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = \Theta(n^2)$$

Sortieren durch Vertauschen (Bubble Sort)

```
void BubbleSort(int a[], int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = n-2; j >= i; j--)
        {
            if (a[j] > a[j+1])
            {
                int h = a[j];
                a[j] = a[j+1];
                a[j+1] = h;
            }
        }
    }
}
```

Vertauschung:
Minimum wandert wie
eine Blase nach vorne

Anmerkung: Man kann auch die Maxima „wandern“ lassen

- Laufzeit: $T(n) = \Theta(n^2)$ (gilt für Best Case, Average Case und Worst Case!)

Beispiel Bubble Sort I

34	45	12	34	23	18	38	17	43	7
34	45	12	34	23	18	38	17	7	43
34	45	12	34	23	18	38	7	17	43
34	45	12	34	23	18	7	38	17	43
34	45	12	34	23	7	18	38	17	43
34	45	12	34	7	23	18	38	17	43
34	45	12	7	34	23	18	38	17	43
34	45	7	12	34	23	18	38	17	43
34	7	45	12	34	23	18	38	17	43
<u>7</u>	34	45	12	34	23	18	38	17	43
<u>7</u>	34	45	12	34	23	18	38	17	43
<u>7</u>	34	45	12	34	23	18	17	38	43

Beispiel Bubble Sort II

<u>7</u>	34	45	12	34	23	17	18	38	43
<u>7</u>	34	45	12	34	17	23	18	38	43
<u>7</u>	34	45	12	17	34	23	18	38	43
<u>7</u>	34	45	12	17	34	23	18	38	43
<u>7</u>	34	12	45	17	34	23	18	38	43
<u>7</u>	<u>12</u>	34	45	17	34	23	18	38	43
<u>7</u>	<u>12</u>	34	45	17	34	23	18	38	43
<u>7</u>	<u>12</u>	34	45	17	34	23	18	38	43
<u>7</u>	<u>12</u>	34	45	17	34	18	23	38	43
<u>7</u>	<u>12</u>	34	45	17	18	34	23	38	43
<u>7</u>	<u>12</u>	34	45	17	18	34	23	38	43
<u>7</u>	<u>12</u>	34	17	45	18	34	23	38	43

Beispiel Bubble Sort III

<u>7</u>	<u>12</u>	<u>17</u>	34	45	18	34	23	38	43
<u>7</u>	<u>12</u>	<u>17</u>	34	45	18	34	23	38	43
<u>7</u>	<u>12</u>	<u>17</u>	34	45	18	34	23	38	43
<u>7</u>	<u>12</u>	<u>17</u>	34	45	18	23	34	38	43
<u>7</u>	<u>12</u>	<u>17</u>	34	45	18	23	34	38	43
<u>7</u>	<u>12</u>	<u>17</u>	34	18	45	23	34	38	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	34	45	23	34	38	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	34	45	23	34	38	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	34	45	23	34	38	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	34	23	45	34	38	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	34	45	34	38	43


Beispiel Bubble Sort IV

<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	34	45	34	38	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	34	45	34	38	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	34	34	45	38	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	34	45	38	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	34	45	38	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	34	38	45	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	38	45	43
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	38	43	45
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>38</u>	43	45
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>38</u>	<u>43</u>	<u>45</u>

Sortieren durch Auswählen (Selection Sort)

```
void SelectionSort(int a[], int n)
{
    int i, j, min;
    for (i = 0; i < n; i++)
    {
        min = i;
        for (j = i; j < n; j++)
        {
            if (a[j] < a[min]) min = j;
        }
        int h = a[i];
        a[i] = a[min];
        a[min] = h;
    }
}
```

Minimum wird ermittelt
und an die richtige
Stelle gesetzt



- Laufzeit: $T(n) = \Theta(n^2)$ (gilt für Best Case, Average Case und Worst Case!)

Beispiel Selection Sort

34	45	12	34	23	18	38	17	43	7
<u>7</u>	45	12	34	23	18	38	17	43	34
<u>7</u>	<u>12</u>	45	34	23	18	38	17	43	34
<u>7</u>	<u>12</u>	<u>17</u>	34	23	18	38	45	43	34
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	23	34	38	45	43	34
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	34	38	45	43	34
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	38	45	43	34
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>38</u>	43	45
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>38</u>	<u>43</u>	45

- Komponentenweiser Aufbau
 - Lösungsfindung erfolgt dadurch, dass die Initiallösung schrittweise vergrößert wird (induktive Fortsetzung)
 - Gleichzeitig wird das Restproblem in jedem Schritt um eins verkleinert
- In jedem Schritt sind $\Theta(n)$ viele Vergleichs- bzw. Vertauschoperationen notwendig, daher **quadratischer Aufwand**
- Einfache Sortieralgorithmen sind nur für kurze Folgen geeignet
- Sortieren durch Einfügen ist ideal, wenn eine sehr gut vorsortierte Folge vorliegt
 - Anwendung z.B. Aktualisierung von geometrischen Datenstrukturen

Quicksort (C.A.R. Hoare, 1962)

```
void PreparePartition(int a[],int f,int l,int &p)
{
    // Pivot-Element
    int pivot = a[f]; p = f-1;
    for (int i = f; i <= l; i++)
    {
        if (a[i] <= pivot)
        {
            p++; swap(a[i],a[p]);
        }
    }
    // Pivot an die
    // richtige Stelle
    swap(a[f],a[p]);
}
```

```
void swap(int &a,int &b)
{
    int h=b;
    b=a;
    a=h;
}
```

```
void Quicksort(int a[],int f,int l)
{
    int part;
    if (f<l) {
        PreparePartition(a,f,l,part);
        Quicksort(a,f,part-1);
        Quicksort(a,part+1,l);
    }
}
```

- Divide & Conquer
- Gruppierung nach
Pivot- bzw. Split-Element

Beispiel Quicksort I

34	45	12	34	23	18	38	17	43	7
34	45	12	34	23	18	38	17	43	7
34	12	45	34	23	18	38	17	43	7
34	12	34	45	23	18	38	17	43	7
34	12	34	23	45	18	38	17	43	7
34	12	34	23	18	45	38	17	43	7
34	12	34	23	18	45	38	17	43	7
34	12	34	23	18	17	38	45	43	7
34	12	34	23	18	17	38	45	43	7
34	12	34	23	18	17	7	45	43	38

Beispiel Quicksort II

7	12	34	23	18	17	<u>34</u>	45	43	38
7	12	34	23	18	17	<u>34</u>	45	43	38
7	12	34	23	18	17	<u>34</u>	45	43	38
7	12	34	23	18	17	<u>34</u>	45	43	38
7	12	34	23	18	17	<u>34</u>	45	43	38
7	12	34	23	18	17	<u>34</u>	45	43	38
7	12	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	12	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	12	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	12	34	23	18	17	<u>34</u>	45	43	38

Beispiel Quicksort III

<u>7</u>	12	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	12	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	12	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	17	23	18	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	17	23	18	<u>34</u>	<u>34</u>	45	43	38

Beispiel Quicksort IV

<u>7</u>	<u>12</u>	17	23	18	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	17	23	18	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	23	18	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	23	18	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	23	18	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	38	43	<u>45</u>

Beispiel Quicksort V

7 12 17 18 23 34 34 **38** 43 45

7 12 17 18 23 34 34 **38** 43 45

7 12 17 18 23 34 34 38 43 45

Beispiel Quicksort (kompakt)

34	45	12	34	23	18	38	17	43	7
7	12	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	12	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	34	23	18	17	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	17	23	18	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	23	18	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	45	43	38
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	38	43	<u>45</u>
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>38</u>	43	<u>45</u>
<u>7</u>	<u>12</u>	<u>17</u>	<u>18</u>	<u>23</u>	<u>34</u>	<u>34</u>	<u>38</u>	<u>43</u>	<u>45</u>

- Anzahl der Schritte hängt von der **Wahl des Pivot-Elements** ab
- Best Case (Halbierung mit jedem Rekursionsaufruf)

$$T(n) = 2T\left(\frac{n}{2}\right) + n = \Theta(n \log n)$$

- Worst Case (Je Rekursionsaufruf wird nur 1 Element bearbeitet)

$$T(n) = T(n - 1) + T(0) + n = \Theta(n^2)$$

- Average Case

$$T_i(n) = T(i - 1) + T(n - i) + n$$

(Anzahl der Schritte bei Trennwert $i \in \{1, \dots, n\}$)

- Annahme: Alle Positionen sind gleichwahrscheinlich, dann gilt:

$$T(n) = \frac{1}{n} \sum_{i=1}^n (T(i - 1) + T(n - i) + n)$$

Laufzeit Quicksort II

- Es gilt:

$$T(n) = \frac{1}{n} \sum_{i=1}^n (T(i-1) + T(n-i) + n)$$

$$= \frac{1}{n} \sum_{i=1}^n (T(i-1) + T(n-i)) + \frac{1}{n} \sum_{i=1}^n n$$

$$= \frac{1}{n} (T(0) + T(n-1) + T(1) + T(n-2) + \dots + T(n-1) + T(0)) + \frac{n^2}{n}$$

$$= \frac{2}{n} \sum_{i=0}^{n-1} T(i) + n$$

- Weiter gilt:

$$n \cdot T(n) = 2 \sum_{i=0}^{n-1} T(i) + n^2$$

- Für $n - 1$ gilt also:

$$(n - 1) \cdot T(n - 1) = 2 \sum_{i=0}^{n-2} T(i) + (n - 1)^2$$

- Differenz der unteren Gleichung von der oberen liefert:

$$n \cdot T(n) - (n - 1) \cdot T(n - 1) = 2 \sum_{i=0}^{n-1} T(i) + n^2 - \left(2 \sum_{i=0}^{n-2} T(i) + (n - 1)^2 \right)$$

- Umformung ergibt:

$$2 \sum_{i=0}^{n-1} T(i) + n^2 - \left(2 \sum_{i=0}^{n-2} T(i) + (n - 1)^2 \right) = 2T(n - 1) + 2n - 1$$

- Insgesamt folgt:

$$n \cdot T(n) - (n - 1) \cdot T(n - 1) = 2T(n - 1) + 2n - 1$$

- D.h.:

$$n \cdot T(n) - (n + 1) \cdot T(n - 1) = 2n - 1$$

- Teilung durch $n(n + 1)$ liefert:

$$\frac{T(n)}{n + 1} = \frac{T(n - 1)}{n} + \frac{2n - 1}{n(n + 1)}$$

- Substitution $\hat{T}(n) = \frac{T(n)}{n+1}$ liefert:

$$\hat{T}(n) = \hat{T}(n - 1) + \frac{2n - 1}{n(n + 1)} = \hat{T}(n - 2) + \frac{2(n - 1) - 1}{(n - 1)((n - 1) + 1)} + \frac{2n - 1}{n(n + 1)}$$

$$\dots = \hat{T}(1) + \sum_{i=2}^n \frac{2i - 1}{i(i + 1)} = \hat{T}(1) + \sum_{i=2}^n \frac{3}{i + 1} - \sum_{i=2}^n \frac{1}{i}$$

- Erinnerung **Harmonische Reihe**:

$$\sum_{k=1}^n \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \Theta(\log n)$$

- Damit folgt:

$$\hat{T}(n) = \hat{T}(1) + \sum_{i=2}^n \frac{3}{i+1} - \sum_{i=2}^n \frac{1}{i} = \hat{T}(1) + \left(\frac{3}{3} - \frac{1}{2}\right) + \left(\frac{3}{4} - \frac{1}{3}\right) + \left(\frac{3}{5} - \frac{1}{4}\right) + \dots$$

$$= \hat{T}(1) - \frac{1}{2} + 2 \sum_{i=3}^n \frac{1}{i} + \frac{3}{n+1} = \Theta(\log n)$$

- Rücksubstitution $T(n) = \hat{T}(n) \cdot (n+1)$ liefert für die Laufzeit von Quicksort:

$$T(n) = \Theta(n \log n)$$