OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

INFORMATIK UND
MATHEMATIK

Algorithmen und Datenstrukturen

- Sortieralgorithmen -

Prof. Dr. Klaus Volbert

Wintersemester 2018/19
Regensburg, 22. Oktober 2018

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

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

Beispiel Quicksort III

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

Beispiel Quicksort IV

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

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

Laufzeit Quicksort I

- 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:

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

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

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

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

Laufzeit Quicksort III

- 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:

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

Laufzeit Quicksort IV

- 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:

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

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

Laufzeit Quicksort V

- Erinnerung Harmonische Reihe:

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

- Damit folgt:

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

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

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

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

Anmerkungen Quicksort

- Quicksort gilt in der Praxis als
  - Schnell, berühmt und breit einsetzbar (Laufzeit ideal)
  - In vielen Programmiersprachen ist die Sortierung von Objekten über Standard-Bibliotheken realisiert und verfügbar
  - Bei Verwendung kann/muss eine individuelle Vergleichsfunktion implementiert werden (siehe z.B. Comparable in Java oder C#)
- Varianten für die Wahl des Pivot-Elements:
  - Erstes, letztes, i.tes Element des Feldes
  - Median aus erstes, letztes, i.tes Element des Feldes
  - Ein zufälliges Element des Feldes (Zufallszahl)
- Die letzten beiden Varianten sind in der Praxis auch gut bei fast sortierten Folgen

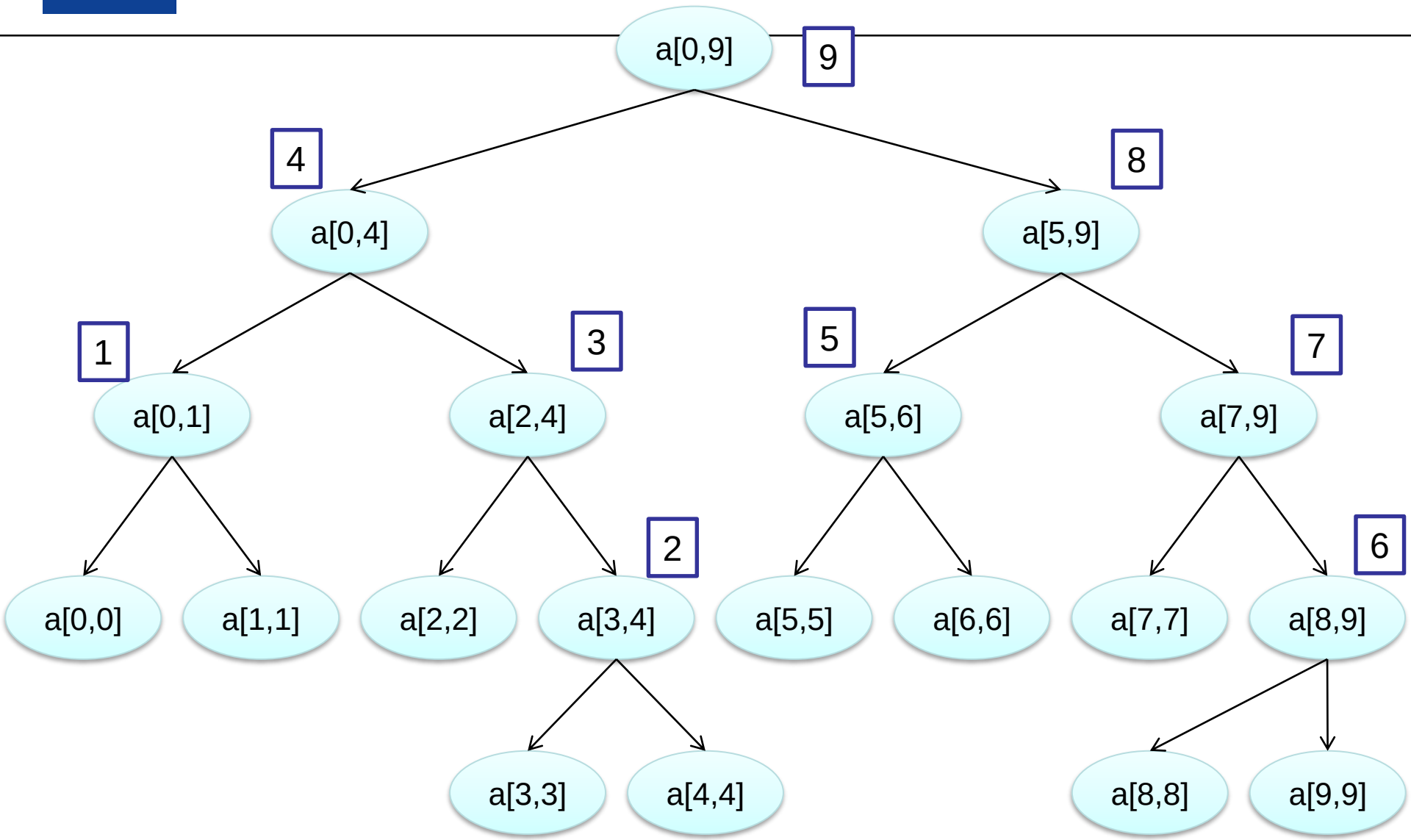
Sortieren durch Mischen (Merge Sort)

```
void Merge(int a[],int f,int l,int m) {
    int i, n = l - f + 1;
    int a1f = f, a1l = m-1;
    int a2f = m, a2l = l;
    int *anew = new int[n];

    for (i = 0;i < n; i++)
    {   if (a1f <= a1l) {
            if (a2f <= a2l)
            {   if (a[a1f] <= a[a2f]) anew[i]=a[a1f++];
                else anew[i]=a[a2f++]; }
            else anew[i]=a[a1f++]; }
        else anew[i]=a[a2f++]; }

    for (i=0;i<n;i++) a[f+i]=anew[i];

    delete [] anew; }
```

```
void MergeSort(int a[],int f,int l)
{   if (f<l) {
        int m = (f+l+1)/2;
        MergeSort(a,f,m-1);
        MergeSort(a,m,l);
        Merge(a,f,l,m); }
}
```

Zwei sortierte Teilfolgen werden unter Verwendung eines neuen Feldes zum Umspeichern zu einer sortierten Folge „gemischt“

Rekursionsbaum über die Aufrufe



Beispiel Merge Sort I

34 45 12 34 23 | 18 38 17 43 7	Teilen
34 45 | 12 34 23 18 38 17 43 7	Teilen
34 | 45 12 34 23 18 38 17 43 7	Teilen
34 45 12 34 23 18 38 17 43 7	Mischen
34 45 12 | 34 23 18 38 17 43 7	Teilen
34 45 12 34 | 23 18 38 17 43 7	Teilen
34 45 12 23 34 18 38 17 43 7	Mischen
34 45 12 23 34 18 38 17 43 7	Mischen
12 23 34 34 45 18 38 17 43 7	Mischen
12 23 34 34 45 18 38 | 17 43 7	Teilen

Beispiel Merge Sort II

12	23	34	34	45	18 |	38	17	43	7	Teilen
12	23	34	34	45	18	38	17	43	7	Mischen
12	23	34	34	45	18	38	17 |	43	7	Teilen
12	23	34	34	45	18	38	17	43 |	7	Teilen
12	23	34	34	45	18	38	17	7	43	Mischen
12	23	34	34	45	18	38	7	17	43	Mischen
12	23	34	34	45	7	17	18	38	43	Mischen
7	12	17	18	23	34	34	38	43	45	Mischen

Laufzeit Merge Sort

- Laufzeit (Best Case, Average Case , Worst Case):

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

- Lösen der Rekursionsgleichung durch
  - Iterationsmethode, oder
  - Master-Methode
- Liefert:

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

- Wir sagen:
  - In jeder Rekursionsstufe werden alle Zahlen gemischt: $\Theta(n)$
  - Halbierung hat $\Theta(\log n)$ Rekursionsstufen, also folgt $\Theta(n \log n)$

Anmerkungen Merge Sort

- Merge Sort geht nach dem Divide-&-Conquer-Entwurfsprinzip vor
- Merge Sort eignet sich ideal zum externen Sortieren, wenn die Daten nicht in den Hauptspeicher passen
  - Teilfolgen können in Dateien vorliegen
  - Folge wird in einer weiteren Datei gemischt
  - Zeitaufwand ist auch hier linear
- Time-Space-Tradeoff (Mehr Platz kann Laufzeit verbessern):
  - Mischen mit Umspeichern (ex situ) hat lineare Komplexität
  - Mischen ohne Umspeichern (in situ) hat quadratische Komplexität
- Merge Sort ist stabil bzgl. der Laufzeit (gleiche Laufzeit in allen Fällen)
- Viele Varianten/Implementierungsarten (z. B. Natural Merge Sort)

Stabilität von Sortieralgorithmen

- Ein Sortieralgorithmus ist stabil, wenn die relative Ordnung der Elemente mit gleichen Schlüsseln durch den Sortierprozess nicht verändert wird
- Beispiel: Sortierung nach Anfangsbuchstaben (1-26)
  - Eingabe: Markus, Sven, Walter, Michael, Franz
    13, 19, 23, 13, 6
  - Ausgabe 1: Franz, Markus, Michael, Sven, Walter
    6, 13, 13, 19, 23
  - Ausgabe 2: Franz, Michael, Markus, Sven, Walter
    6, 13, 13, 19, 23
- Welche der Ausgaben kommt von einem stabilen Sortieralgorithmus?
  - Ausgabe 1: Markus steht weiterhin vor Michael
- Welche Sortieralgorithmen sind stabil?
  - Einfache: Insertion Sort, Bubble Sort
  - Fortgeschrittene: Merge Sort