Hochschule Offenburg
University of Applied Sciences

# Algorithmen und Datenstrukturen
# 3. Sortieren

Prof. Dr. Klaus Dorer

# Übersicht



- Sortieren
  - Elementare Sortierverfahren
    - Bubblesort
    - Sortieren durch Einfügen
    - Shellsort
    - Sortieren durch Auswahl
  - Quicksort
  - Heapsort
  - Mergesort
  - Radixsort

- http://www.youtube.com/watch?v=k4RRi_ntQc8

# Ziele

- Relevante Begriffe kennen
- Verschiedene Sortierverfahren kennen und implementieren können
- Komplexität der Verfahren kennen und einordnen können

# Quellen


- Thomas Ottmann und Peter Widmayer, Algorithmen und Datenstrukturen, 4. Auflage, Spektrum, Berlin, 2002
- Robert Sedgewick, Kevin Wayne, Algorithmen - Algorithmen und Datenstrukturen (4. aktualisierte Auflage), Pearson Studium 2014
- Stephan Trahasch, Vorlesungsfolien Algorithmen und Daten-strukturen, Hochschule Offenburg
- www.wikipedia.de

# Begriffe

- Satz
  - Enthält die Daten
  - Besitzt einen Schlüssel
- Schlüssel
  - Kriterium nach dem sortiert werden soll
  - Für Schlüssel ist eine Ordnungsrelation $<$ oder $\leq$ definiert
- Intern
  - Alle Daten haben im Hauptspeicher platz
- Extern
  - Daten sind auf einem externen Medium gespeichert
- In-situ
  - Daten werden innerhalb der Quelliste/array sortiert, kein zusätzlicher Speicher nötig

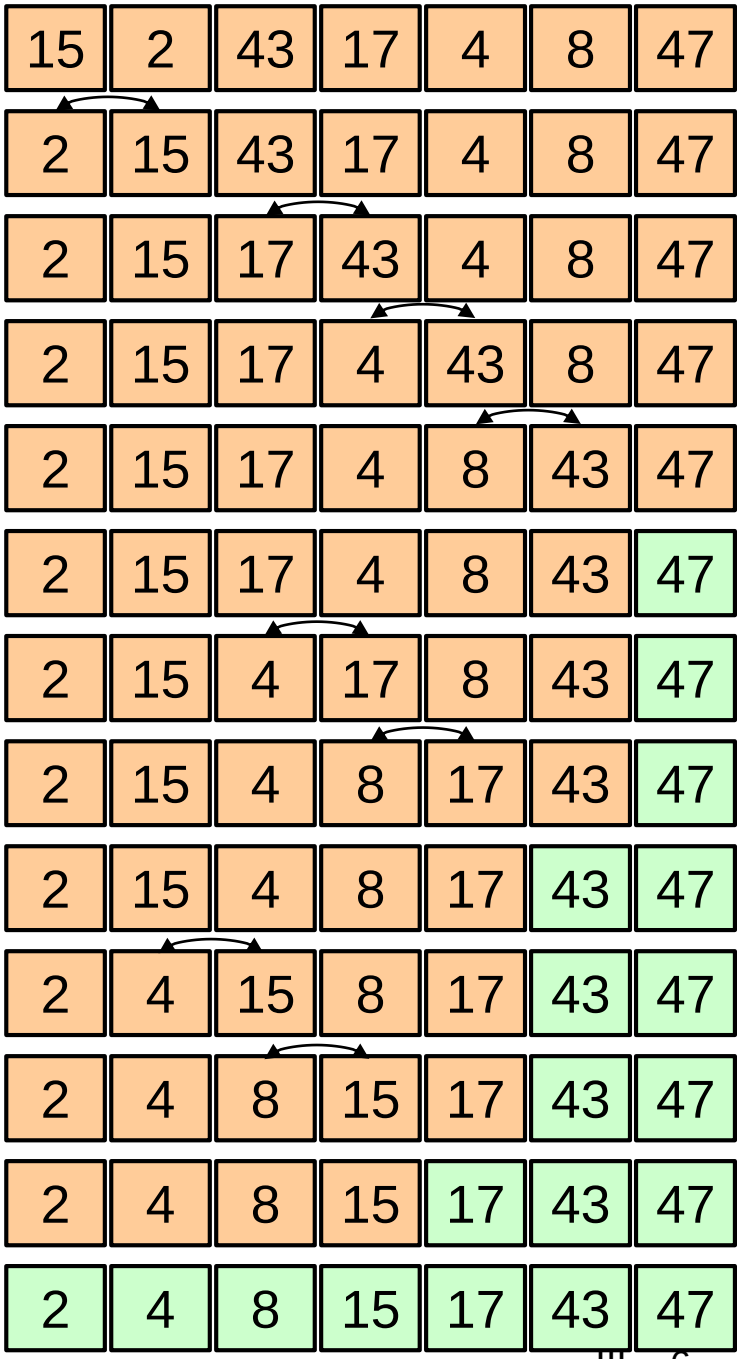
# Begriffe

- Schlüsselvergleich
  - Vergleich zweier Schlüssel im Bezug auf die Ordnungsrelation
- Bewegung
  - Verschiebung eines oder Vertauschung zweier Datensätze
- Stabil
  - Bereits sortierte Daten werden nicht mehr bewegt
- Glatt
  - n verschiedene Schlüssel werden in O(n log n), n gleiche in O(n) sortiert
- Allgemeine Sortierverfahren
  - Verfahren, für das von den Schlüsseln nur eine Ordnungsrelation verlangt wird

# Bubblesort

- Durchlaufe den unsortierten Bereich (rot) und tausche zwei benachbarte Elemente, wenn die Ordnung nicht stimmt
- Nach erstem Durchlauf ist das größte Element am Ende
- Wiederhole jeweils für den noch verbleibenden unsortierten Teil

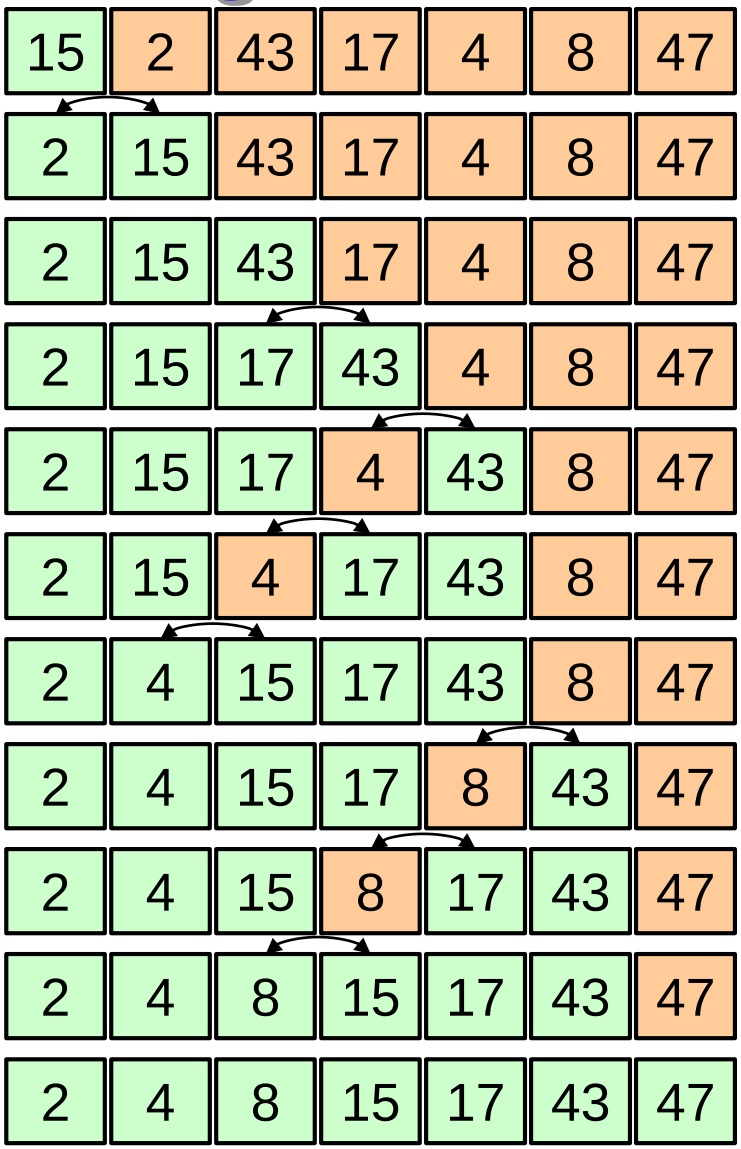
- Laufzeit
  - Schlüsselvergleiche:
  - Bewegungen:



# Sortieren durch Einfügen

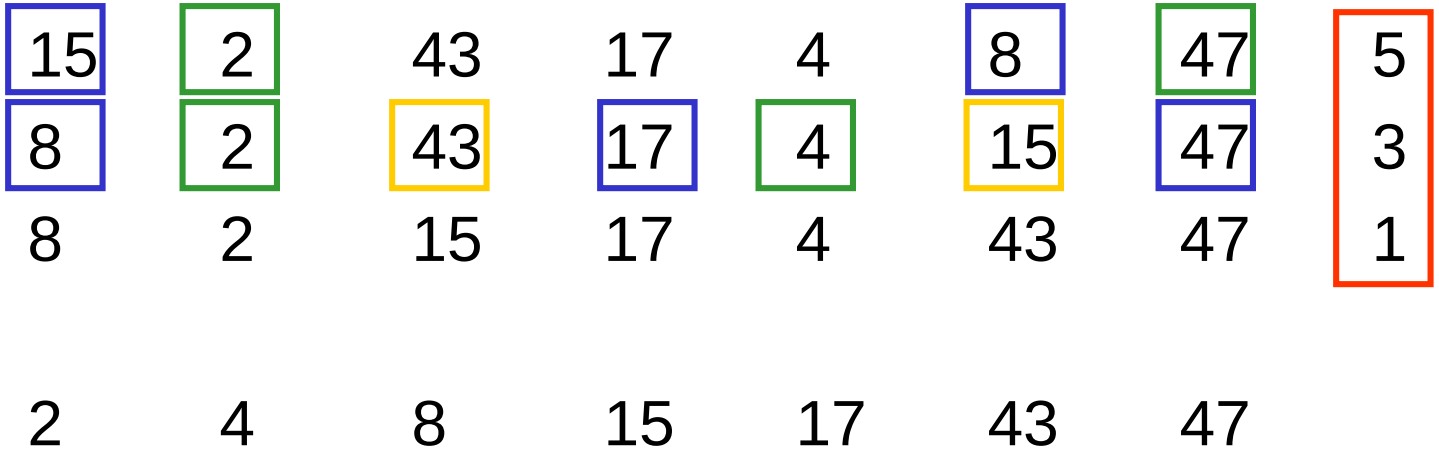
- Tausche das erste Element der unsortierten Teilliste (rot) so lange mit dem linken Nachbarn, bis es an der richtigen Stelle im sortierten Teil (grün) steht

- Laufzeit:
  - Vergleiche:
  - Bewegungen:



# Shellsort

- Wähle abnehmende Folge von Inkrementen (z.B. 5, 3, 1)
- Sortiere alle ‚Teilfolgen' mit Sortieren durch Einfügen

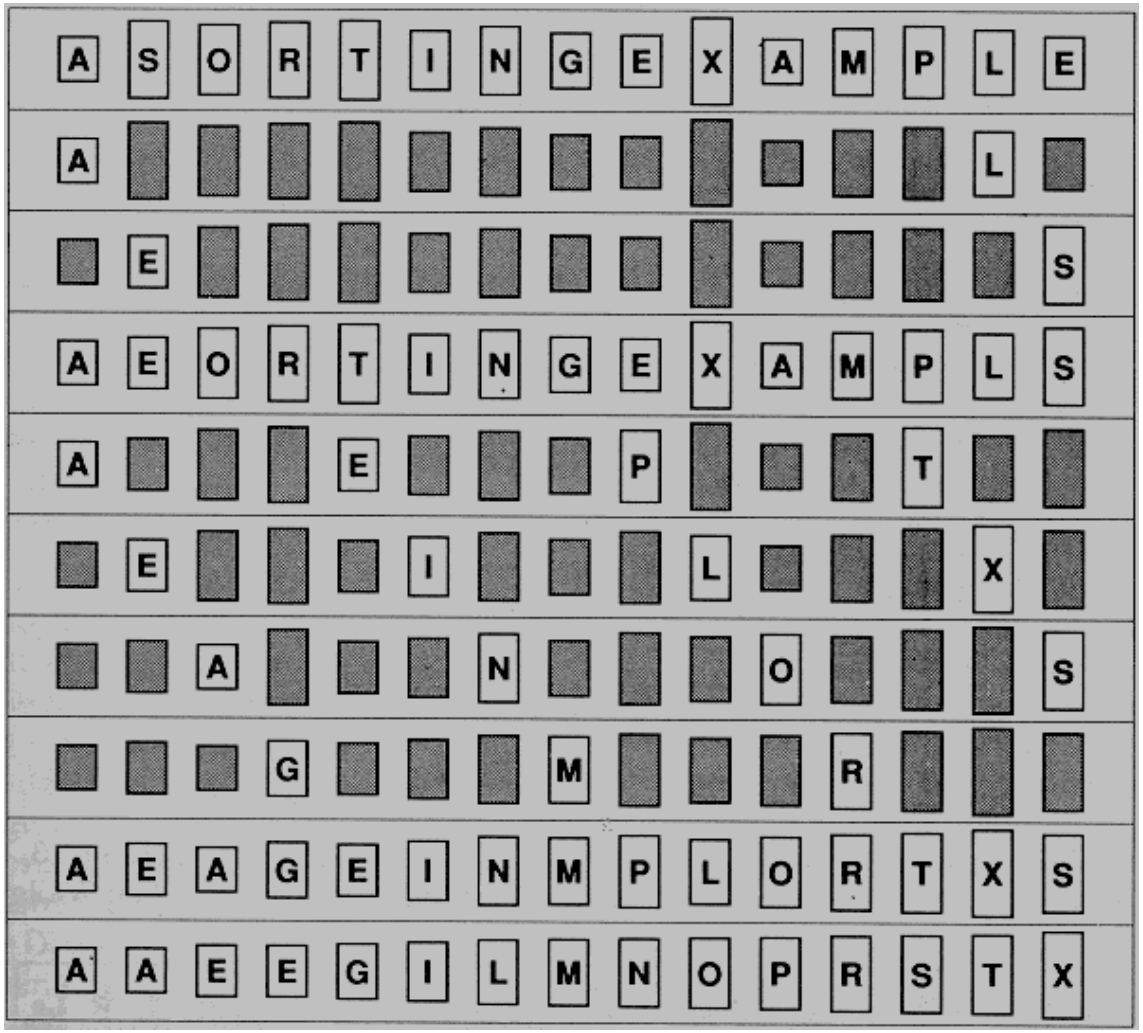



- Laufzeit: hängt von der Folge von Inkrementen ab
  - $O(n^{1.5})$, wenn Inkrement eine Folge $2^p-1$ ist
  - $O(n \log^2 n)$, wenn Inkremente eine Folge $2^p3^q$ ist

Quelle: Ottmann

# Shellsort

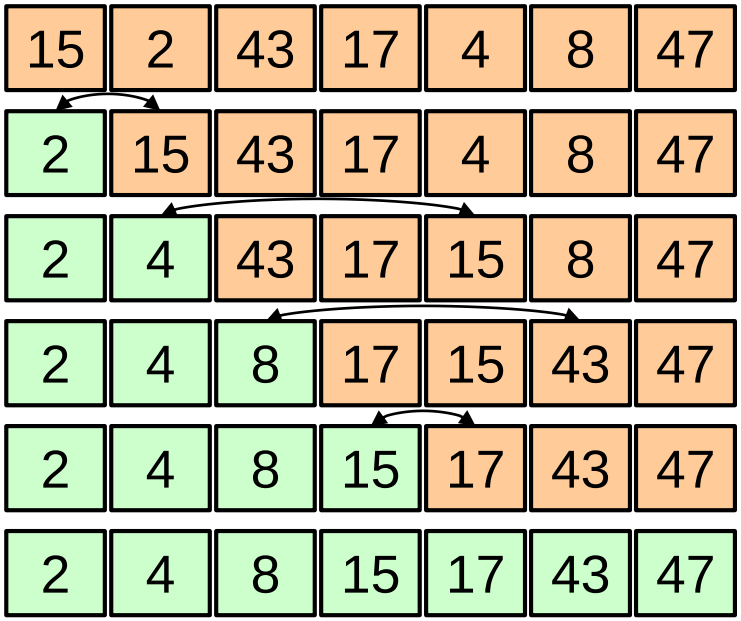
- Beispiel mit Schrittweiten 13, 4, 1




Quelle: Sedgewick

# Sortieren durch Auswahl

- Bestimme die Position j des kleinsten Schlüssels im unsortierten Teil und tausche mit dem ersten des unsortierten Teils
- Wiederhole für alle entsprechend

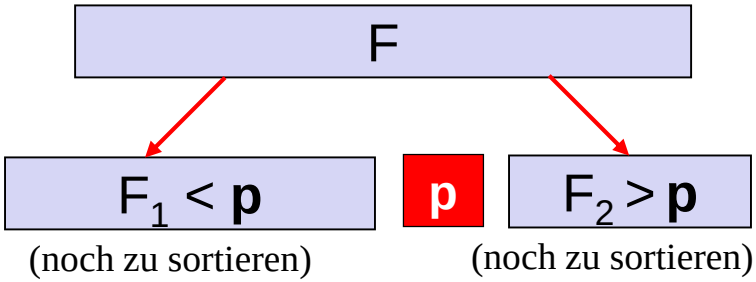



- Laufzeit:
  - Schlüsselvergleiche:
  - Bewegungen:

# Quicksort

*Tony Hoare*

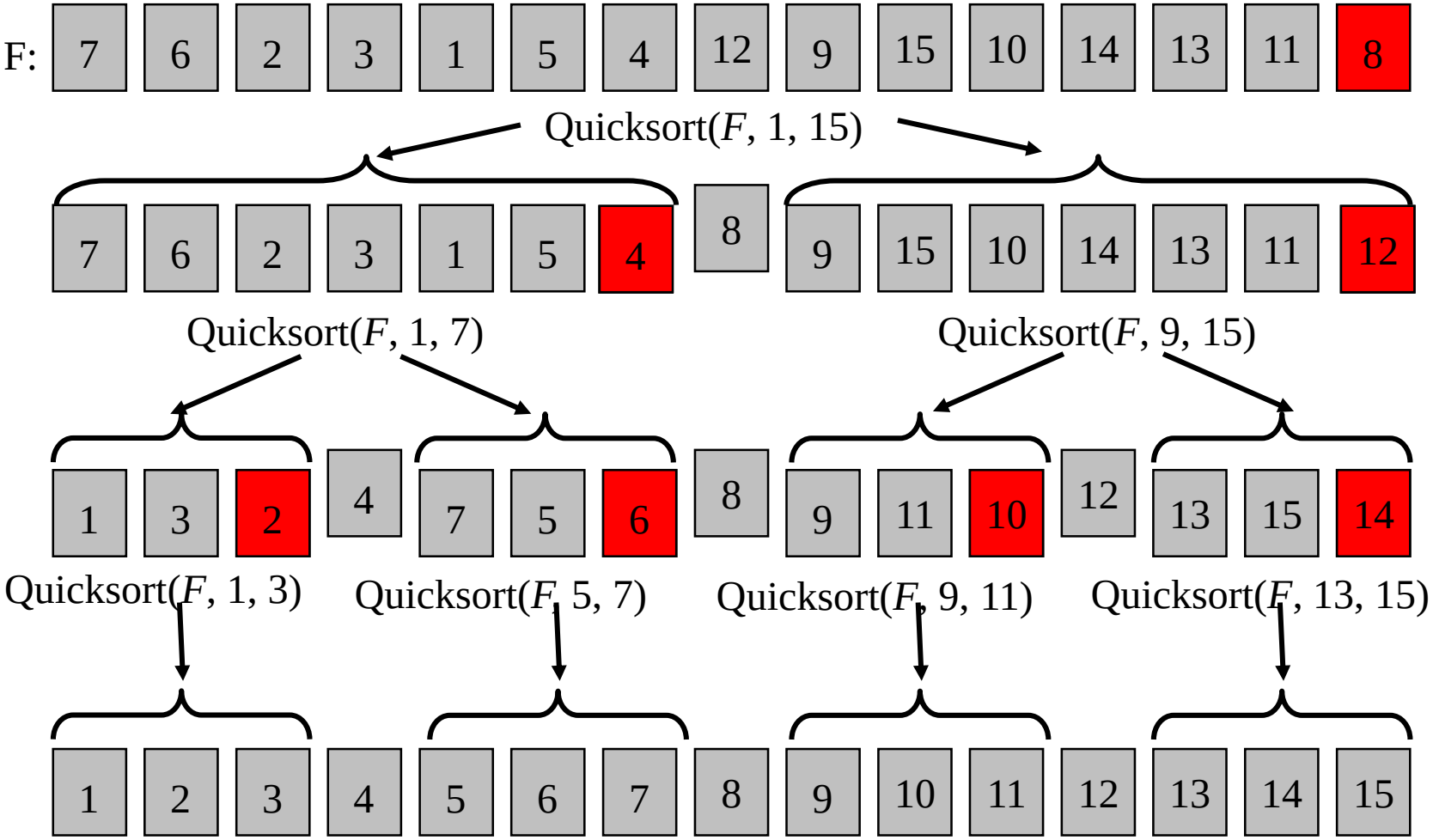
- 1962 von Hoare vorgestellt
- Divide and conquer
  - Wähle ein beliebiges Element p (Pivotelement) und teile die Folge ohne p in 2 Teile, so dass in Folge 1 nur Elemente <= p und in Folge 2 nur Elemente > p sind
  - Setze p zwischen die beiden Folgen an die richtige Stelle




  - Sortiere die linke und die rechte Teilfolge ebenso
- Laufzeit:
  - $O(n^2)$ im schlechtesten Fall
  - O(n log n) im Mittel und besten Fall

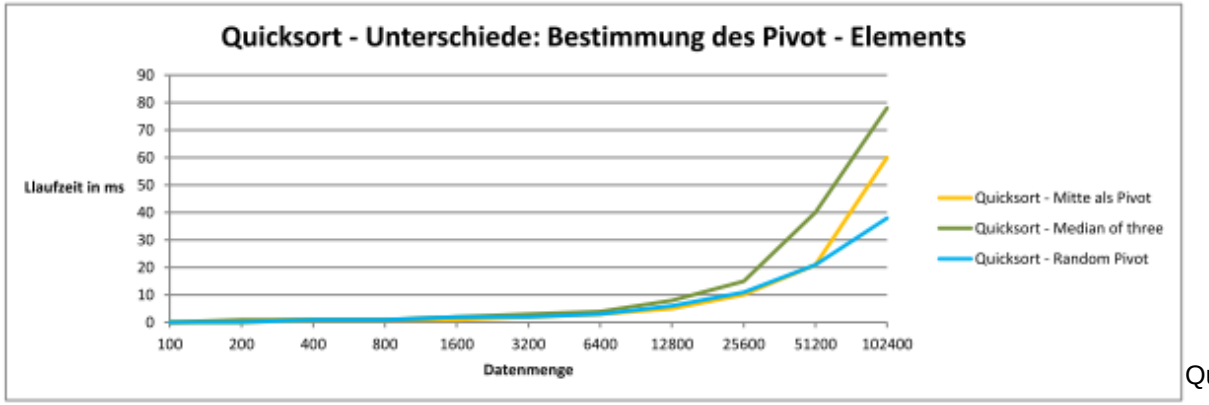
# Quicksort

- Beispiel



# Quicksort

- Problem: wählt man letztes Element als Pivotelement und ist die Folge bereits sortiert ist die Laufzeit in $O(n^2)$
- Wahl des Pivotelements
  - Im schlechtesten Fall ist das Pivotelement immer das kleinste/größte
  - 3-Median-Strategie: wähle mittleres des ersten, letzten und mittleren Elements
  - Zufallsstrategie: wähle eine Element zufällig
  - Median-Strategie: wähle das mittlere aller Elemente als Pivotelement. Gut?

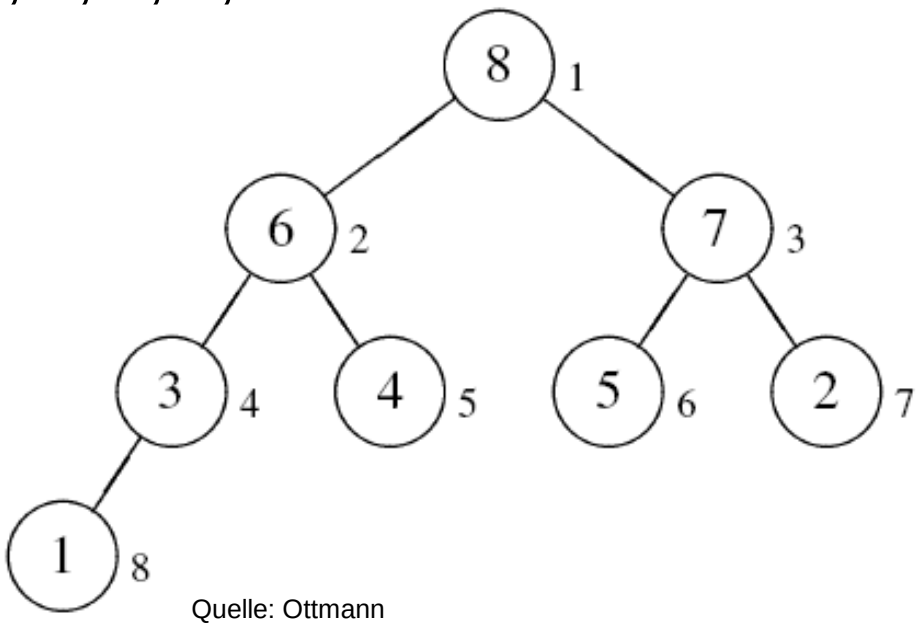



Quelle: Sven Keller

# Heapsort

*Robert W. Floyd*

- 1964 von Floyd und Williams vorgestellt
- Sortieren mit Auswahl, aber geschickte Auswahl
- Datenstruktur Heap als Binärbaum repräsentieren
  - Bestimmung des Maximums in einem Schritt möglich
  - Folge $F=k_1,k_2,...k_n$ ist Heap, wenn $k_i \geq k_{2i}$ und $k_i \geq k_{2i+1}$
- Beispiel: 8, 6, 7, 3, 4, 5, 2, 1
- Als Binärbaum



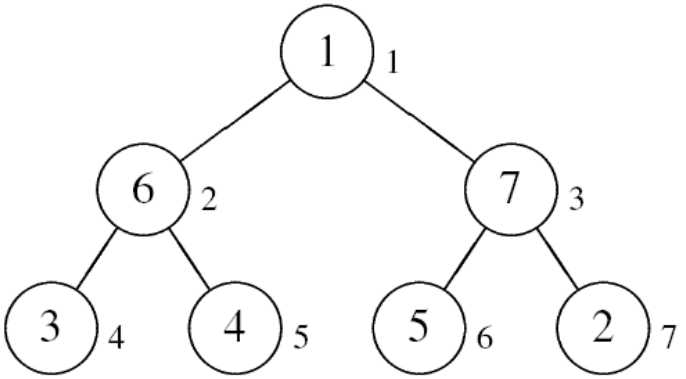

Quelle: Ottmann

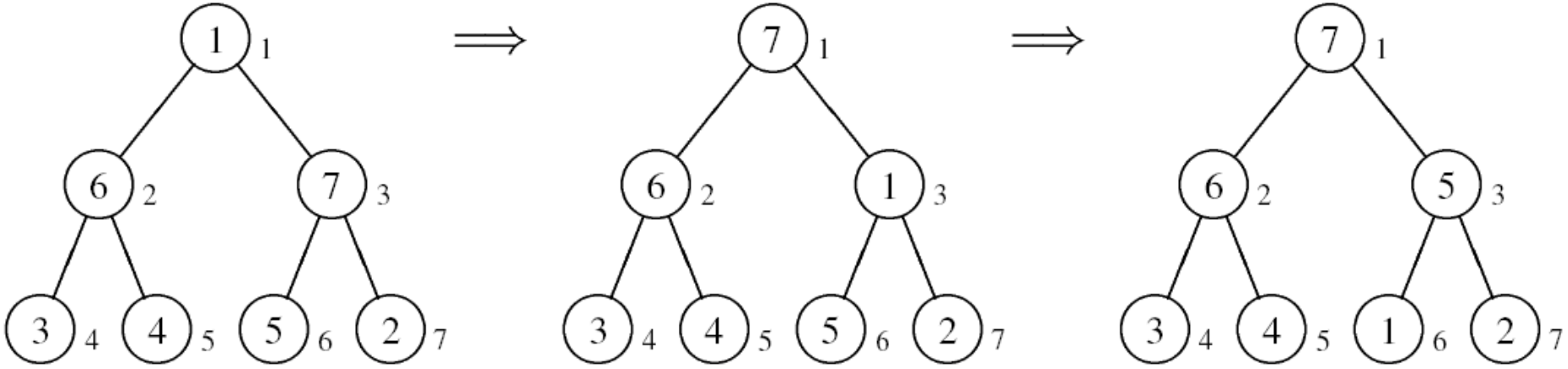
# Heapsort

- Stelle aus einer Folge einen Heap her
  - Lasse $k_{n/2}$, $k_{n/2-1}$,... $k_1$ in F ‚versickern'
  - O(n)
- Gib den Heap in sortierter Reihenfolge aus
  - $k_1$ ausgeben und aus dem Heap entfernen
  - Heapbedingung für die restlichen Schlüssel wiederherstellen
    - $k_m$ mit $k_1$ tauschen (m ist höchster Index)
    - Lasse $k_1$ ‚versickern' im Bereich 1..m-1
  - O(n log n)
- Eigenschaften
  - Im schlechtesten Fall O(n log n)
  - Nicht stabil
  - In-Situ

# Heapsort

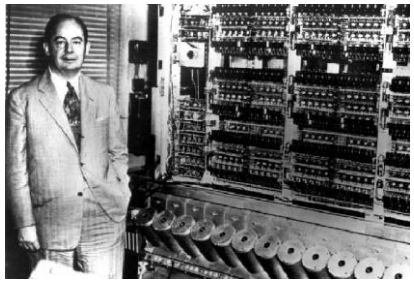
- Versickern lassen
  - Baum 1 genügt nicht der Heapbedingung
  - 1 versickern lassen
  - Tausche 1 mit dem größeren der beiden Kinder, bis kein Kind mehr größer ist




Quelle: Ottmann

# Mergesort



*Von Neumann*

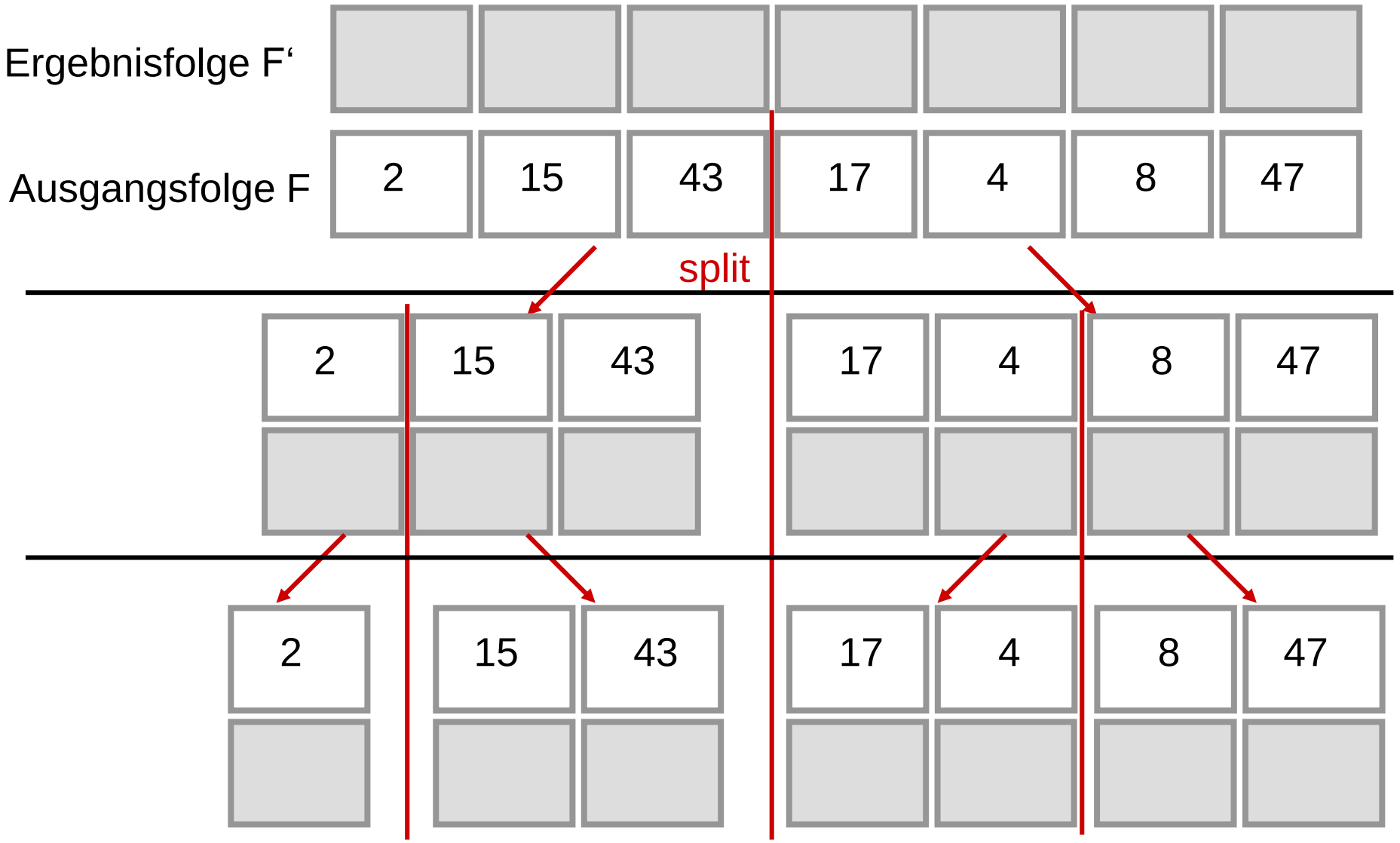
- Von John von Neuman 1945 vorgeschlagen
- 2-Wege-Mergesort
  - Rekursiv
- Reines 2-Wege-Mergesort
  - Iterativ
- Natürliches 2-Wege-Mergesort
  - Nützt bereits sortierte Teilfolgen aus
- Eigenschaften
  - Benötigt zusätzlichen Speicher: O(n)
  - Stabil
  - Feld wird immer durchwandert, kein direkter Zugriff auf ein Feldelement nötig

# 2-Wege-Mergesort

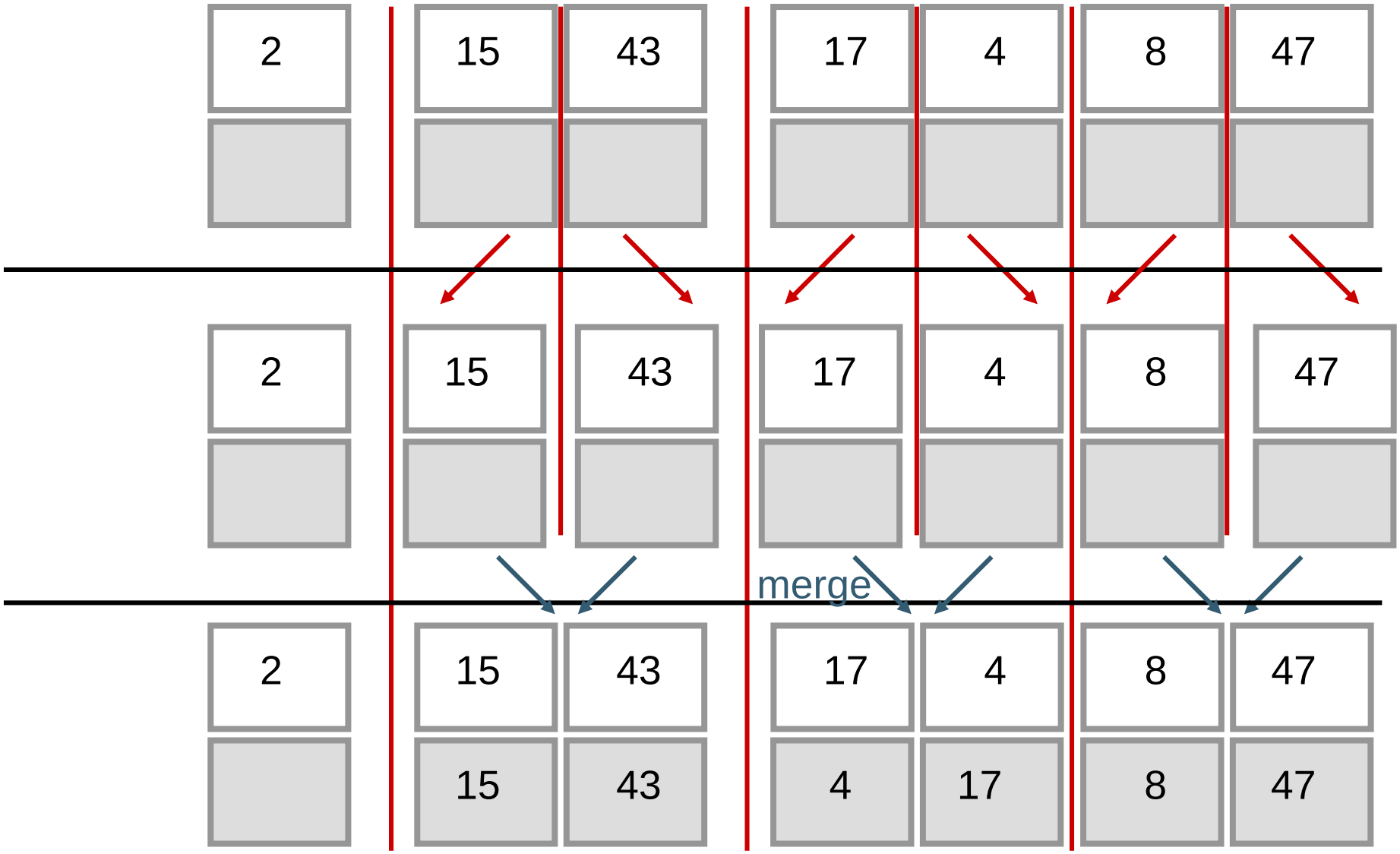
- Divide and conquer
  - Teile Folge F in 2 möglichst gleich große Teilfolgen
  - Sortiere linke und rechte Folge (mit Mergesort)
  - ‚Verschmelze' linke und rechte Folge

| | F1 | F2 | Ergebnis |
|---|---|---|---|
| | 1, 2, 3, 5 | 4, 5, 6, 7 | |
| Anfang | ↑i | ↑j | - |
| 1<4 | ↑i | ↑j | 1 |
| 2<4 | ↑i | ↑j | 1, 2 |
| 3<4 | ↑i | ↑j | 1, 2, 3 |
| 5>4 | ↑i | ↑j | 1, 2, 3, 4 |
| 5=5 | ↑i | ↑j | 1, 2, 3, 4, 5 |
| F1 fertig | ↑i | ↑j | 1, 2, 3, 4, 5, 5 |
| F1 fertig | ↑i | ↑j | 1, 2, 3, 4, 5, 5, 6 |
| F1 fertig | ↑i | ↑j | 1, 2, 3, 4, 5, 5, 6, 7 |

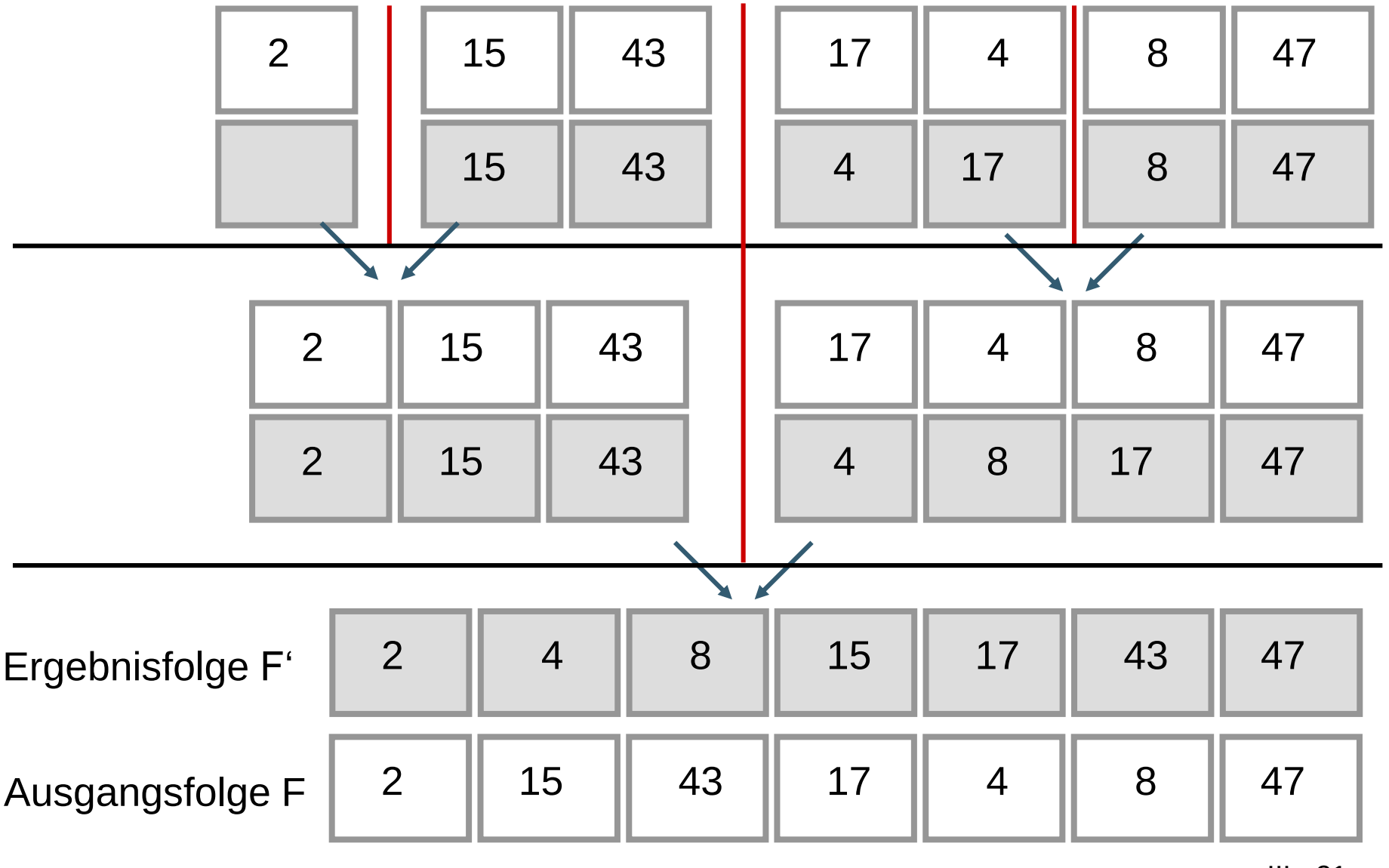
# Beispiel: Mergesort



# Beispiel: Mergesort



# Beispiel: Mergesort



# Reines 2-Wege-Mergesort

- Verschmelze sortierte Teilfolgen zu immer längeren
- Erster Durchgang: verschmelze $k_1$ mit $k_2$, $k_3$ mit $k_4$, usw.
- Dann $k_1$, $k_2$ mit $k_3$, $k_4$ usw.
- Solange bis nur noch 2 Teilfolgen verschmolzen wurden
- Auch Bottom-up Mergesort genannt
- Beispiel

| Länge | Folge |
|---|---|
| 1 | 2 \| 1 \| 3 \| 9 \| 5 \| 6 \| 7 \| 4 \| 8 |
| 2 | 1, 2 \| 3, 9 \| 5, 6 \| 4, 7 \| 8 |
| 4 | 1, 2, 3, 9 \| 4, 5, 6, 7 \| 8 |
| 8 | 1, 2, 3, 4, 5, 6, 7, 9 \| 8 |
| fertig | 1, 2, 3, 4, 5, 6, 7, 8, 9 |

# Natürliches 2-Wege-Mergesort

- Anstatt immer Teilfolgen der Länge 1 zu verschmelzen kann man Vorsortierung nutzen und möglichst lange sortierte Teilfolgen verschmelzen
  - Teile F in möglichst lange sortierte Teilfolgen
  - Verschmelze benachbarte Teilfolgen
- Beispiel

| Schritt | Folge |
|---|---|
| 1 | 2 \| 1, 3, 9 \| 5, 6, 7 \| 4, 8 |
| 2 | 1, 2, 3, 9 \| 4, 5, 6, 7, 8 |
| 3 | 1, 2, 3, 4, 5, 6, 7, 8, 9 |

- Eigenschaften
  - Im besten Fall O(n)
  - Im durchschnittlichen und schlechtesten Fall O(n log n)

# Radixsort

- Bisher waren Schlüsselvergleiche die einzige Informationsquelle für das Sortieren
- Jetzt seien Schlüssel Wörter aus einem Alphabet mit m Elementen
  - m = 10 und die Schlüssel sind Dezimalzahlen
  - m = 2 und die Schlüssel sind Dualzahlen
  - m = 26 und die Schlüssel sind Wörter über dem Alphabet {a..z}
- Die Schlüssel sind also m-adische Zahlen mit der Wurzel (lat. Radix) der Darstellung m
- Annahmen
  - Die maximale Länge der Schlüssel ist bekannt
  - Es gibt eine Funktion $z_m(i,k)$, die in konstanter Zeit die i te Ziffer (von rechts) vom m-adischen Schlüssel k liefert: $z_{10}(1, 345) = 4$

# Radixsort

- Wiederhole von der niedrigsten zur höchsten Ziffer
  - 1. Teile Folge von Schlüsseln in m Teile (Fächer) aufgrund der aktuellen Ziffer im Schlüssel (Partitionieren)
  - 2. Kopiere die Fächer von links nach rechts unter Beibehaltung der Reihenfolge zurück (Sammeln)
- Eigenschaften
  - Laufzeit: ?
  - Speicher: ?

Quelle: Ottmann

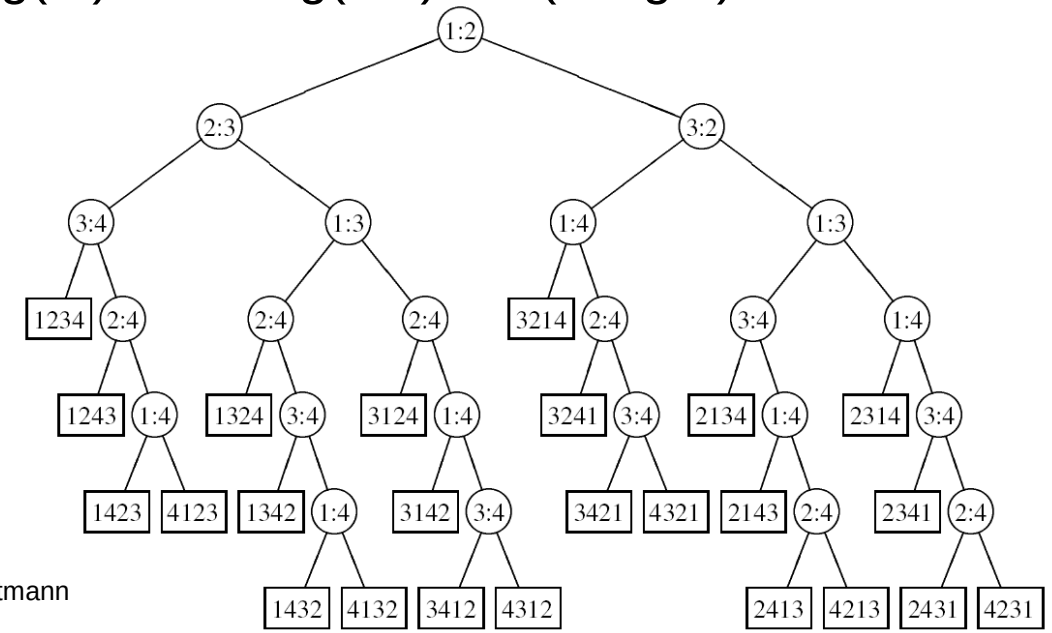
# Radixsort

- Beispiel

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Folge | 123, 823, 181, 885, 125 | | | | | | | | |
| Partitionieren | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| (erste Ziffer) | | 181 | | 123 | | 885 | | | | |
| | | | | 823 | | 125 | | | | |
| Sammeln | 181, 123, 823, 885, 125 | | | | | | | | | |
| Partitionieren | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| (zweite Ziffer) | | | 123 | | | | | | 181 | |
| | | | 823 | | | | | | 885 | |
| | | | 125 | | | | | | | |
| Sammeln | 123, 823, 125, 181, 885 | | | | | | | | | |
| Partitionieren | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| (dritte Ziffer) | | 123 | | | | | | | 823 | |
| | | 125 | | | | | | | 885 | |
| | | 181 | | | | | | | | |
| Sammeln | 123, 125, 181, 823, 885 | | | | | | | | | |

# Untere Schranke

- Alle elementaren Sortierverfahren benötigen im Mittel und im schlechtesten Fall **Ω(n log n)** Schlüsselvergleiche
  - Entscheidungsbaum enthalte alle n! Permutationen einer Folge als Blätter, Knoten entsprechen einem Schlüsselvergleich
  - Maximale und mittlere Tiefe ist dann >= log(n!)
  - Untere Abschätzung: $n! \geq (n/2)^{n/2}$
  - Damit gilt $\log(n!) \geq n/2 \log(n/2) = \Omega(n \log n)$




Quelle: Ottmann

# Sortieren in Java

- Die Klassen Arrays und Collections bieten Sortiermethoden

```
// sorting arrays with natural ordering
Arrays.sort(namesArray);
// sorting lists
Collections.sort(namesList);

// using comparator
Comparator<String> comparator = new DescendingComparator<String>();
Arrays.sort(namesArray, comparator);

// comparator class
class DescendingComparator<T extends Comparable<T>> implements Comparator<T>
{
    @Override
    public int compare(T first, T second)
    {
        return first.compareTo(second) * -1;
    }
}
```

# Zusammenfassung

| Algorithmus | Bester | Mittel | Schlechtester | Stabil | Rekursiv |
|---|---|---|---|---|---|
| Auswahl<br>Vergleiche, Bewegungen | $O(n^2)$<br>$\sim n^2/2$, $3n$ | $O(n^2)$<br>$\sim n^2/2$, $3n$ | $O(n^2)$<br>$\sim n^2/2$, $3n$ | nein | nein |
| Einfügen | $O(n)$<br>$n$, $2n$ | $O(n^2)$<br>$\sim n^2/4$, $\sim n^2/4$ | $O(n^2)$<br>$\sim n^2/2$, $\sim n^2/2$ | ja | nein |
| Shellsort | | $O(n^{1.25})$ | $O(n \log^2 n)$ | nein | nein |
| Bubblesort | $O(n)$<br>$n-1$, $0$ | $O(n^2)$<br>$\sim n^2/2$, $\sim n^2/4$ | $O(n^2)$<br>$\sim n^2/2$, $\sim n^2/2$ | ja | nein |
| Quicksort | $O(n \log n)$ | $O(n \log n)$ | $O(n^2)$ | nein | ja |
| Heapsort | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ | nein | nein |
| Mergesort | $O(n)$<br>(natürliches) | $O(n \log n)$ | $O(n \log n)$ | ja | ja |

Quelle: Wikipedia