DHBW Lörrach
Duale Hochschule
Baden-Württemberg

Algorithmen und Komplexität
TIF 21A/B
Dr. Bruno Becker

6. Prioritätswarteschlangen

Prioritätswarteschlangen

- **Grundlegende Eigenschaften**
- Binäre Heaps
- Heapsort

Prioritätswarteschlangen (Priority Queues)

- **Warteschlangen:** Supermarktkasse, Auto-Maut, Druckaufträge,…
  First in, First out (FIFO)
- **Operationen*:***
  - Erster Kunde (Element) wird bedient
  - Neuer Kunde (Element) wird hinten angehängt

Anwendungen für Priority Queues

- Betriebssysteme: Job Scheduling, Load Balancing
- Suche in Graphen: Dijkstra Algorithmus für kürzeste Wege (später in der Vorlesung)
- Datenkompression: Huffman-Codes (später in der Vorlesung)
- …

Abstrakter Datentyp für Priority Queue

- Input: Elemente vom totalgeordneten Typ *Item.*

  „Ältestes“ Element = Maximum

```
Public class PQ <Item>  implements iterable <item>
        Queue() /* Ezeugt leere PQ
        void  insert (Item item)  /* Fügt ein Element hinzu
        Item searchMax () /* Sucht Maximum
        Item  deleteMax()  /* Entfernt Maximum
        boolean isEmpty() /* Ist die PQ leer?
        int size()  /* Anzahl der Elemente in der PQ
```

Implementierung Priority Queue

- Verkettete Liste:
  - Unsortiert: Einfügen schnell, aber Suche nach Maximum O (*n*)
  - Sortierte Liste: Suche Maximum schnell (konstanter Aufwand), aber Liste sortiert halten O (*n*)
- Baumstruktur:
  - Binärer Suchbaum im Worst Case schlecht (wie lineare Liste)
  - Balancierter Baum: Kompliziert, wenn Schlüssel mehrfach vorkommen…

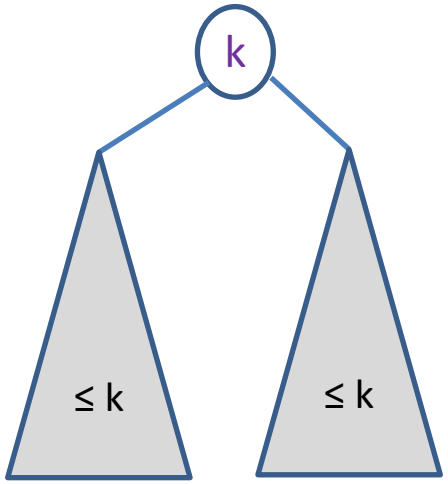
Idee: Struktur muss nicht komplett sortiert sein. Nur das Maximum muss immer greifbar sein

Prioritätswarteschlangen

- Grundlegende Eigenschaften
- **Binäre Heaps**
- Heapsort

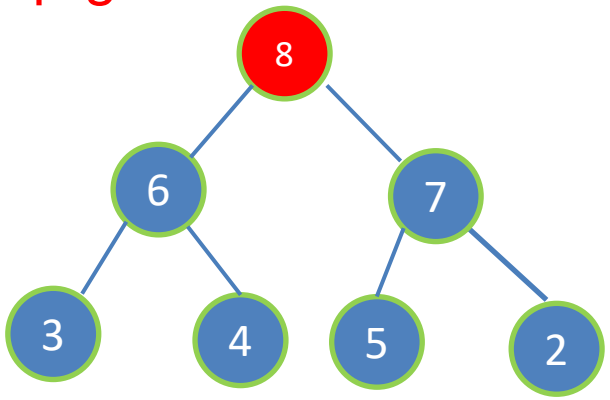
Heap-Ordnung

Ein Binärbaum mit Schlüsseln aus einer Totalordnung „≤“ heißt *Heap-geordnet,* wenn gilt:
Der Schlüssel in *jedem* Knoten ist ≥ seiner Kinder



Binäre Heaps

Beispiele für Heap-geordnete Bäume:



aber auch lineare, absteigende Liste ist Heap-geordnet.
=> Anforderungen an Ordnung und Struktur!

Binärbaum heißt **fast vollständig**, wenn

- Alle Ebenen (bis auf die unterste) voll besetzt sind
- Die unterste Ebene von links nach rechts lückenlos

Binäre Heaps

- Definition von „fast vollständig“ sehr ähnlich zu „höhenbalanciert“

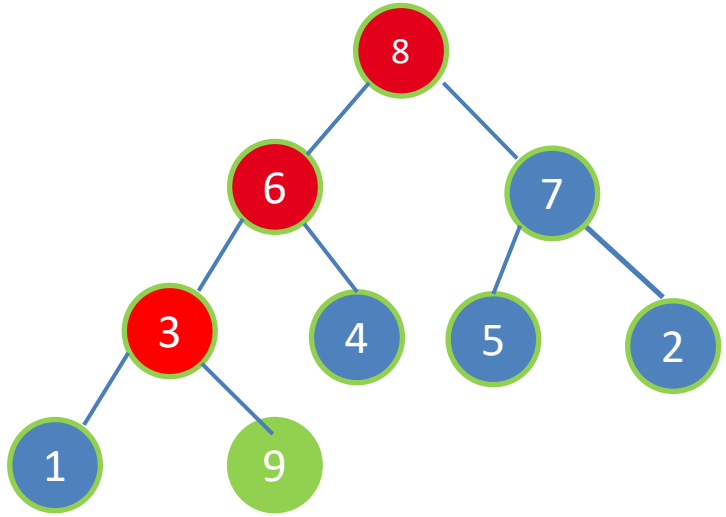
➔ Ähnliche Eigenschaften, insbesondere logarithmische Höhe

Ein **binärer (Max)-Heap** ist ein Binärbaum, der

- *Heap-geordnet* ist, und
- *fast vollständig* ist

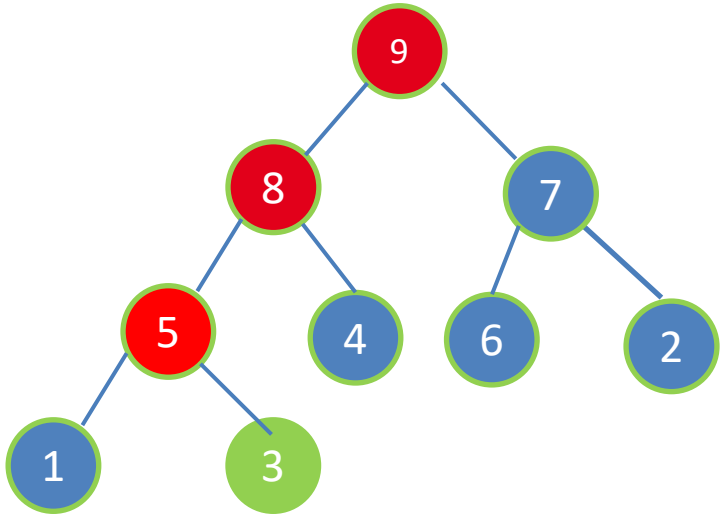
Einfügen in binäre Heaps

- Neuen Knoten am Ende des Baumes einfügen -> Baum bleibt fast vollständig
- Neuer Knoten „*schwimmt hoch*“, solange er die Heap-Ordnung verletzt
  - d.h. vertausche Knoten mit seinem Vater
- Vertauschung endet spätestens in der Wurzel, d.h. maximal *log N*-Schritte



Maximum löschen aus binären Heaps

- Maximum in Wurzel durch letzten Knoten *k* ersetzen,
  d.h. Baum bleibt fast vollständig
- Knoten *k* „*sinkt*“ von der Wurzel ab, solange er die Heap-Ordnung verletzt
  – d.h. vertausche Knoten mit seinem größeren Sohn
- Vertauschung endet spätestens auf unterstem Niveau, maximal O(*log N*)



Implementierung der Heap-Datenstruktur

Baum – dynamische Struktur

- Zeiger auf Kinder und Rückverweis auf Vater
- Vorteil: Variable Größe

Array – implizite Baum-Struktur (statische Struktur)

- Wurzel in a [1]
- Kinder von a[i] in a[2*i] und a[2*i +1]
- Sehr schnelle Navigation (Indexrechnung)
- Bei binärem Heap lückenlos besetzt

Übung: Algorithmen für Einfügen und Maximum entfernen

Prioritätswartechlangen

- Grundlegende Eigenschaften
- Binäre Heaps
- **Heapsort**

Sortieren mit Heaps

Prioritätswarteschlangen (PQ) können zum Sortieren verwendet werden

1. Füge alle Elemente in anfangs leere PQ ein: *n* mal insert
2. Auslesen und Löschen des Maximums aus PQ, bis PQ leer: n-mal

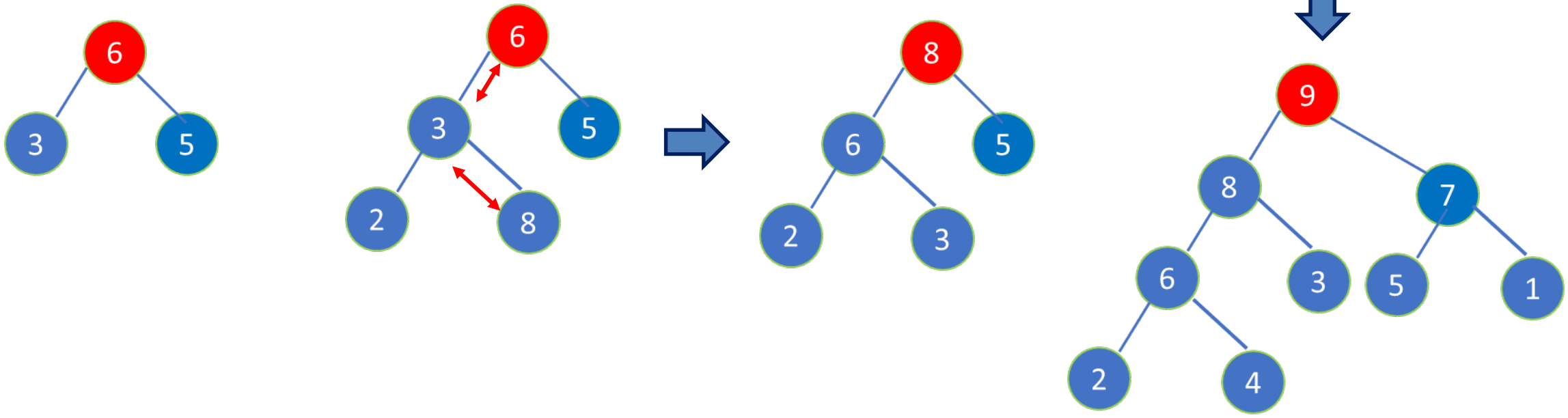
Damit dieses Sortierverfahren effizient ist, muss PQ als Heap implementiert sein

Beispiel

Sortiere Folge mit Heapsort

3-6-5-2-8-7-1-9-4

1. Heap aufbauen

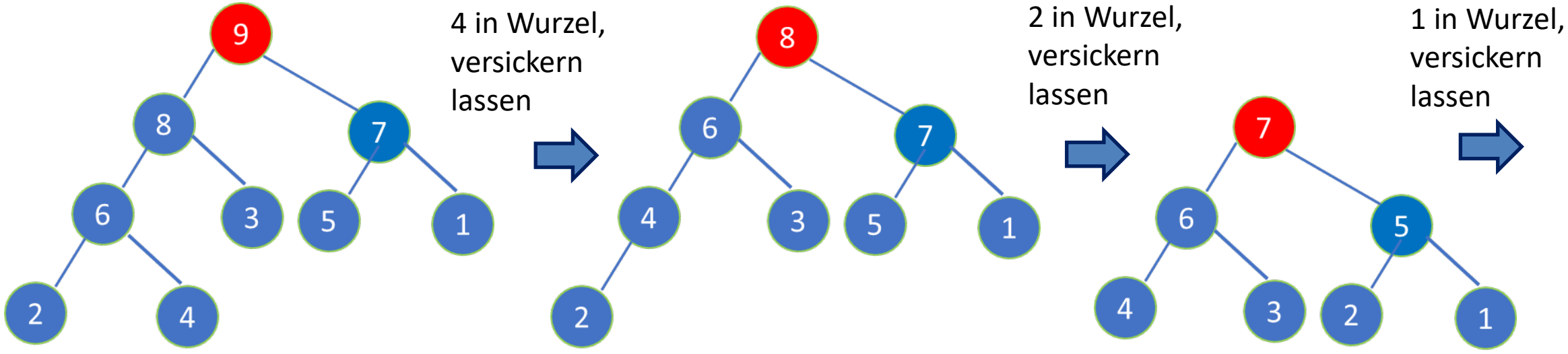


Beispiel

Sortiere Folge mit Heapsort

3-6-5-2-8-7-1-9-4

2. n-mal Maximum entfernen

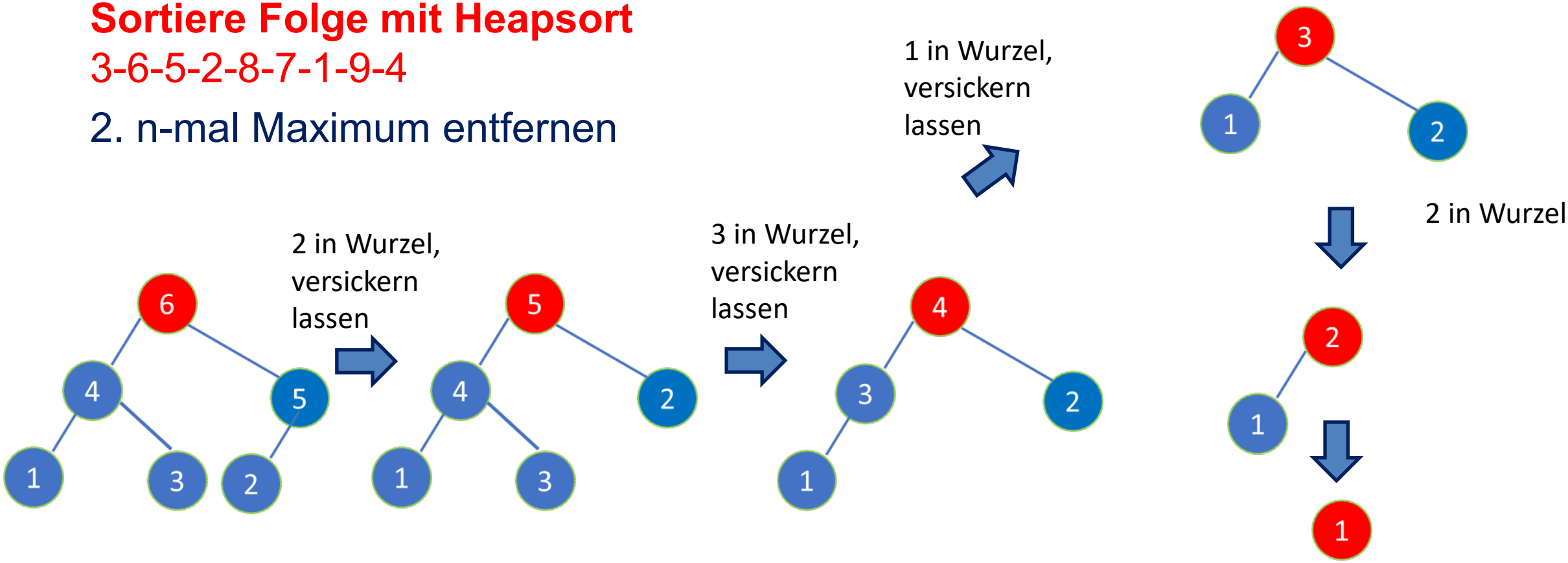


Beispiel

Sortiere Folge mit Heapsort

3-6-5-2-8-7-1-9-4

2. n-mal Maximum entfernen



Heapsort Aufwandsanalyse

Worst case Analyse

1. Füge alle Elemente in anfangs leere PQ ein: *n* mal insert
   => O (n log n)
2. Auslesen und Löschen des Maximums aus PQ, bis PQ leer: n-mal
   => O(n log n)

→**Heapsort Worst case optimal**

In place: Ja (z.B. Array von hinten auffüllen, Heap schrumpft)

→Heapsort Sortierverfahren, dass in place ist und Wort case optimal hinsichtlich Laufzeit! (Mergesort nicht in Place, Quicksort worst case $O(n^2)$)

In der Praxis aber meistens Quicksort schneller