Informatik II
Prüfungsvorbereitungskurs

Tag 3, 22.6.2016

Giuseppe Accaputo

g@accaputo.ch

Themenübersicht

- 20.3: **Java**
- 21.3: **Objektorientierte Programmierung**
- 22.3: **Dynamische Datenstrukturen**
- 23.3: **Datenbanksysteme**

Programm für heute

- Repetition von gestrigen Themen
- Asymptotische Komplexität
- Arrays
- Verkettete Liste
- Stapel (Stack)
- Hashtabelle (lineares Sondieren)
- Zufallszahlen
- Bäume
- Heaps
- Online Median

Repetition:
Objektorientierte Programmierung

Prüfung 02.2014 Aufgabe 1a

- Whiteboard

Prüfung 02.2014 Aufgabe 7c & 7d

- Whiteboard

Prüfung 01.2015 Aufgabe 6

- Whiteboard

Repetition der Flashcards aus der Vorlesung

- Flashcard 8 (Whiteboard)
- Flashcard 9 (Whiteboard)

Part 3:
Dynamische Datenstrukturen

Asymptotische Komplexität

- Big-O Notation: obere Schranke für die Laufzeit eines Algorithmus (*worst case*)
- Ignoriert konstante Faktoren
  - Beispiel: $3\mathrm{n} \in O(n)$, d.h. $3\mathrm{n}$ wächst höchstens so schnell wie n

Asymptotische Komplexität

- Sei n die Grösse der zu verarbeitenden Daten

1. $O(1)$: in konstanter Zeit ausführbar (am schnellsten)
2. $O(\log(n))$: logarithmische Laufzeit
3. $O(n)$: lineare Laufzeit
4. $O(n^2)$: quadratische Laufzeit
5. $O(c^n)$: exponentielle Laufzeit (am langsamsten)

Asymptotische Komplexität

- $O(1)$: Direkter Zugriff auf Element in Array
- $O(\log(n))$: Traversierung Binärbaum
- $O(n)$: Liste durchsuchen
- $O(n^2)$: Matrix-Vektor Produkt

Aufgabe Asymptotische Komplexität

- Was ist die Komplexität der folgenden Funktion?

```
public static void f1(int a[]){
    if(a.length != 0)
        a[0] = 1;
}
```

Lösung Asymptotische Komplexität

- Was ist die Komplexität der folgenden Funktion?

```
public static void f1(int a[]){
    if(a.length != 0)
        a[0] = 1;
}
```

$$O(1)$$

Aufgabe Asymptotische Komplexität

- Was ist die Komplexität der folgenden Funktion?

```
public static void f2(int n){
    int res = 0;
    for(int i = 0; i<n; i++)
        res += i;
}
```

Lösung Asymptotische Komplexität

- Was ist die Komplexität der folgenden Funktion?

```
public static void f2(int n){
    int res = 0;
    for(int i = 0; i<n; i++)
        res += i;
}
```

$$O(n)$$

Aufgabe Asymptotische Komplexität

- Was ist die Komplexität der folgenden Funktion?

```
public static void f3(int n){
   int res = 0;
   for(int i = 0; i<n; i++)
      for(int j = 0; j<n; j++)
         res += j*i;
}
```

Lösung Asymptotische Komplexität

- Was ist die Komplexität der folgenden Funktion?

```
public static void f3(int n){
   int res = 0;
   for(int i = 0; i<n; i++)
      for(int j = 0; j<n; j++)
         res += j*i;
}
```

$$O(n^2)$$

Aufgabe Asymptotische Komplexität

- Was ist die Komplexität der folgenden Funktion?

```
public static void f4(int n){
   int res = 0;
   for(int i = 0; i<n; i++)
          res += i;
   for(int j = 0; j<n; j++)
          res += j;
}
```

Lösung Asymptotische Komplexität

- Was ist die Komplexität der folgenden Funktion?

```
public static void f4(int n){
   int res = 0;
   for(int i = 0; i<n; i++)
         res += i;
   for(int j = 0; j<n; j++)
         res += j;
}
```

$$O(2n) \Longrightarrow O(n)$$

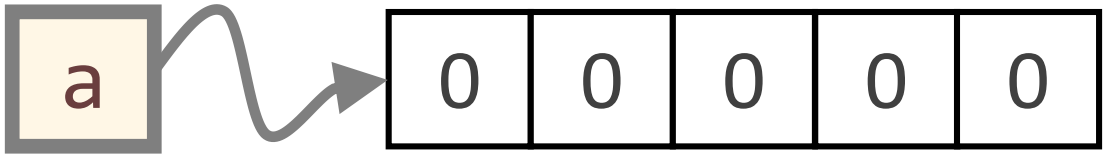
Prüfung 08.2015 Aufgabe 6

- Whiteboard

Arrays

- Zusammenhängender Speicherbereich

```
int a[] = new int[5];
```



- Wahlfreier Zugriff auf `i`-tes Element
- Fixierte Länge
- Alle Elemente sind vom gleichen Typ

Probleme mit Arrays

- Element in der Mitte einfügen oder löschen ist aufwändig
- Whiteboard

Probleme mit Arrays

- Element in der Mitte einfügen oder löschen ist aufwändig, da evtl. Zellen verschoben werden müssen
- Frage: Laufzeit um Element in der Mitte zu entfernen?

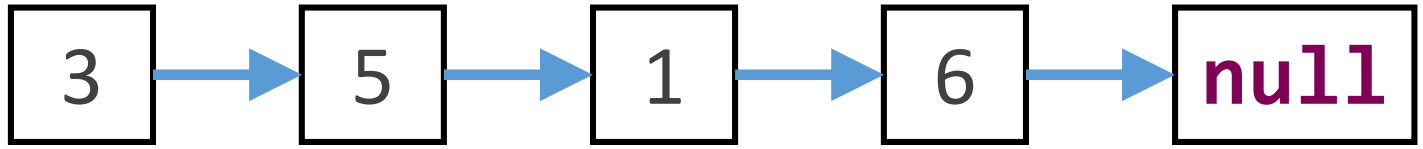
Probleme mit Arrays

- Element in der Mitte einfügen oder löschen ist aufwändig, da evtl. Zellen verschoben werden müssen
- Frage: Laufzeit um Element in der Mitte zu entfernen?

$$O(n)$$

Verkettete Liste

- Container um Folge von Daten zu speichern
- Alle Elemente sind vom gleichen Typ
- Kein zusammenhängender Speicherbereich, d.h. kein wahlfreier Zugriff
  - muss durch Liste gehen um auf i-tes Element zuzugreifen

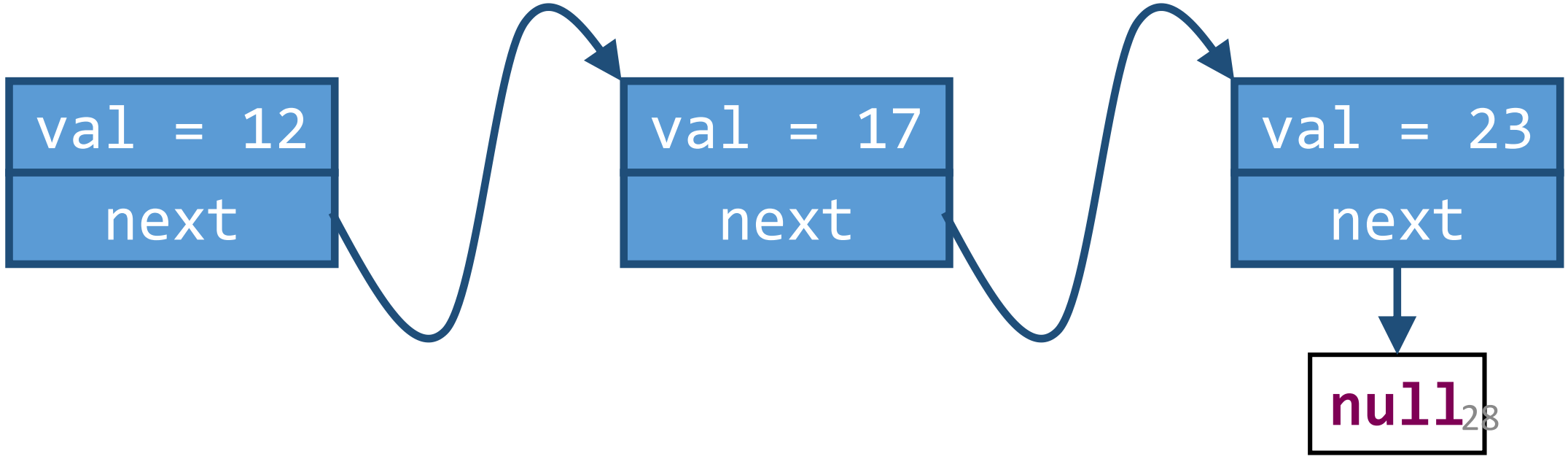


Einfach verkettete Liste: Code

```
class Node
{
    double val;
    Node next;
    Node (double v, Node nxt)
    {
        val = v;
        next = nxt;
    }
}
```

Einfach verkettete Liste: Im Speicher

```
Node n1 = new Node(23, null);
Node n2 = new Node(17, n1);
Node n3 = new Node (12, n2);
```



Liste traversieren

```
void printList(Node n) {
  while (n != null){
    System.out.println(n.value);
    n = n.next; // Wichtig!
  }
}
```

- Frage: Laufzeit von `printList`?

Liste traversieren

```
void printList(Node n) {
  while (n != null){
    System.out.println(n.value);
    n = n.next; // Wichtig!
  }
}
```

- Frage: Laufzeit von `printList`?

$$O(n)$$

Prüfung 08.2015 Aufgabe 4c

- Was ist die Komplexität im schlechtesten Fall für das Suchen eines Elementes in einer einfach verketteten sortierten Liste der Länge n?
- Whiteboard

Prüfung 01.2015 Aufgabe 4a

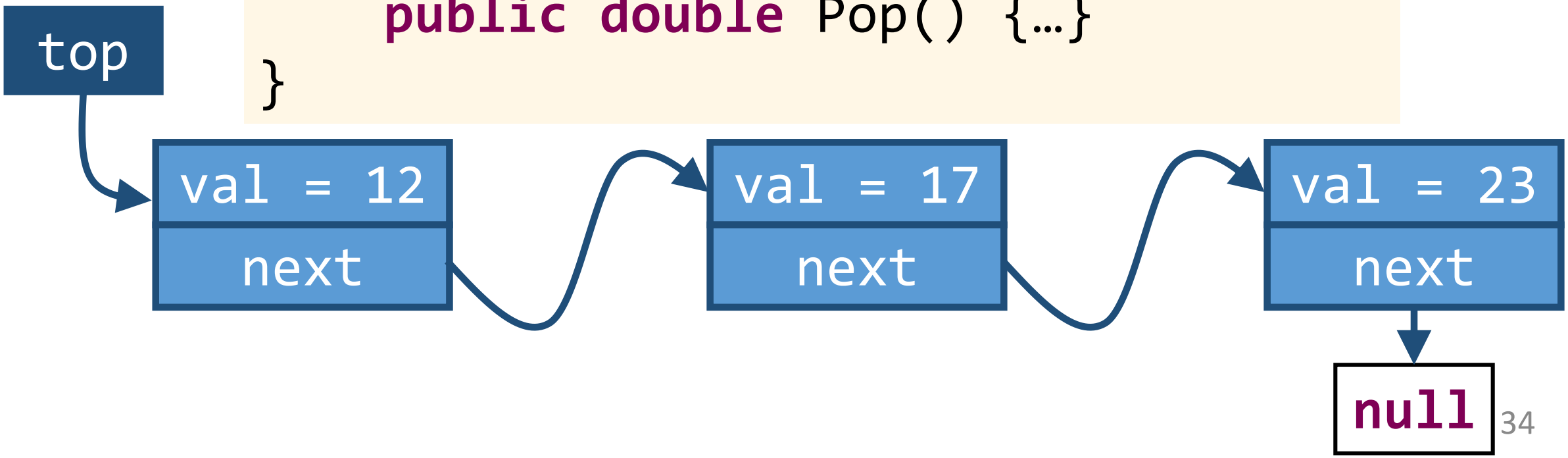
- Whiteboard

Stapel (Stack)

- LIFO: Last In First Out
- Kann mittels verkettete Liste implementiert werden
- Funktionalität:
  - Knoten vorne einfügen mittels Push
  - Knoten vorne herausnehmen mittels Pop

Stapel (Stack): Code

```
public class Stack {
    private Node top = null;
    public void Push(double val) {…}
    public double Pop() {…}
}
```



Beispiel Implementation von Push und Pop

- `Push`: Knoten vorne einfügen
- `Pop`: Knoten vorne herausnehmen und Wert zurückgeben
- Whiteboard

Prüfung 08.2014 Aufgaben 8a & 8b

- Whiteboard

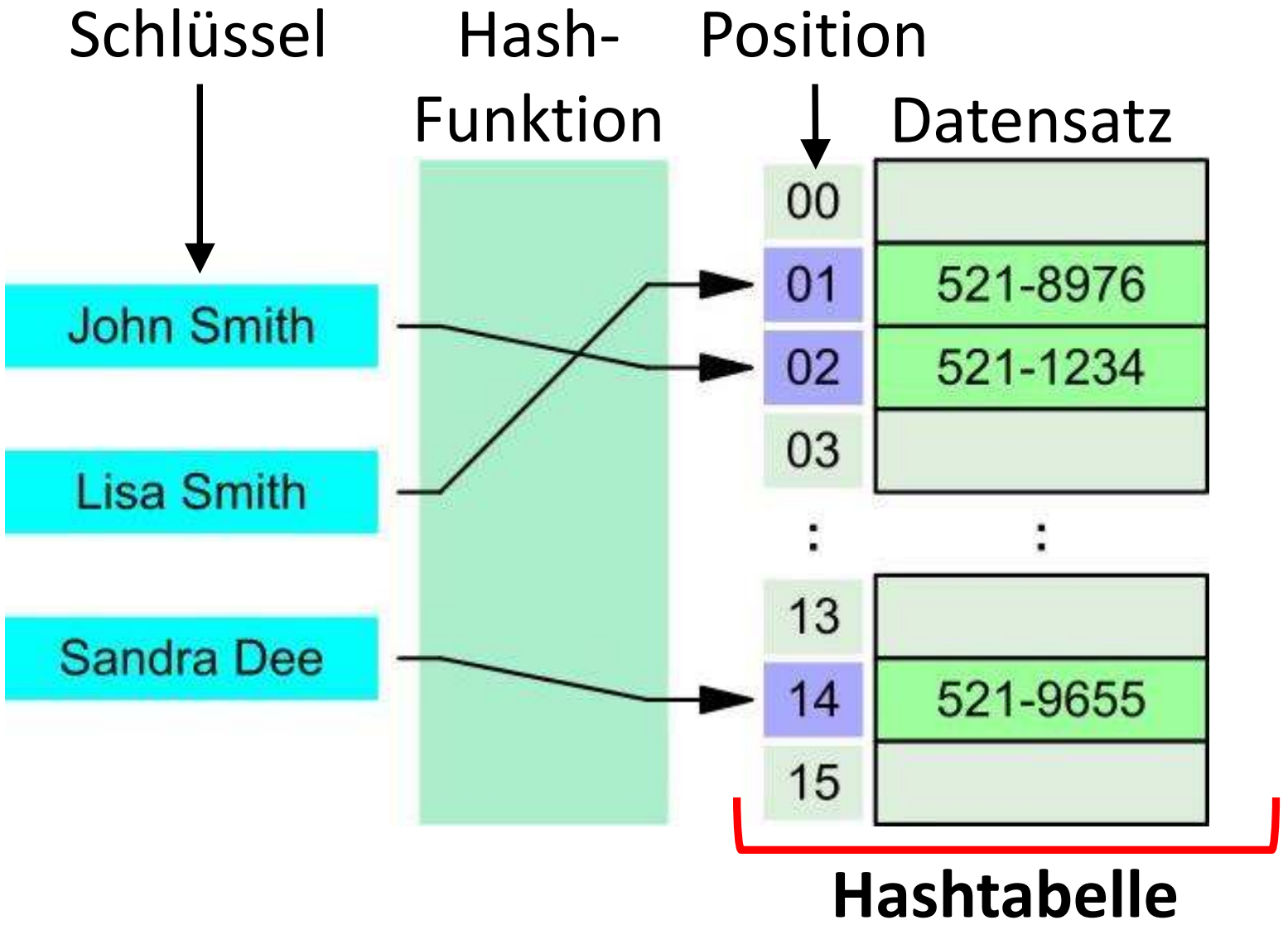
Hashtabelle

- Objekte werden in einer Hashtabelle gespeichert
- Position der Objekte in Hashtabelle wird mittels Hashfunktion berechnet
- Einfügen, Löschen und Suchen von Objekten in $O(1)$ Zeit (sehr schnell!)

Hashfunktion

- Mathematische Funktion, die Position eines Objektes in der Hashtabelle berechnet
- Zum Berechnen der Position wird Schlüssel benötigt, der Objekt eindeutig identifiziert
  - Beispiel: Beim Telefonbuch wäre der Schlüssel der Name der Person im Telefonbuch
- Problem: Hashfunktionen können Konflikte generieren, d.h. zwei verschiedene Schlüssel können auf gleiche Position zeigen

Beispiel Telefonbuch



Bildquelle: Wikipedia

Aufbau einer Hashtabelle

1. Array für Schlüssel und Daten
2. Hashfunktion zur Berechnung des Array-Index (Position) der einzufügenden Objekten
3. Datenzugriff auf Array
   Frage: Was geschieht, wenn wir auf einer Position bereits einen Eintrag haben (Kollision)?

Lineares Sondieren

- Bei einer Kollision wird linear nach dem nächsten freien Platz in der Hashtabelle gesucht
  - Position wird immer um 1 erhöht, bis ein freier Platz gefunden wird
- Haben wir einen freien Platz gefunden, so wird der Eintrag an der neuen Position gespeichert

Zufallszahlen

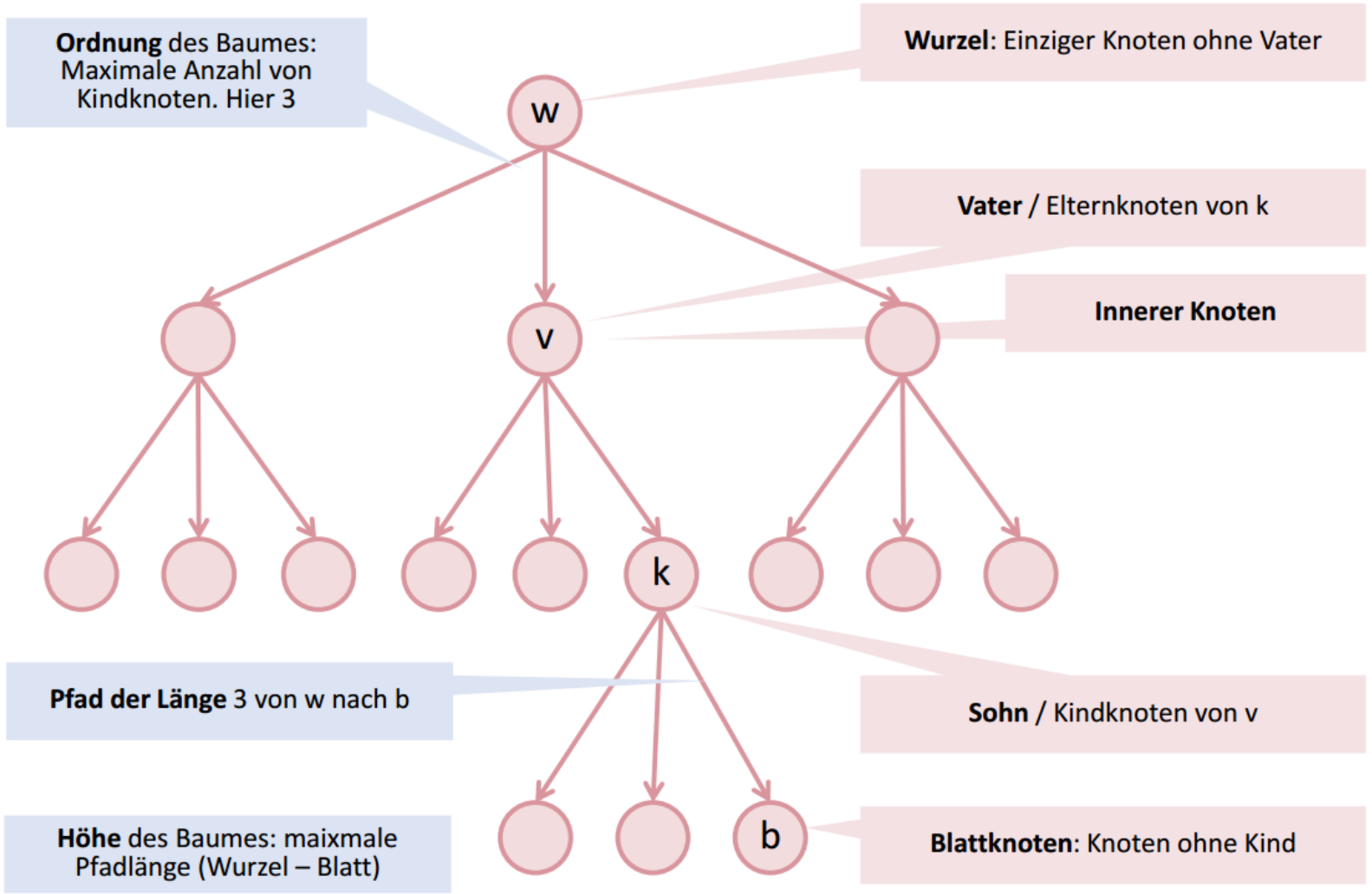
- `Math.random()` generiert eine Zufallszahl im Intervall $[0,1)$
- Zufallszahl im Intervall $\{a, a+1, a+2, \ldots, b\}$ generieren:

```
int r = (int)(Math.random() * (b + 1 - a) + a);
```

- Zufallszahl im Intervall $\{0,1,2, \ldots, k\}$ generieren:

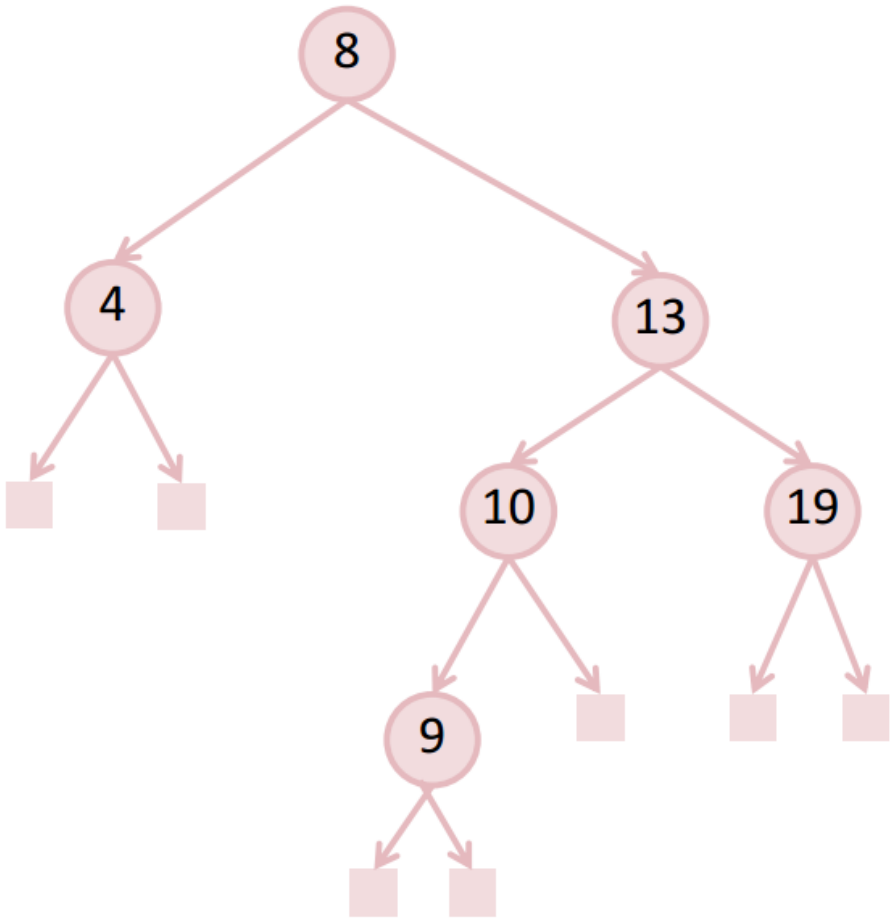
```
int r = (int)(Math.random() * (k + 1));
```

Bäume



Bildquelle:
Informatik II Vorlesung

Binärer Suchbaum



- Baum der Ordnung 2
- Schlüssel im linken Teilbaum kleiner als am Knoten
- Schlüssel im rechten Teilbaum grösser als am Knoten

Bildquelle:
Informatik II Vorlesung

Bäume: Datenstruktur

```
public class SearchNode {
    int key;
    SearchNode left;
    SearchNode right;
    SearchNode(int k){
        key = k;
        left = right = null;
    }
}
```

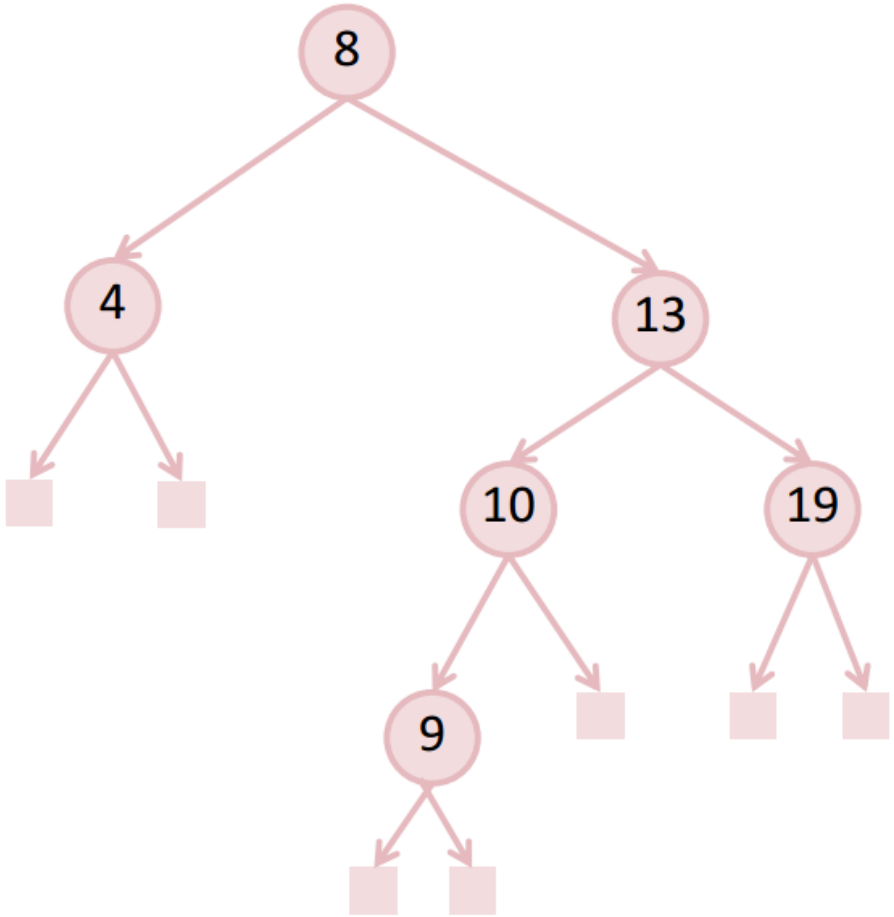
Baum durchsuchen

```
public SearchNode Search (int k){
    SearchNode n = root;
    while (n != null && n.key != k){
        if (k < n.key) n = n.left;
        else n = n.right;
    }
    return n;
}
```

Bäume: Knoten einfügen

1. Traversiere Baum bis zum passenden Blattknoten
2. Ersetze Blattknoten mit Knoten

Beispiel Knoten einfügen



- Knoten 14 einfügen

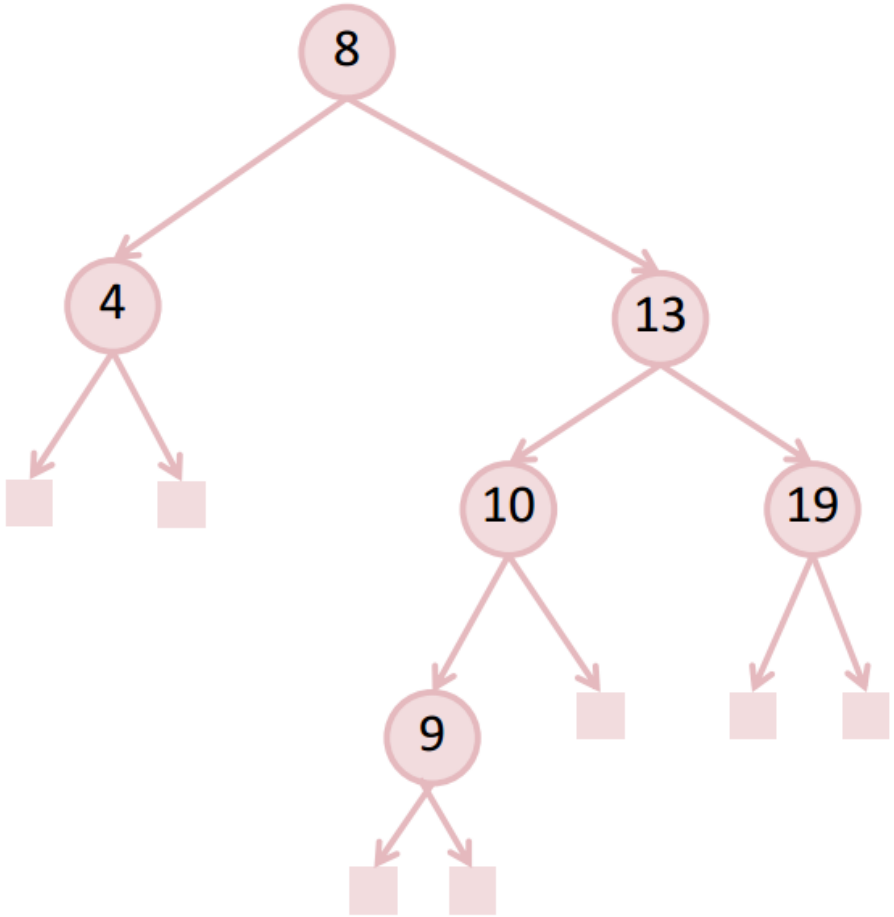
Bildquelle:
Informatik II Vorlesung

Bäume: Knoten löschen

3 Fälle sind zu unterscheiden:

1. Knoten hat keine Kinder
   - Knoten einfach löschen
2. Knoten hat nur ein Kind
   - Knoten durch Kind ersetzen
3. Knoten hat zwei Kinder
   - Knoten durch symmetrischen Nachfolger ersetzen

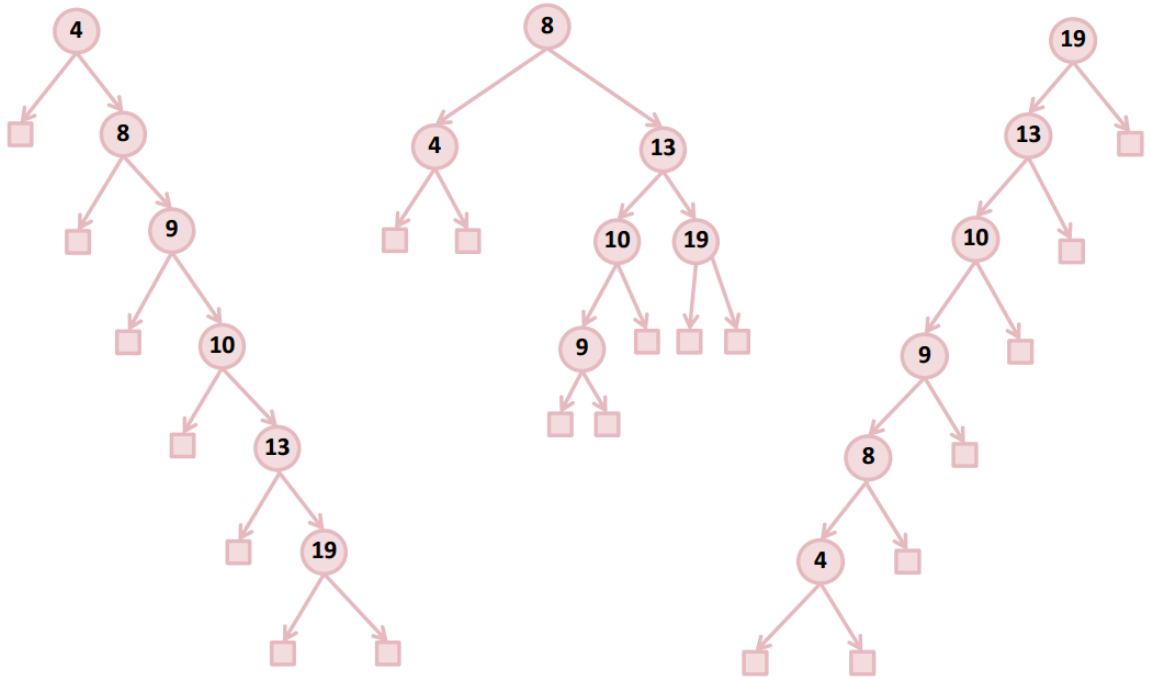
Symmetrischer Nachfolger



- Der symmetrische Nachfolger eines Knoten ist der Knoten im rechten Teilbaum, welcher am weitesten links steht
- Beispiel: Der symmetrische Nachfolger von 8 ist 9

Bildquelle:
Informatik II Vorlesung

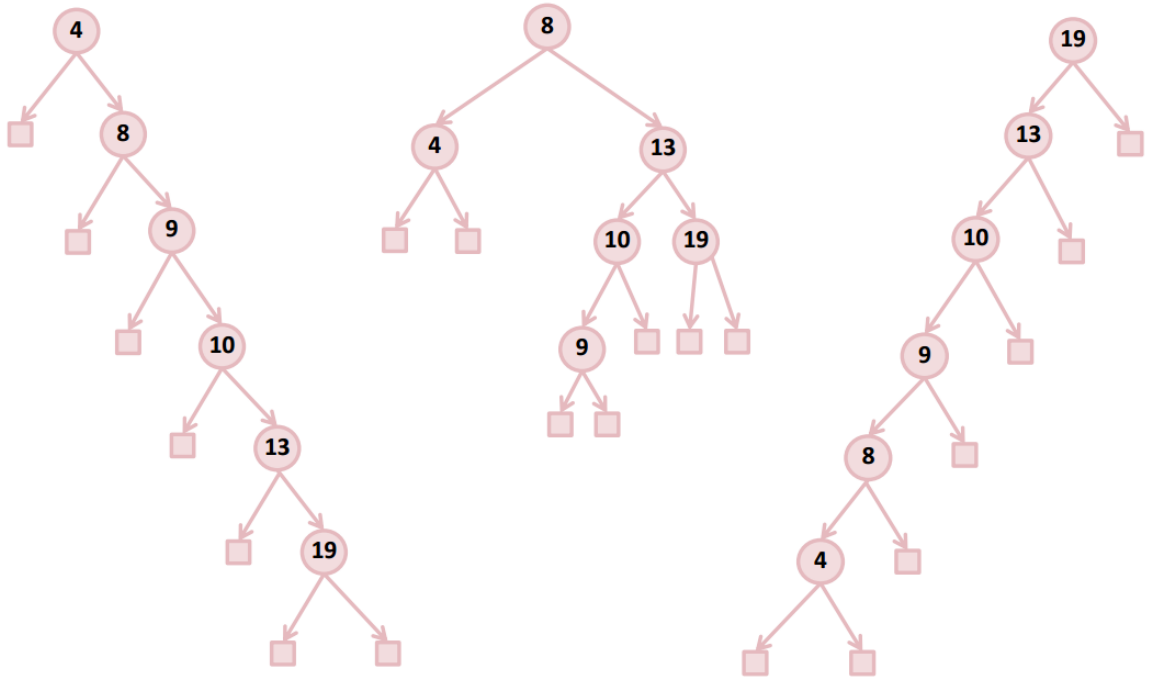
Degenerierte Bäume



- Je nach Einfügereihenfolge können binäre Bäume zu Listen degenerieren
- Frage: Laufzeit von Suchen / Einfügen / Löschen im schlimmsten Fall bei binären Bäumen?

Bildquelle:
Informatik II Vorlesung

Degenerierte Bäume



- Je nach Einfügereihenfolge können binäre Bäume zu Listen degenerieren
- Frage: Laufzeit von Suchen / Einfügen / Löschen im schlimmsten Fall bei binären Bäumen?

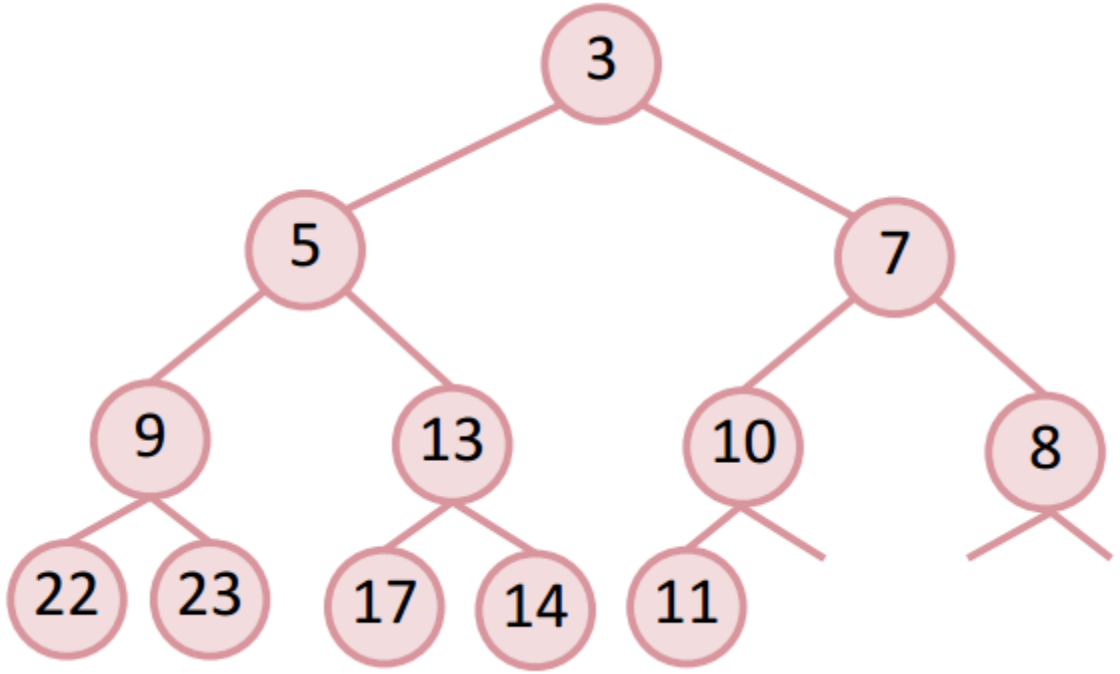
$$O(n)$$

Bildquelle:
Informatik II Vorlesung

Balancierte Bäume

- Komplexität von Suchen / Einfügen / Löschen eines Knotens bei binären Suchbäumen im Mittel $O(log_2(n))$
  - Im schlimmsten Fall (bei degeneriertem Baum): $O(n)$
- Balancierte Bäume: verhindern Degenerierung durch Balancierung des Baumes
  - Im schlimmsten Fall auch $O(log_2(n))$ (wesentlich besser!)
  - Balancierung: garantiere, dass Binärbaum mit n Knoten stets eine Höhe von $O(log_2(n))$

Heaps

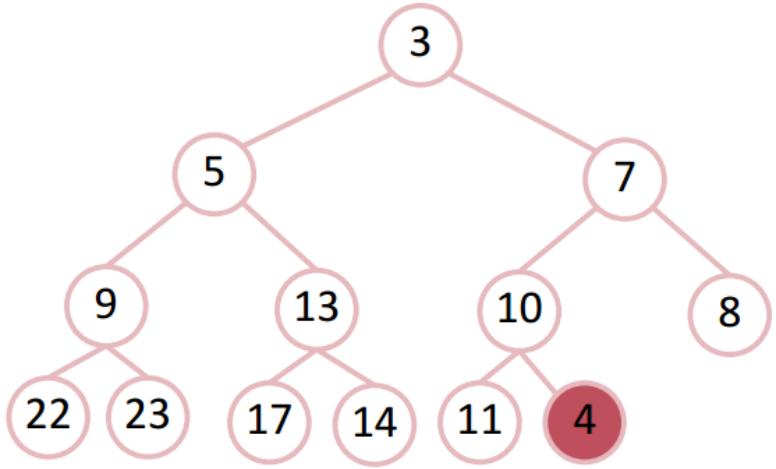


- Heap ist ein Binärbaum
- Min-Heap-Eigenschaft: Schlüssel eines Kindes ist immer grösser als der des Vaters
  - Max-Heap: Kinder-Schlüssel immer kleiner als Vater-Schlüssel
- Heap hat nur Lücken in der letzten Ebene; müssen alle rechts liegen

Bildquelle:
Informatik II Vorlesung

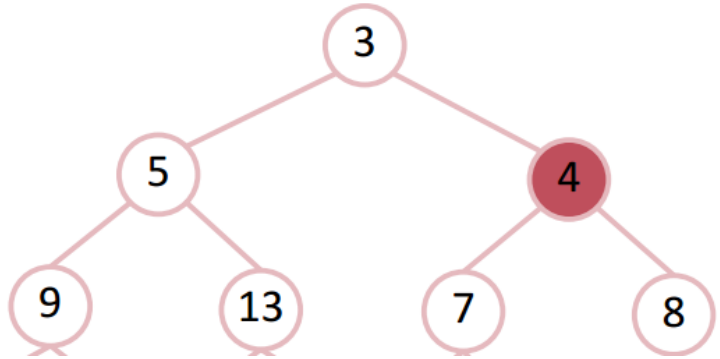
Heaps: Knoten einfügen

1. Füge neuen Knoten an ersten freien Stelle (verletzt Heap-Eigenschaft)



2. Stelle Heap-Eigenschaft wieder her durch sukzessives Aufsteigen des Knotens

$$O(\log(n))$$



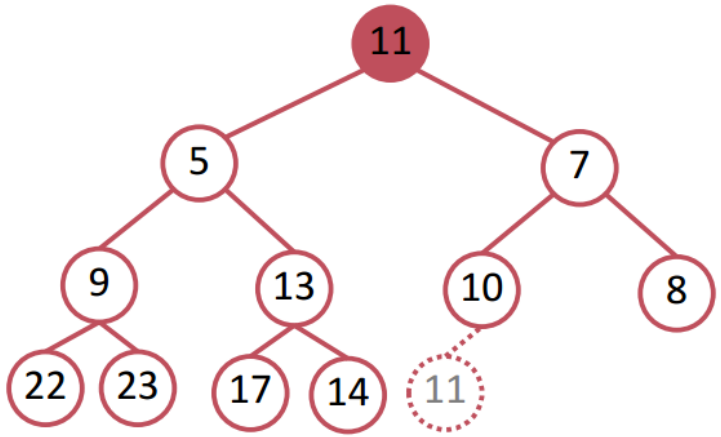
Bildquelle:
Informatik II Vorlesung

Heaps: Wurzelknoten

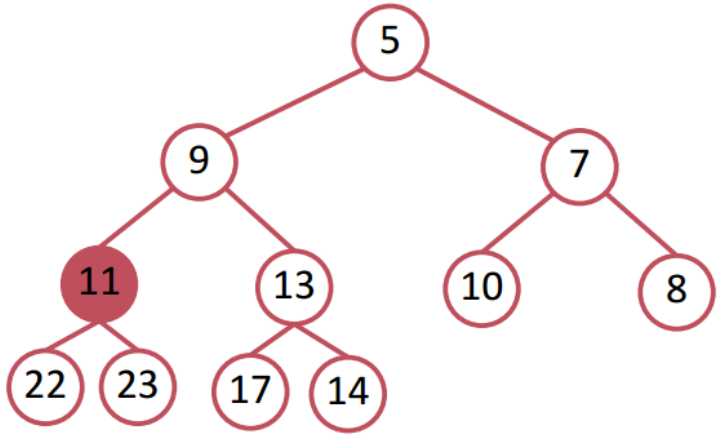
- Bei min-Heap ist der Wurzelknoten das kleinste Element
- Bei max-Heap ist der Wurzelknoten das grösste Element

Heaps: Wurzelknoten entfernen

1. Ersetze Wurzel durch den letzten Knoten



2. Lasse Wurzel nach unten sinken, um Heap-Eigenschaft wiederherzustellen



$O(\log(n))$

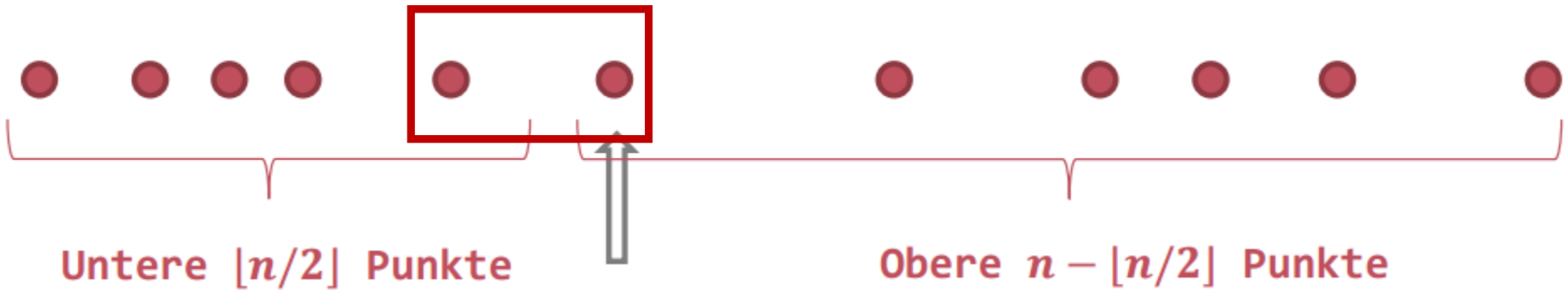
Bildquelle:
Informatik II Vorlesung

Prüfung 02.2016 Aufgaben 7a & 7b

- Whiteboard

Online Median

- Verwende das schnelle Auslesen, Extrahieren und Einfügen vom Minimum (bzw. Maximum) im Heap für den Online Median
- Median bildet sich aus Minimum der oberen Hälfte der Daten und / oder Maximum der unteren Hälfte



Online Median: Algorithmus

- Verwende Max-Heap um untere Hälfte und Min-Heap um obere Hälfte der Elemente zu speichern
- Füge Wert v in
  - Max-Heap, falls kleiner oder gleich Wurzelknoten ist
  - Min-Heap sonst
- Rebalancieren der beiden Heaps
  - Falls Max-Heap mehr als Hälfte der Elemente hat, dann platziere max(Max-Heap) (Wurzel) in Min-Heap
  - Falls Max-Heap weniger als Hälfte der Elemente hat, dann platziere min(Min-Heap) (Wurzel) in Max-Heap

$$O(\log(n))$$

Online Median: Berechnung Median

- Wenn Anzahl Elemente ungerade ist, dann ist Median gleich der min(Min-Heap)
- Wenn Anzahl Elemente gerade ist, dann ist der Median definiert als [max(Max-Heap) + min(Min-Heap)]/2

$$O(1)$$