

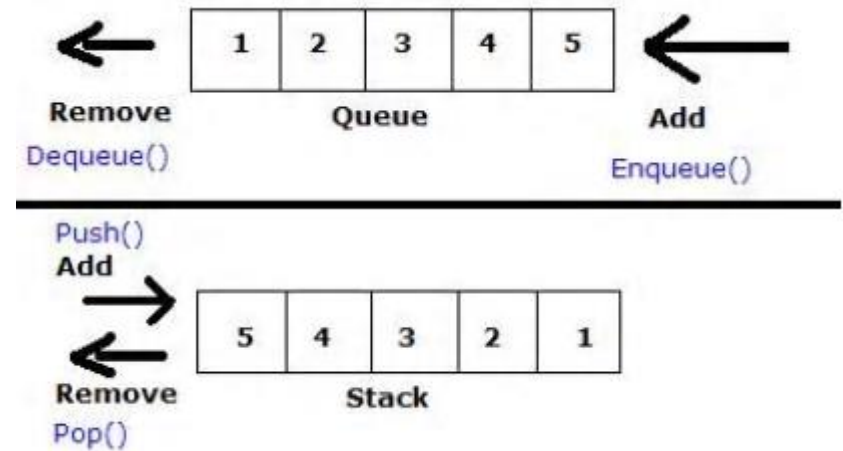
Priority Queues

Algorithmen und Datenstrukturen 2

- Grüne Farbe: Bitte im Script nachtragen

Priority Queues

- Zwischenlagerung für Elemente
- Geordnet nach Priorität
- Schneller Zugriff auf Min / Max Element
- Orientierungen
 - Min-Oriented Priority Queues
 - Max-Oriented Priority Queues



Priority Queue Interface

```
public interface MinPriorityQueue<K extends Comparable<? super K>> {  
    void add(K element); // fügt beliebiges Element hinzu  
    K min();              // Wert des minimalen Elements (Queue wird nicht verändert)  
    K removeMin();        // entfernt das derzeit minimale Element und gibt es zurück  
    int size();           // Anzahl aktuell in der Queue gespeicherte Elemente  
}
```

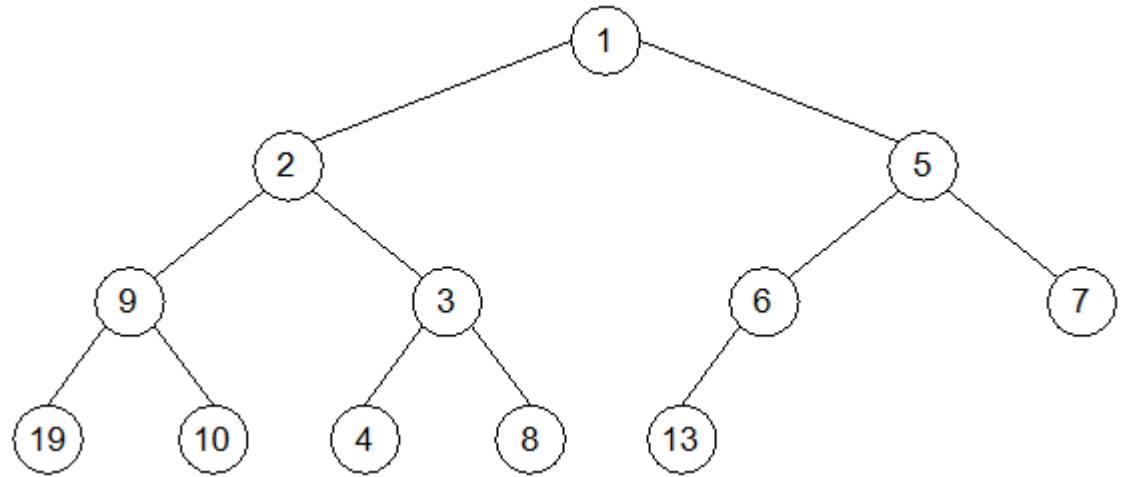
- Elemente müssen in Ordnungsrelation stehen nach Priorität (Comparable)
 - `x.compareTo(y)`, liefert int zurück:
 - `> 0`: $x > y$
 - `== 0`: $x == y$
 - `< 0`: $x < y$

Priority Queues mit bekannten Datenstrukturen (Seite 2)

Operationen	Array		LinkedList		Baum	
	Sortiert <i>(min am Ende)</i>	Unsortiert	Sortiert <i>(min am Anfang)</i>	Unsortiert	Allgemein	AVL
add(element)						
min()						
removeMin()						

Heap Datenstruktur (Kap. 5.5)

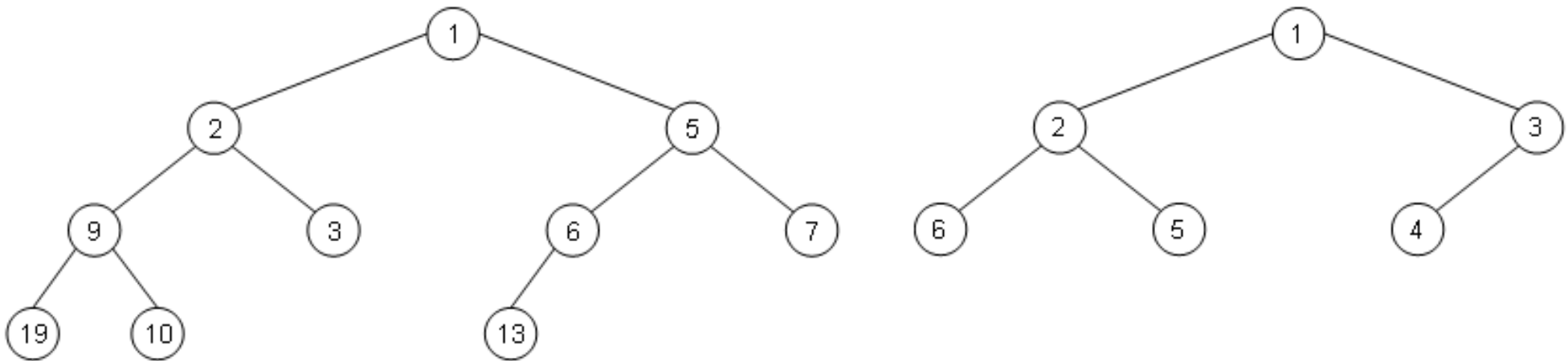
- Sehr gut geeignet für PQ
- Bild zeigt einen **Min-Heap**
- Invariante Eigenschaften:
 - Binärer Baum
 - Struktur-Eigenschaft:
 - Ordnungs-Relation Eigenschaft:



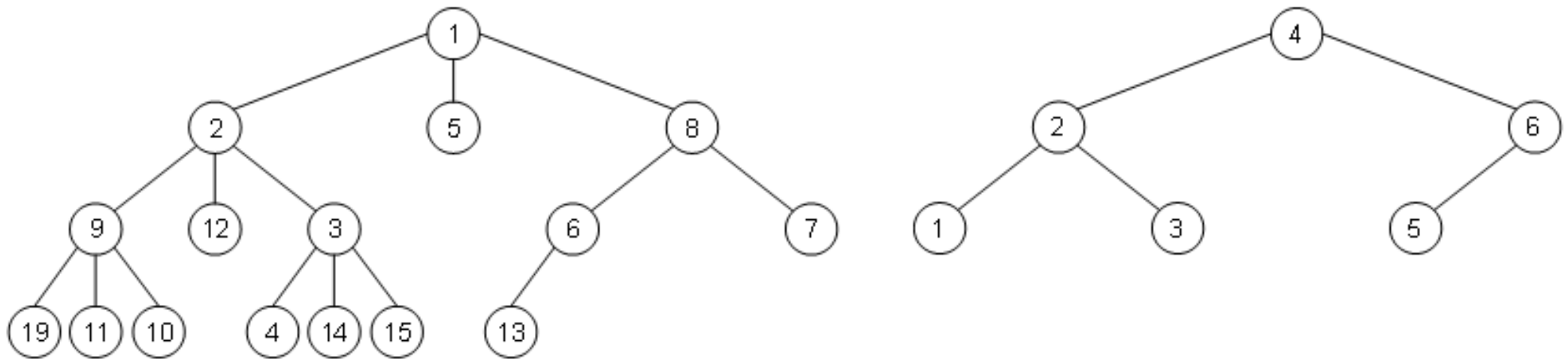
Beispiele: Heaps und nicht-Heaps (Seite 3)

- Geben Sie für jedes Beispiel an:
 - Ist es ein Heap?
 - Falls ja:
 - ein Min-Heap oder
 - ein Max-Heap
 - Falls nein:
 - Welche invariante Eigenschaft wurde verletzt?

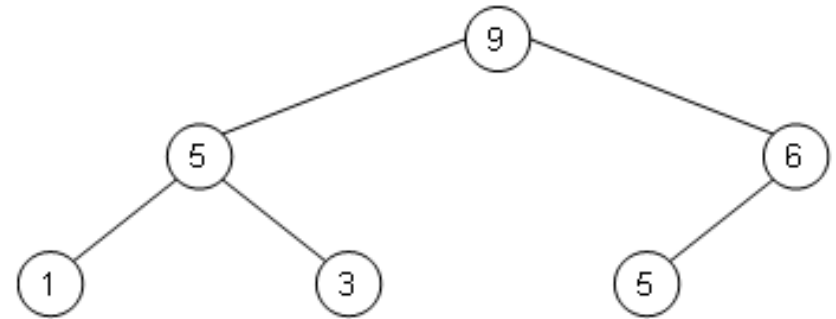
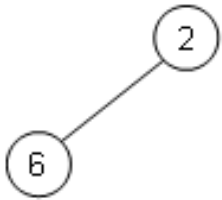
Heaps-Beispiele (1)



Heaps-Beispiele (2)

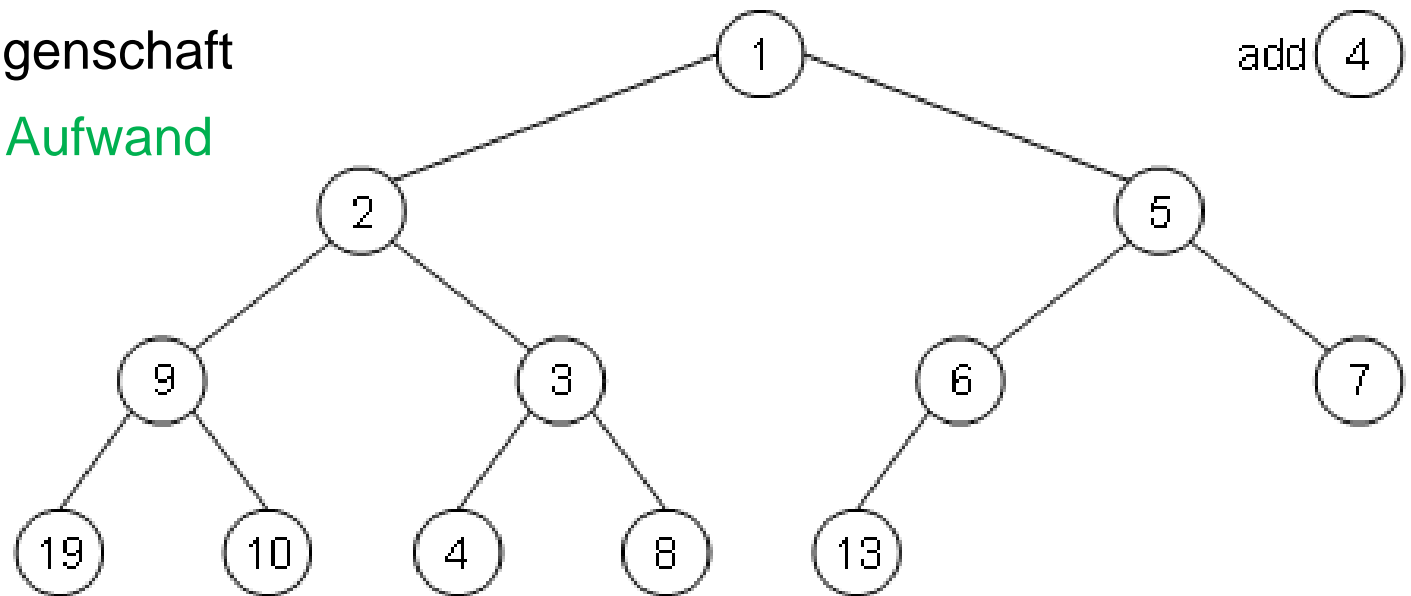


Heaps-Beispiele (3)



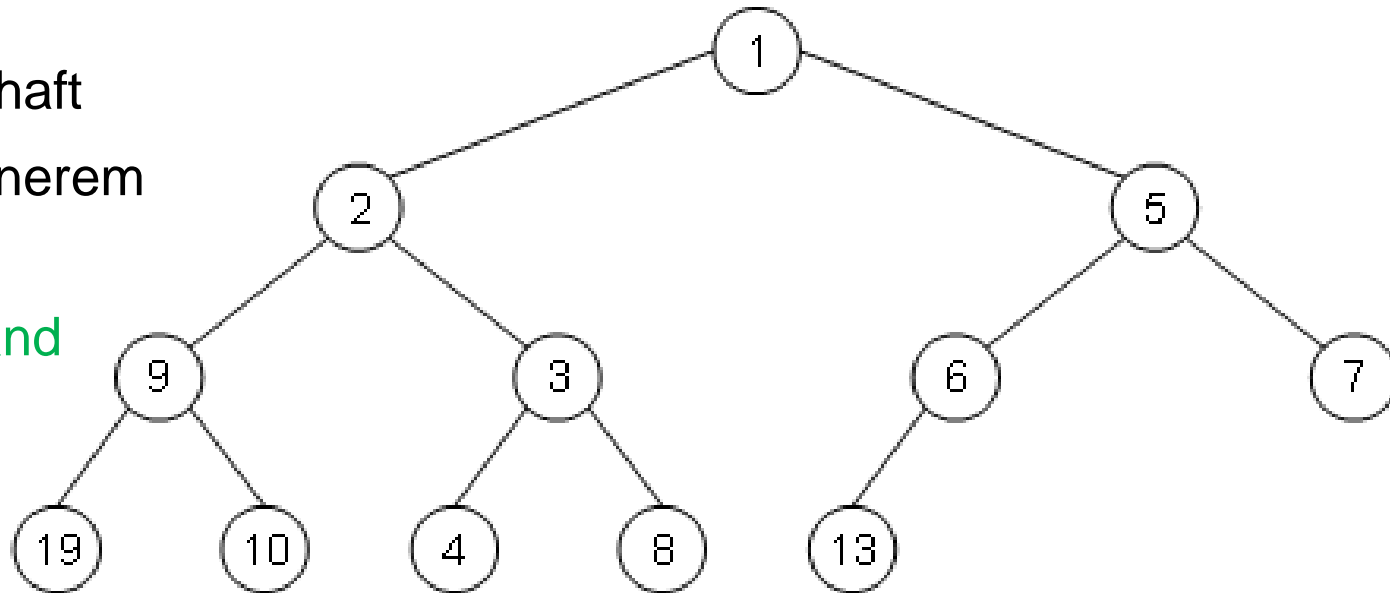
Operationen auf Heaps (Kap 5.6)

- **Hinzufügen von Elementen**
 - Invariante Eigenschaften müssen erhalten bleiben
 - Binärer Baum
 - Struktur-Eigenschaft
 - Ordnungs-Eigenschaft
 - Asymptotischer Aufwand
 - $O(\log n)$



Operationen auf Heaps (Kap 5.6)

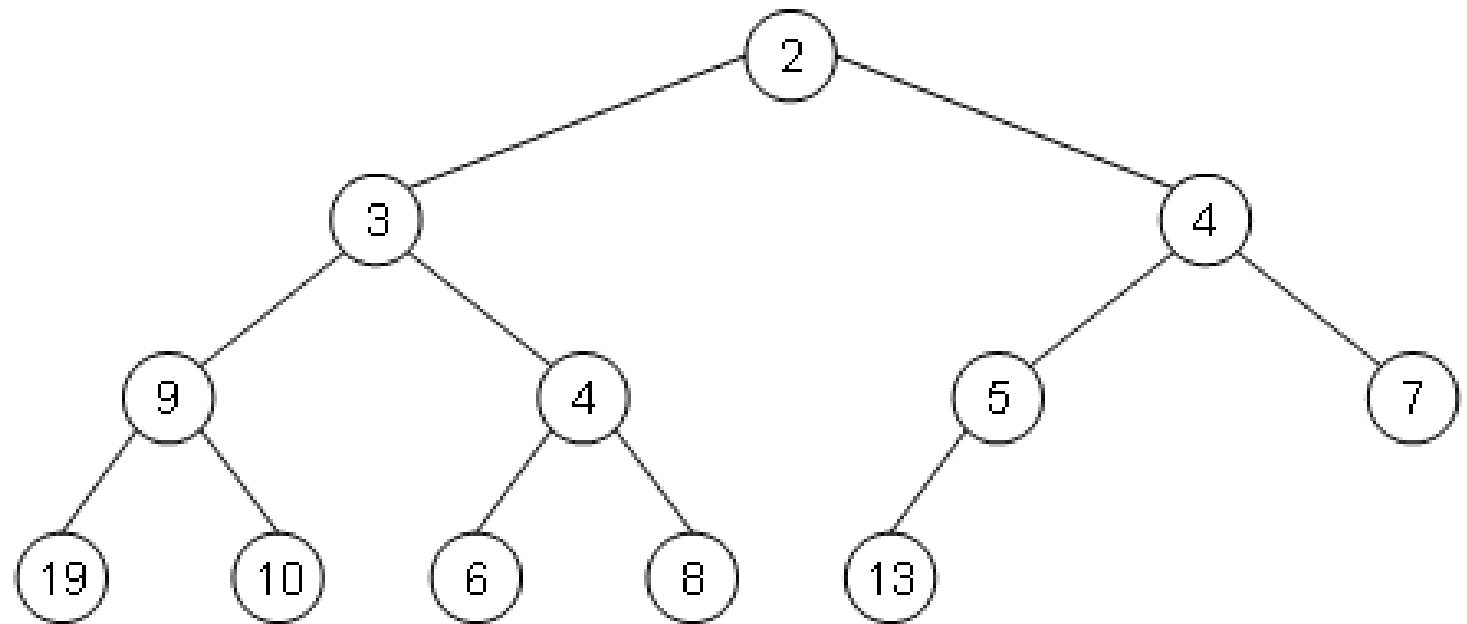
- **Entfernen des kleinsten Elements (*removeMin()*)**
 - Invariante Eigenschaften müssen erhalten bleiben
 - Struktur-Eigenschaft
 - Binärer Baum
 - Ordnungs-Eigenschaft
 - * Immer mit kleinerem vertauschen
 - Asymptotischer Aufwand
 - $O(\log n)$



Beispiel im Script (Seite 4)

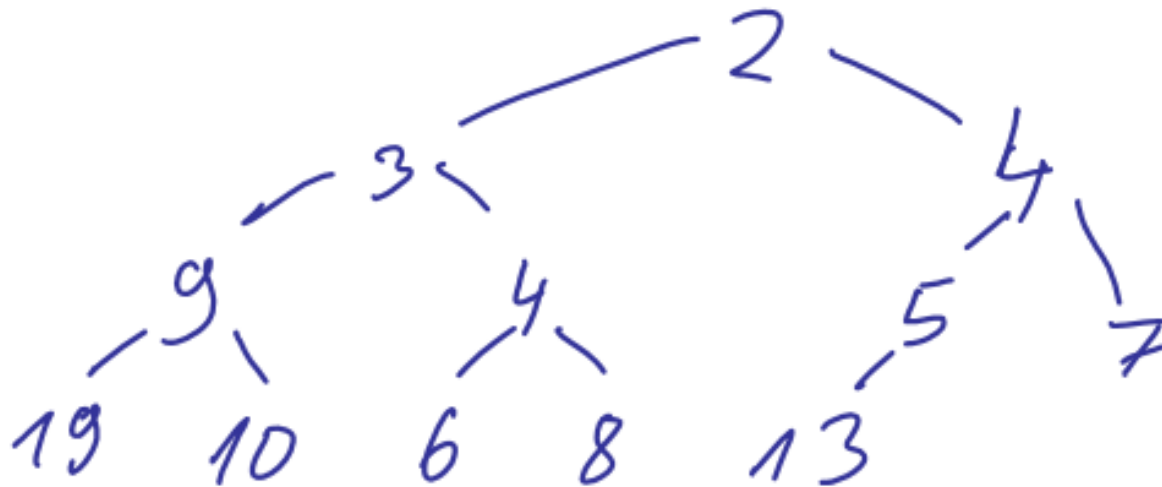
Beispiel

add(2);
removeMin();



Beispiel im Script (Seite 4)

Lösung:

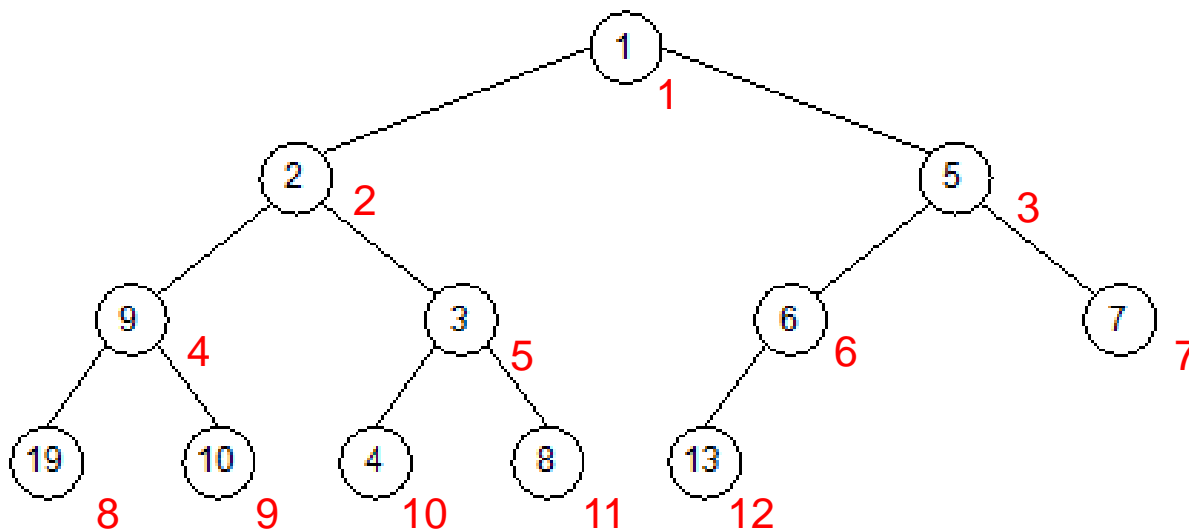


Komplexitätsanalyse von Heaps

Operationen	Array		LinkedList		Baum		
	Sortiert (<i>min am Ende</i>)	Unsortiert	Sortiert (<i>min am Anfang</i>)	Unsortiert	Allgemein	AVL	Heap
add(element)	$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(\log n)$	
min()	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(\log n)$	
removeMin()	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(\log n)$	

Heap-Implementation in Array

- Heaps können elegant in Arrays implementiert werden
- Version mit Start-Index 1:



Vaterknoten von i:

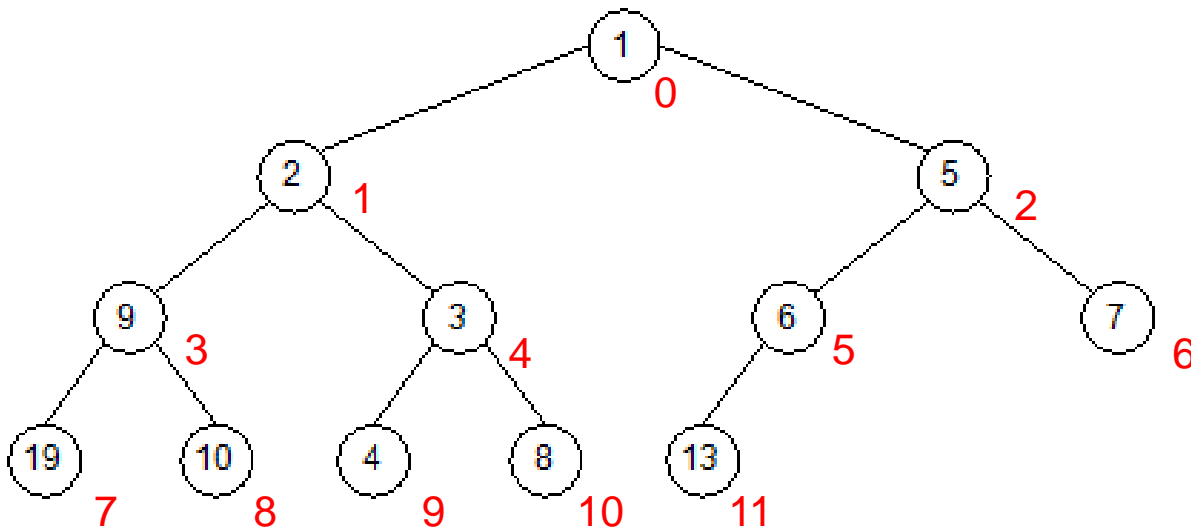
Linker Nachfolger von i:

Rechter Nachfolger von i:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Element:		1	2	5	9	3	6	7	19	10	4	8	13	

Heap-Implementation in Array

- Version mit Start-Index 0:



<i>Index</i>	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Element:	1	2	5	9	3	6	7	19	10	4	8	13		

Heap-Implementation in Array

- $F_1(x)$: Berechnung Vorgänger / Nachfolger ab Index 1
- $F_0(x)$: Berechnung Vorgänger / Nachfolger ab Index 0
- $F_0 = F_1(x+1) - 1$

	Wurzel bei Index 1 ($f_1(x)$)	Wurzel bei Index 0 ($f_0(x)$)
Vaterknoten von i:	$i / 2$ (int div)	$(i + 1) / 2 - 1 = (i - 1) / 2$
Linker Nachfolger von i:	$2i$	$2 (i + 1) - 1 = 2i + 1$
Rechter Nachfolger von i:	$2i + 1$	$2 (i + 1) + 1 - 1 = 2i + 2$

Selbststudium

1. Schaut, ob ihr bis hierher alles verstanden habt (Heap-Aufbau, Operationen, Array-Implementierung). Bei Unklarheiten: Fragen
2. Löst die folgenden Aufgaben:
 1. **Script Seite 1: Kapitel 5.3 lesen und Anwendungsbeispiel (Ansatz 1 (Komplexitätsanalyse) und Ansatz 2 (Implementation auf Papier))**
 2. Implementieren eines Min-Heaps. Im Script auf den Seiten 5 und 6 gibt es Beschreibungen zu den Methoden: `min()`, `siftUp(index)`, `siftDown(index)`, `add(Element)`, `removeMin()`. Ihr könnt dies zuerst versuchen von Hand zu lösen oder die Anleitung verwenden, um gleich **Aufgabe 1 im Selbststudium** (Eclipse-Projekt HeapTest) zu implementieren. Hinweis: Implementiert zuerst `siftUp` und `siftDown`, weil diese Methoden die Grundlagen für das `add` und `remove` sind.