Prioritätswarteschlangen

- Prioritätswarteschlangen
- Bäume und Heaps
- Darstellung von Heaps
- Algorithmen auf Heaps
- Einen Heap bauen
- Heapsort

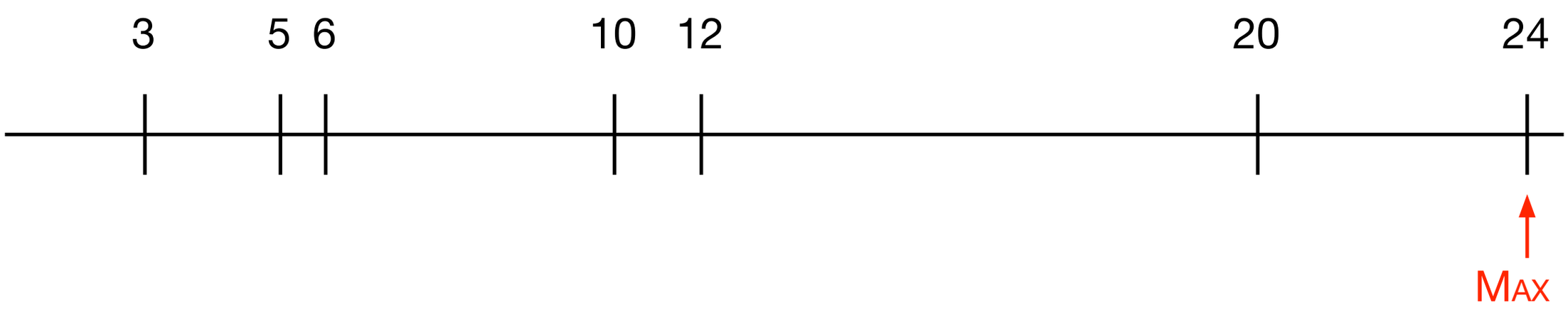
Holger Dell
Folien adaptiert von Philip Bille

Prioritätswarteschlangen

- Prioritätswarteschlangen
- Bäume und Heaps
- Darstellung von Heaps
- Algorithmen auf Heaps
- Einen Heap bauen
- Heapsort

Prioritätswarteschlangen

- Prioritätswarteschlangen. Verwalte eine dynamische Menge S und unterstütze die folgenden Operationen. Jedes Element hat Schlüssel x.key und Daten x.data.
  - MAX(): liefere das Element mit dem größten Schlüssel.
  - EXTRACTMAX(): liefere und entferne das Element mit dem größten Schlüssel.
  - INCREASEKEY(x, k): setze x.key = k. (wir nehmen an, dass k ≥ x.key)
  - INSERT(x): füge x in die Menge S ein.



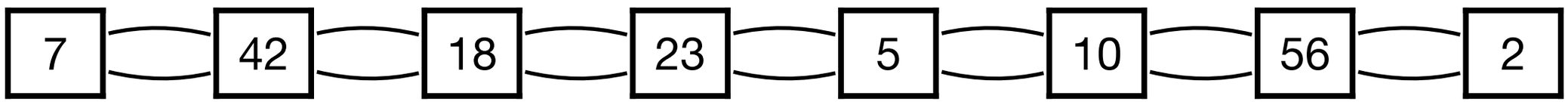
Prioritätswarteschlangen

- Anwendungen.
  - Scheduling
  - Kürzeste Wege in Graphen (Dijkstras Algorithmus)
  - Minimaler Spannbaum in Graphen (Jarník–Prims algorithm)
  - Kompression (Huffman-Codierungen)
  - …

- Frage. Wie würden wir die Datenstruktur mit den uns bereits bekannten Techniken implementieren?

Prioritätswarteschlangen

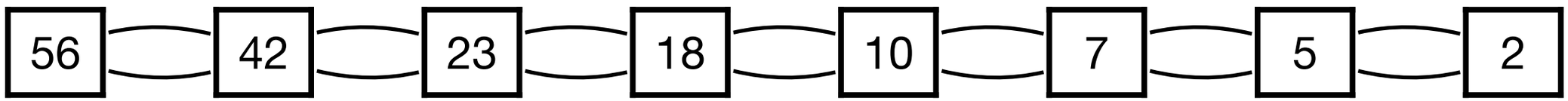
- Lösung 1: Verwalte S in einer doppelt verketteten Liste.



- MAX(): lineare Suche nach dem größten Schlüssel.
- EXTRACTMAX(): lineare Suche nach dem größten Schlüssel. Entferne und liefere das Element.
- INCREASEKEY(x, k): setze x.key = k.
- INSERT(x): füge x an den Anfang der Liste ein (hierbei nehmen wir an, dass das Element noch nicht in S vorkommt).
- Zeit.
  - MAX und EXTRACTMAX brauchen Zeit O(n), wobei n = |S|.
  - INCREASEKEY und INSERT brauchen Zeit O(1).
- Platz.
  - O(n).

Prioritätswarteschlangen

- Lösung 2: Verwalte S in einer sortierten doppelt verketteten Liste.



- MAX(): liefere das erste Element.
- EXTRACTMAX(): liefere das erste Element und lösche es aus der Liste.
- INCREASEKEY(x, k): setze x.key = k. Lineare Suche, um x an richtige Stelle zu bringen.
- INSERT(x): lineare Suche, um x an der richtigen Stelle einzufügen.
- Zeit.
  - MAX und EXTRACTMAX in Zeit O(1).
  - INCREASEKEY und INSERT in Zeit O(n).
- Platz.
  - O(n).

Prioritätswarteschlangen

Datenstruktur	MAX	EXTRACTMAX	INCREASEKEY	INSERT	Space
verkettete Liste	O(n)	O(n)	O(1)	O(1)	O(n)
sortierte verkettete Liste	O(1)	O(1)	O(n)	O(n)	O(n)

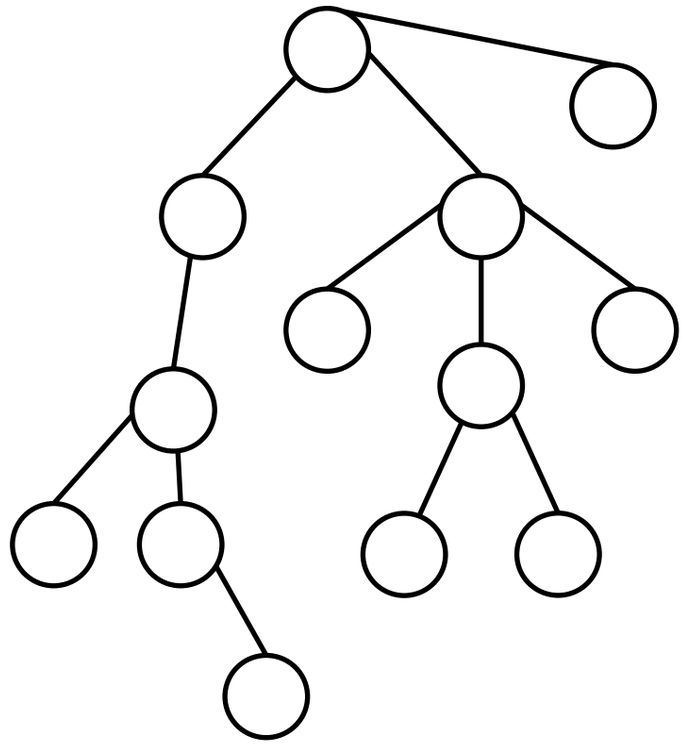
- Frage. Geht das signifikant besser?

Prioritätswarteschlangen

- Prioritätswarteschlangen
- Bäume und Heaps
- Darstellung von Heaps
- Algorithmen auf Heaps
- Einen Heap bauen
- Heapsort

Bäume

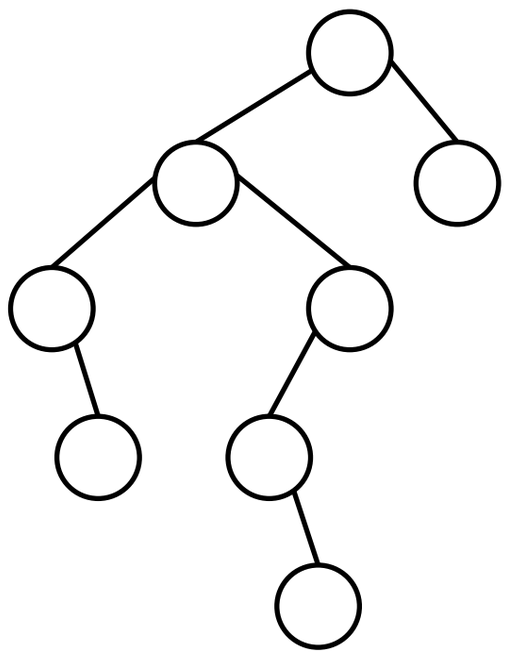
- gewurzelte Bäume.
  - Knoten (oder Ecken) verbunden durch Kanten.
  - Verbunden und azyklisch.
  - Designierter Wurzelknoten.
  - Spezialfall eines Graphen.



- Terminologie.
  - Kinder, Eltern, Nachfahre, Vorfahre, Blätter, interne Knoten, Pfad, ...
- Tiefe und Höhe.
  - Tiefe von v = Länge des Pfads von v zur Wurzel.
  - Höhe von v = Länge des Pfads von v zu tiefstem Blatt unter v.
  - Tiefe von T = Höhe von T = Länge eines längsten Wurzel-Blatt Pfads.

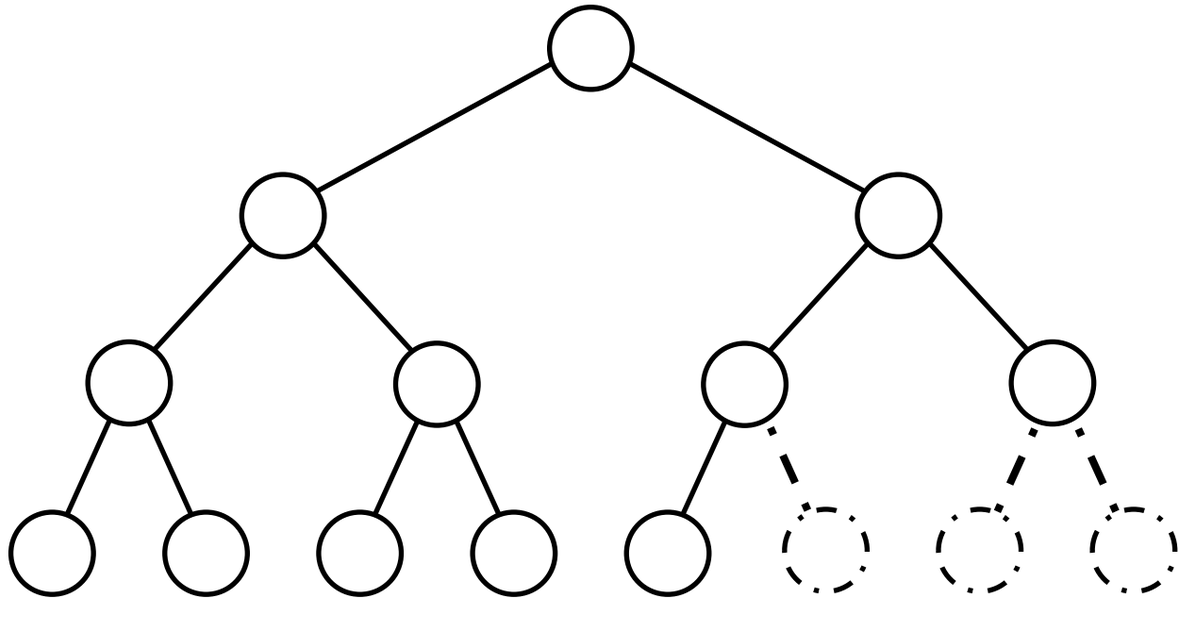
Bäume

- Binärbaum.
  - gewurzelter Baum.
  - Jeder Knoten hat höchstens zwei Kinder, nämlich das linke Kind und das rechte Kind.



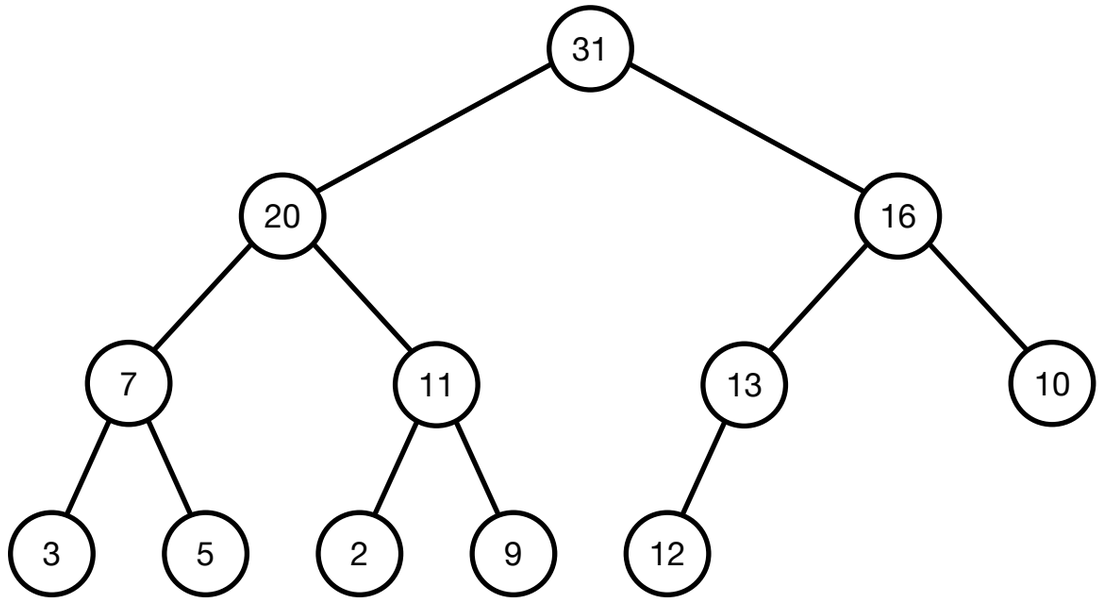
- Vollständiger Binärbaum. Binärbaum, in dem alle Ebenen voll sind.
- Fast vollständiger Binärbaum. Vollständiger Binärbaum, bei dem man aber 0 oder mehr von den rechtesten Blättern gelöscht hat.

- Lemma. Die Höhe eines fast vollständigen Binärbaums mit n Knoten ist Θ(log n).
- Beweis. Siehe Übungen.



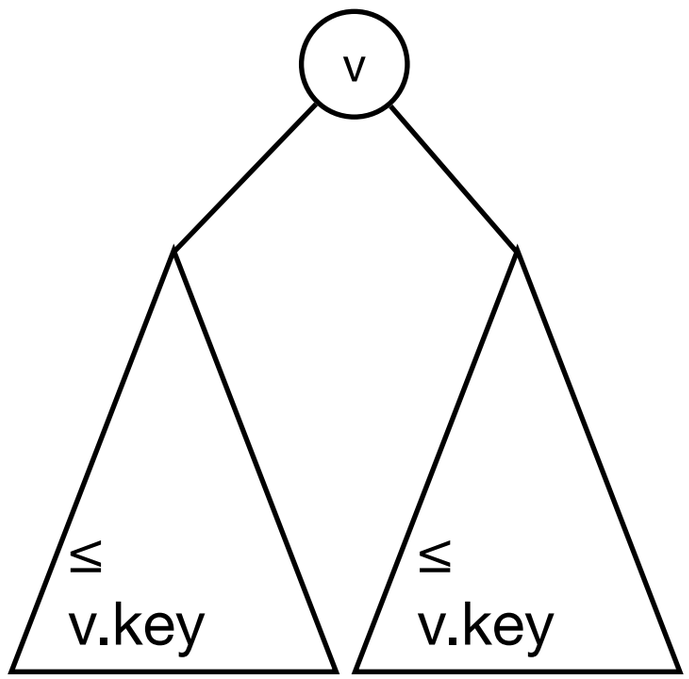
Heaps

- Heaps. Fast vollständiger Binärbaum. Alle Knoten speichern ein Element und der Baum erfüllt die Heap-Eigenschaft.



- Heap-Eigenschaft.
  - Für alle Knoten v gilt:
    - alle Schlüssel im linken und rechten Unterbaum von v sind ≤ v.key.

- Max-Heap vs. Min-Heap.



Prioritätswarteschlangen

- Prioritätswarteschlangen
- Bäume und Heaps
- Darstellung von Heaps
- Algorithmen für Heaps
- Einen Heap bauen
- Heapsort

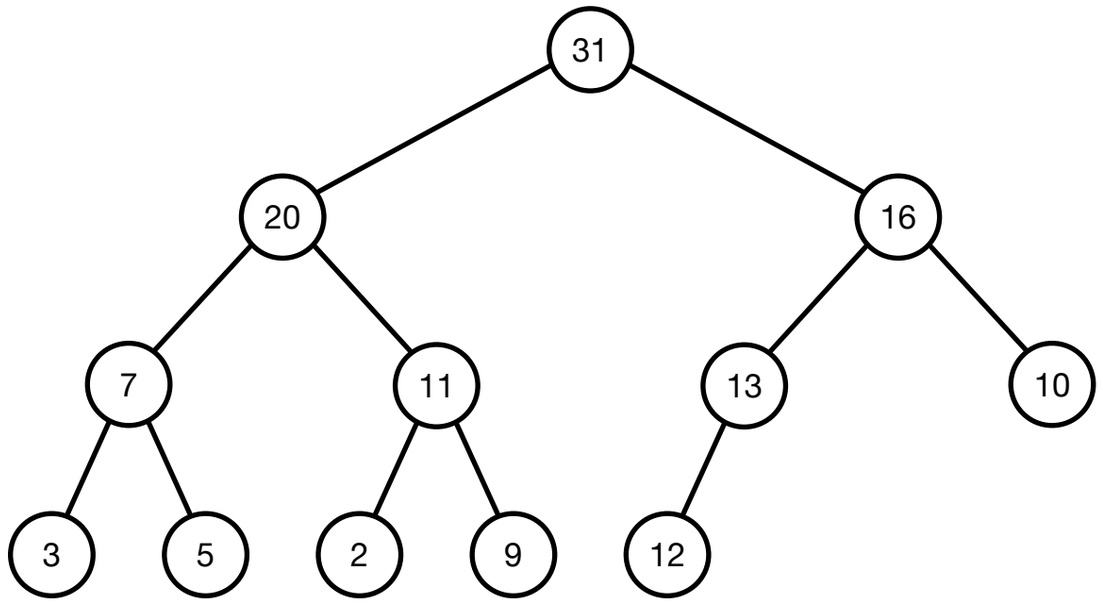
Heap

- Datenstruktur. Wir brauchen die folgenden Operationen zum Navigieren.
  - PARENT(x): liefere den Elternknoten von x.
  - LEFT(x) : liefere das linke Kind von x.
  - RIGHT(x): liefere das rechte Kind von x.

- Frage. Wie können wir einen Heap kompakt so darstellen, dass diese Navigationsoperationen schnell sind?

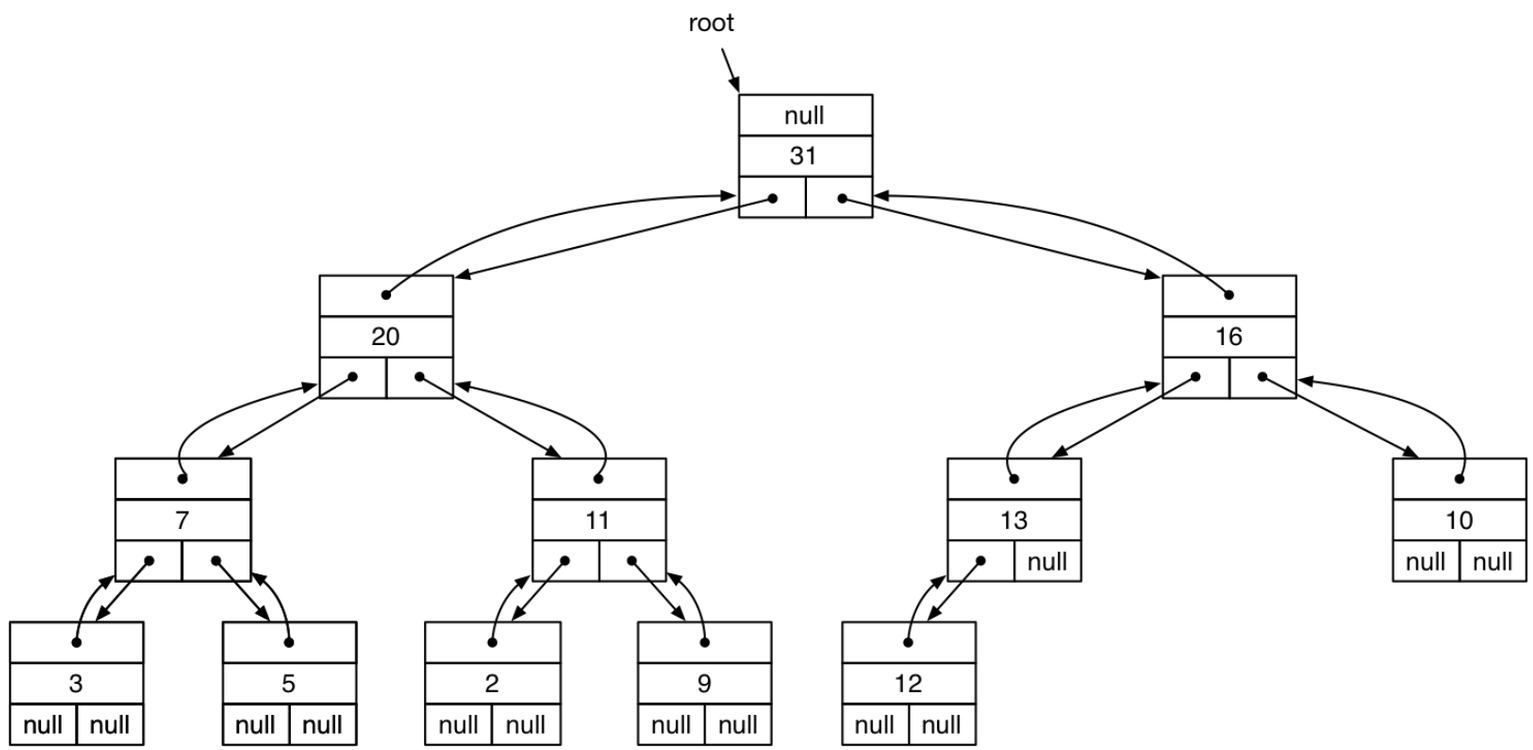
Heap

- Verkettete Darstellung. Jeder Knoten speichert
  - v.key
  - v.parent
  - v.left
  - v.right



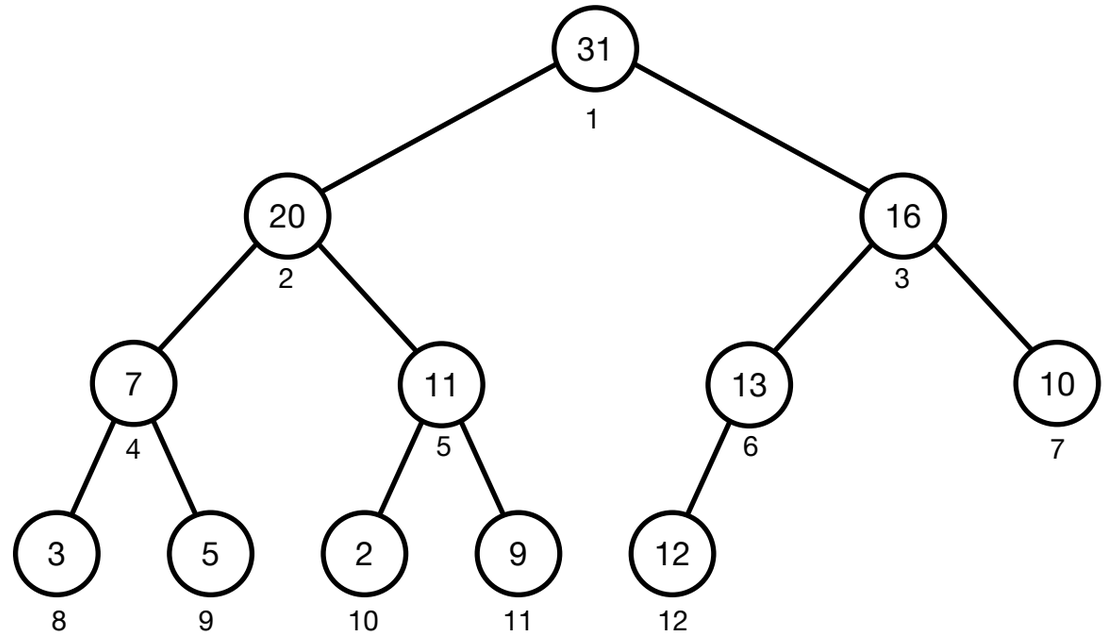
- PARENT, LEFT, RIGHT folgen den Zeigern.

- Zeit. O(1)
- Platz. O(n)

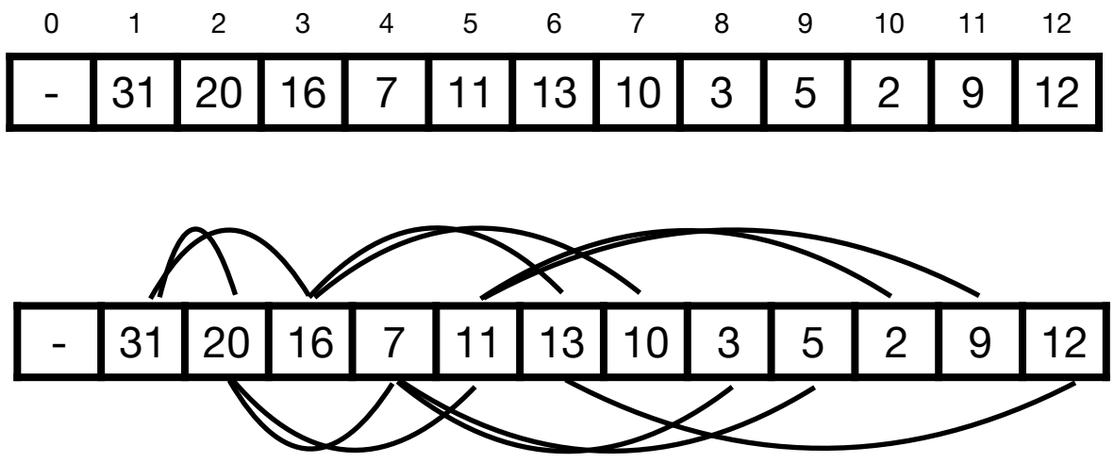


Heap

- Darstellung durch Feld.
  - Feld H[0..n]
  - H[0] unbenutzt
  - H[1..n] speichert Knoten Ebene für Ebene (*level order*).



- PARENT(x): liefere ⌊x/2⌋
- LEFT(x) : liefere 2x.
- RIGHT(x): liefere 2x + 1



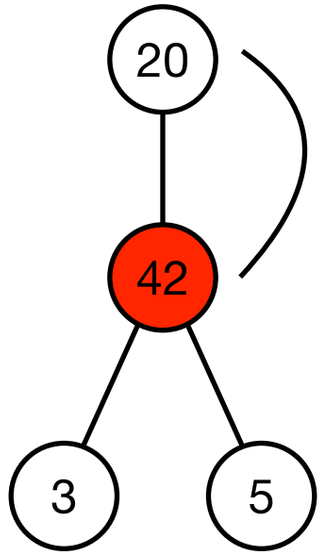
- Zeit. O(1)
- Platz. O(n)

Prioritätswarteschlangen

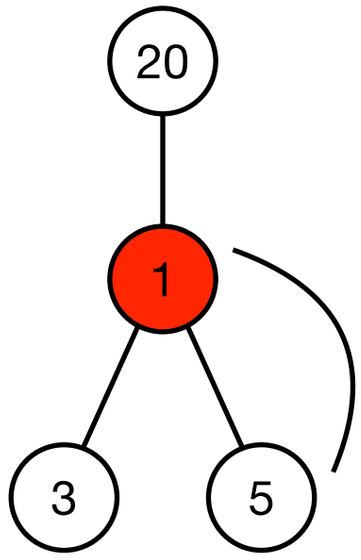
- Prioritätswarteschlangen
- Bäume und Heaps
- Darstellung von Heaps
- Algorithmen für Heaps
- Einen Heap bauen
- Heapsort

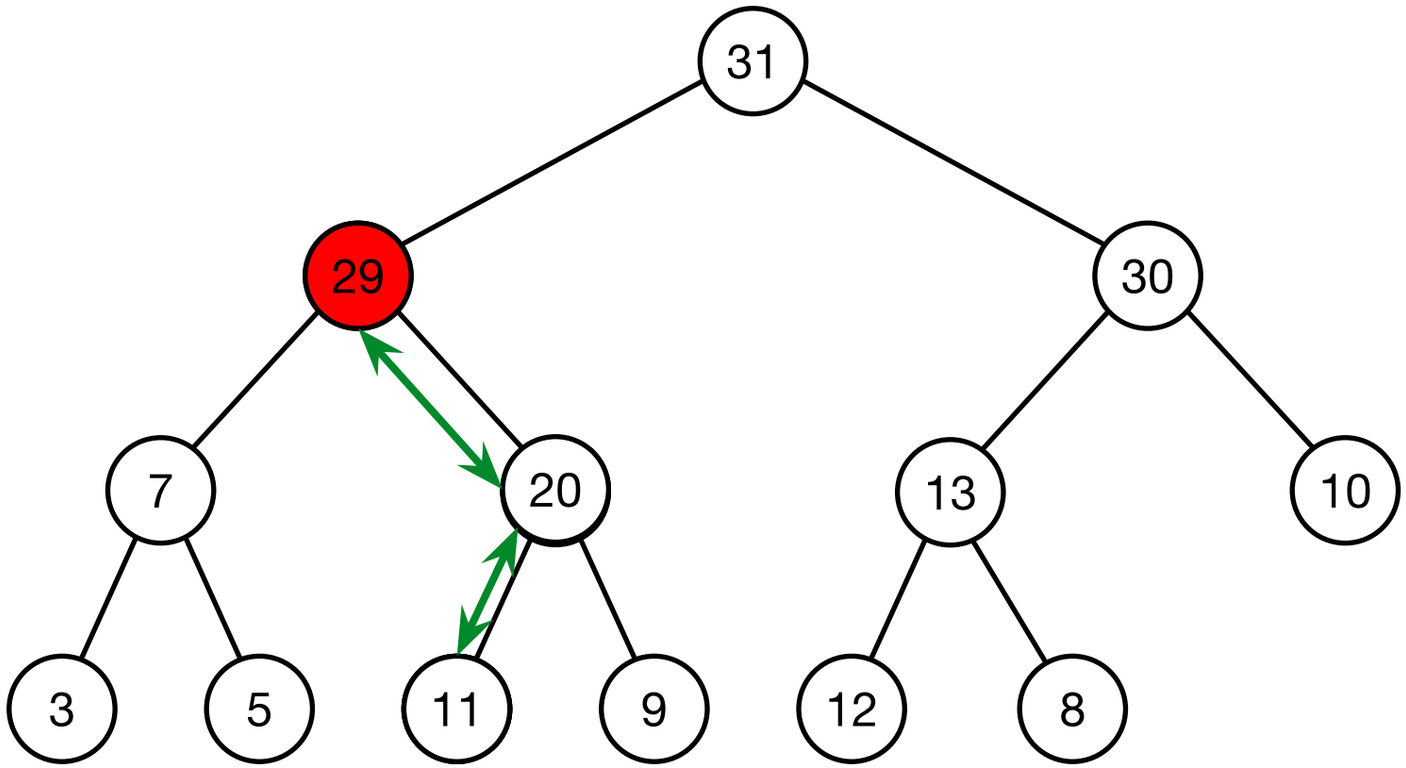
Algorithmen für Heaps

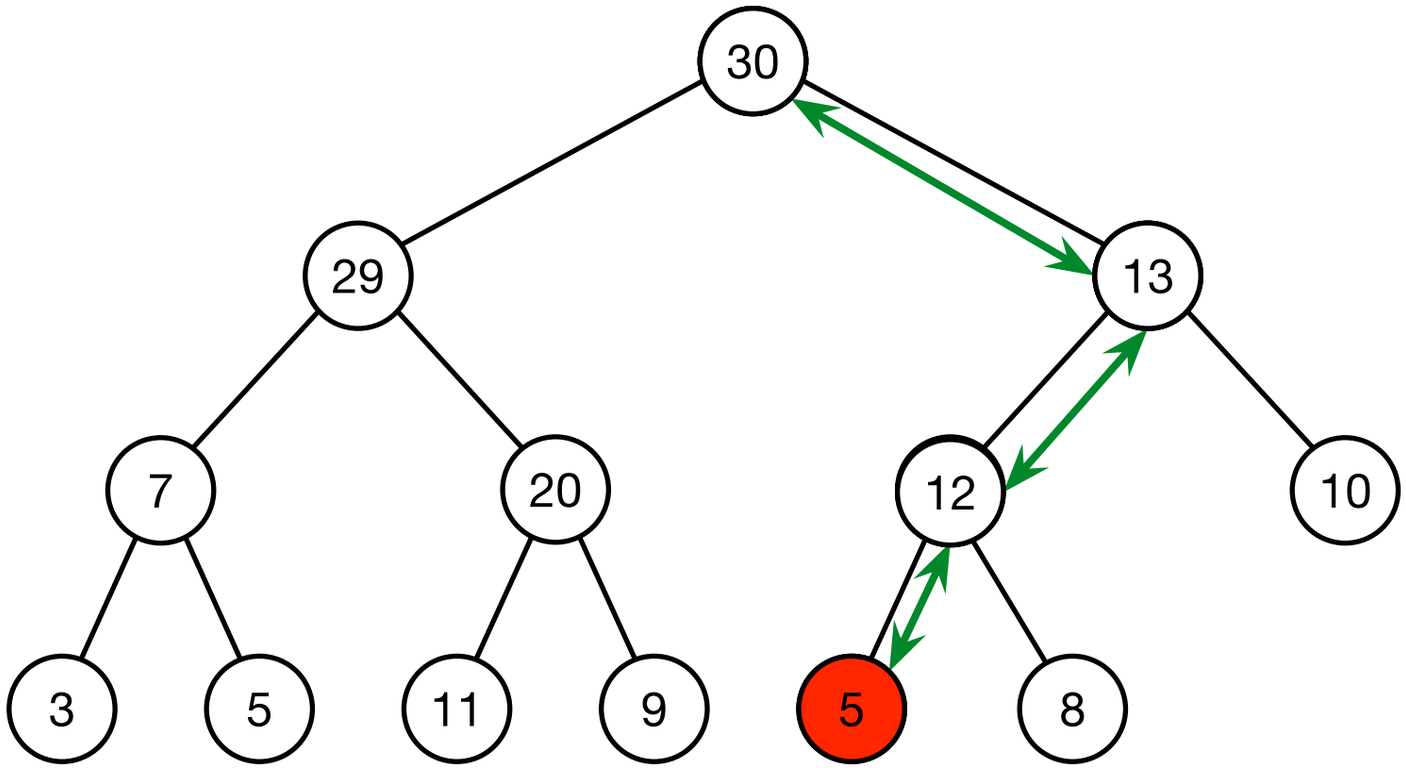
- BubbleUp(x):
  - Wenn Heap-Eigenschaft an Knoten x verletzt ist, weil x.key größer als der Schlüssel des Elternknoten ist:
  - Tausche x und Elternknoten
  - Wiederhole BubbleUp mit Elternknoten bis Heap-Eigenschaft erfüllt ist.



- BubbleDown(x):
  - Wenn Heap-Eigenschaft an Knoten x verletzt ist, weil x.key kleiner als der Schlüssel des linken oder rechten Kindes:
  - Tausche x und Kind c mit größtem Schlüssel.
  - Wiederhole BubbleDown mit dem Kind, bis Heap-Eigenschaft erfüllt ist.

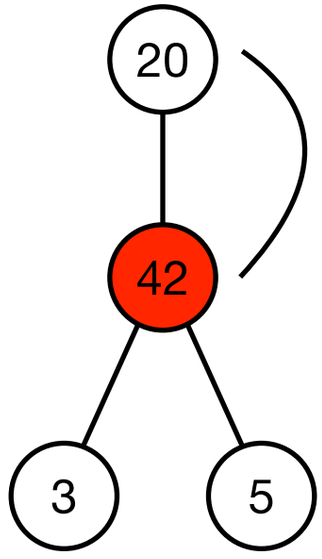


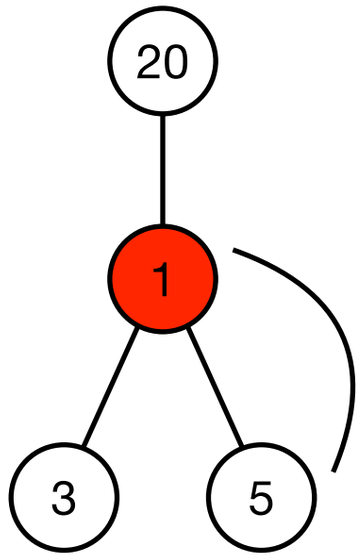

31
29
30
7
20
13
10
3
5
11
9
12
8


30
29
13
7
20
12
10
3
5
11
9
5
8

Algorithmen für Heaps

- BubbleUp(x):
  - Wenn Heap-Eigenschaft an Knoten x verletzt ist, weil x.key größer als der Schlüssel des Elternknoten ist:
  - Tausche x und Elternknoten
  - Wiederhole BubbleUp mit Elternknoten bis Heap-Eigenschaft erfüllt ist.
- BubbleDown(x):
  - Wenn Heap-Eigenschaft an Knoten x verletzt ist, weil x.key kleiner als der Schlüssel des linken oder rechten Kindes:
  - Tausche x und Kind c mit größtem Schlüssel.
  - Wiederhole BubbleDown mit dem Kind, bis Heap-Eigenschaft erfüllt ist.
- Zeit. BubbleUp und BubbleDown in O(log n) Zeit.
- Wie kann man dies benutzen, um eine Prioritätswarteschlange zu implementieren?





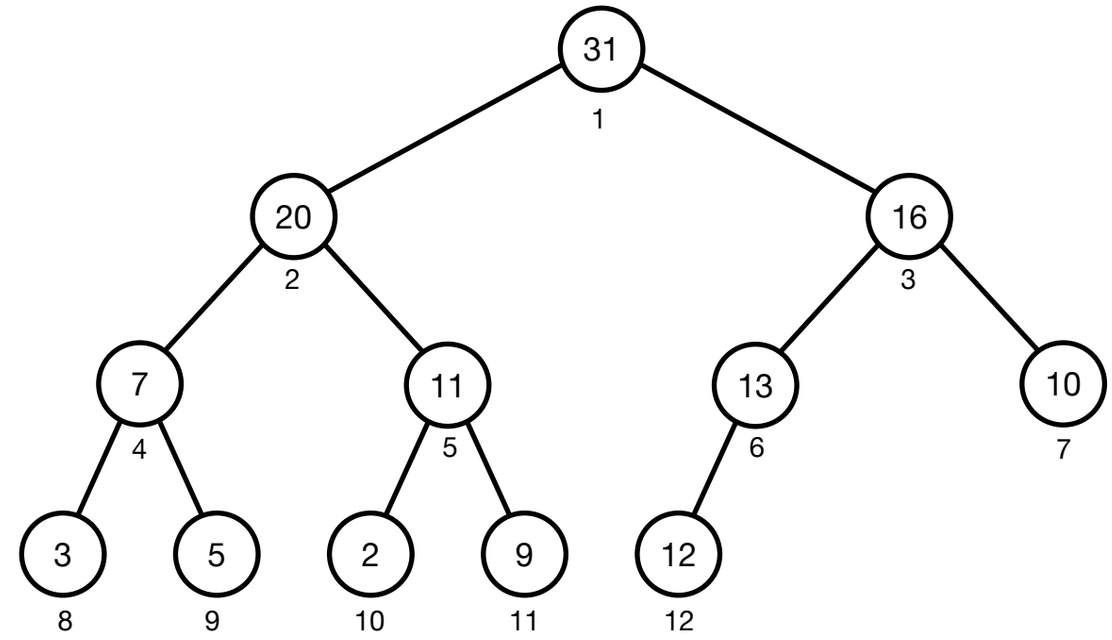
Prioritätswarteschlangen

```
Max()
    return H[1]
```

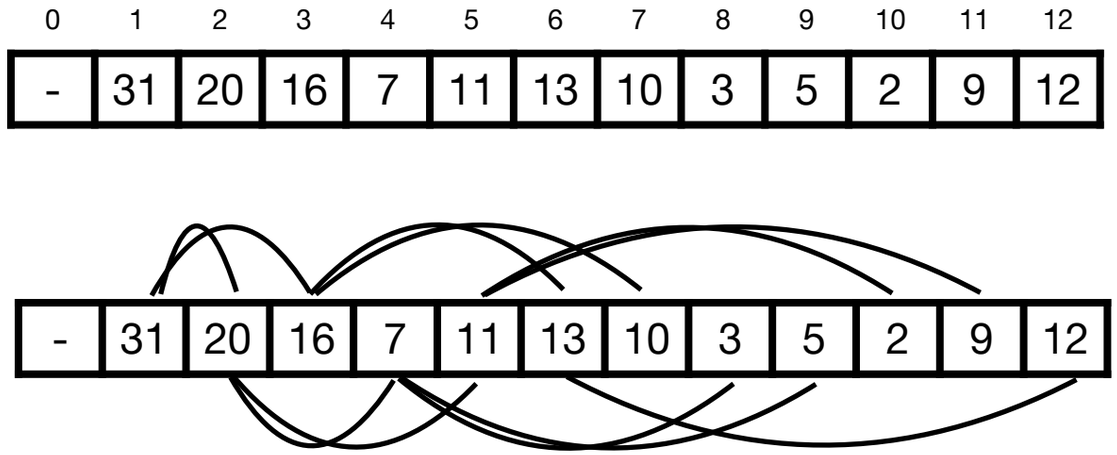
```
ExtractMax()
    r = H[1]
    H[1] = H[n]
    n = n - 1
    BubbleDown(1)
    return r
```

```
Insert(x)
    n = n + 1
    H[n] = x
    BubbleUp(n)
```

```
IncreaseKey(x,k)
    H[x] = k
    BubbleUp(x)
```



- Beispiel. Führe von Hand die folgenden Operationen auf einem anfangs leeren Heap aus:
  - `Insert` je 2, 5, 7, 6, 4.
  - Dann zweimal `ExtractMax`.
  - Zeichne den Heap nach jeder einzelnen Operation.



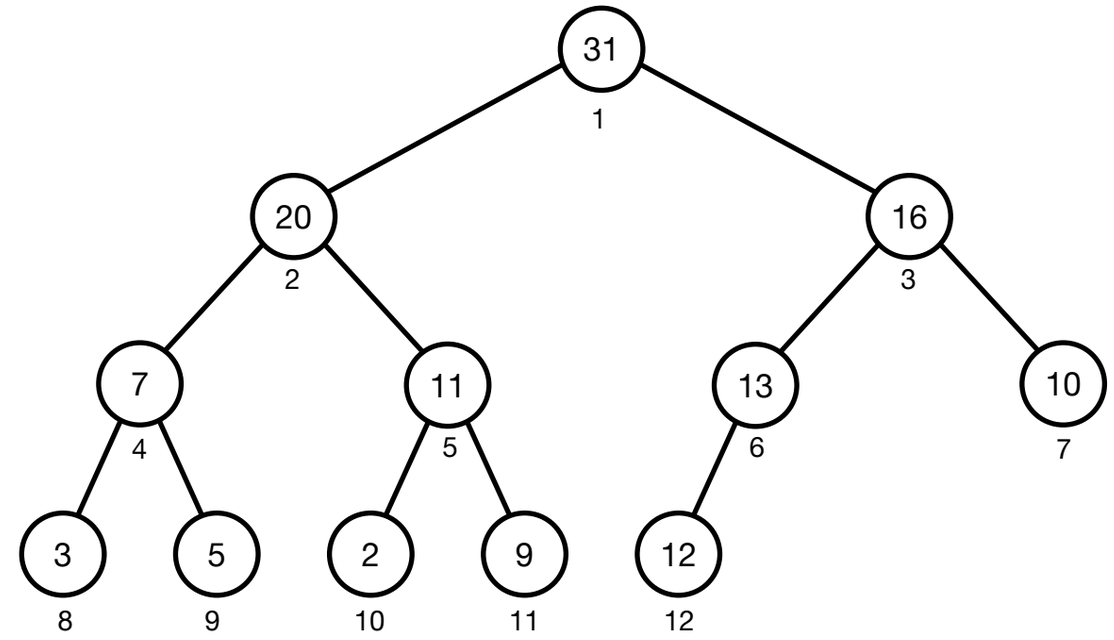
Prioritätswarteschlangen

```
Max()
    return H[1]
```

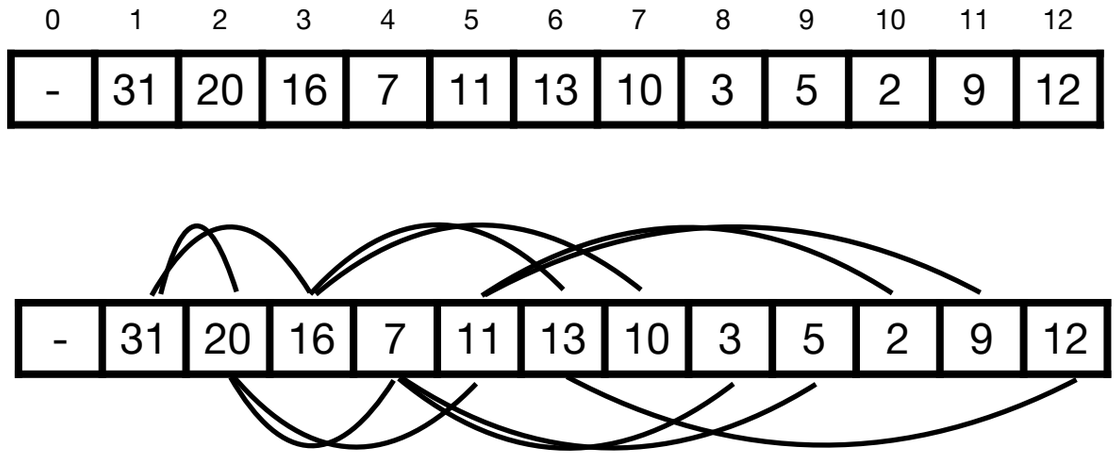
```
ExtractMax()
    r = H[1]
    H[1] = H[n]
    n = n - 1
    BubbleDown(1)
    return r
```

```
Insert(x)
    n = n + 1
    H[n] = x
    BubbleUp(n)
```

```
IncreaseKey(x,k)
    H[x] = k
    BubbleUp(x)
```



- Zeit.
  - Max in Zeit O(1).
  - ExtractMax, IncreaseKey, und Insert in Zeit O(log n).



Prioritätswarteschlangen

Datenstruktur	Max	ExtractMax	IncreaseKey	Insert	Space
verkettete Liste	O(n)	O(n)	O(1)	O(1)	O(n)
sortierte verkettete Liste	O(1)	O(1)	O(n)	O(n)	O(n)
Heap	O(1)	O(log n)	O(log n)	O(log n)	O(n)

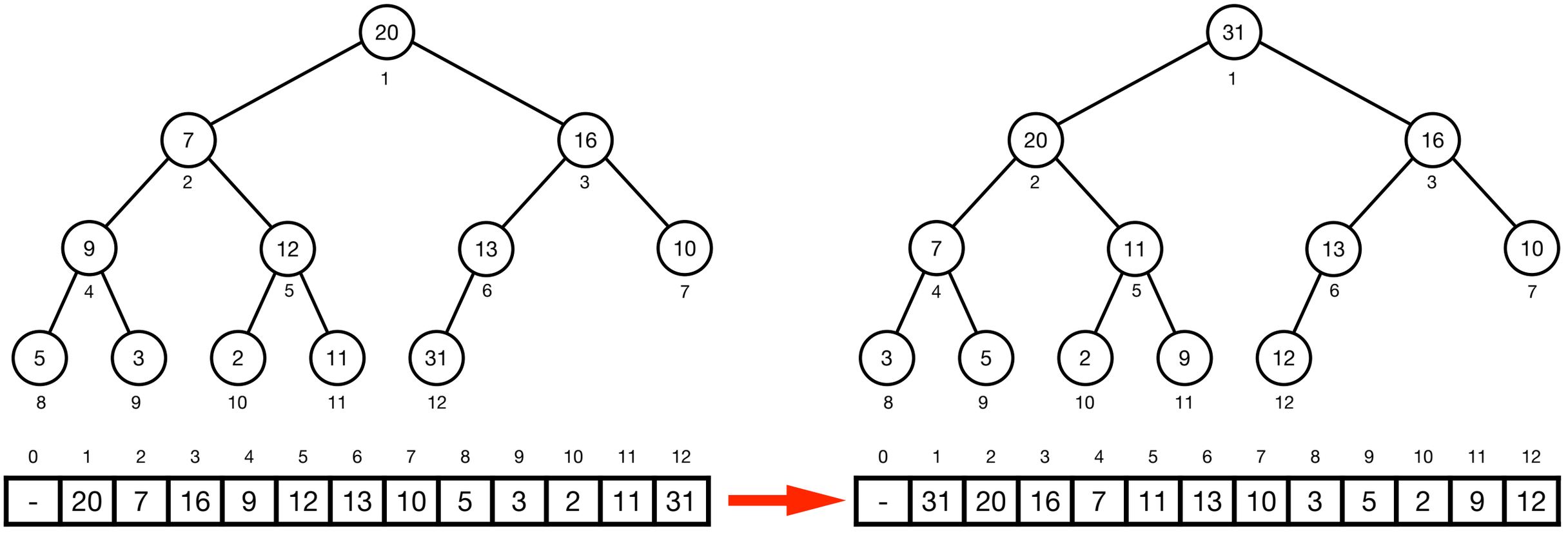
- Heaps mit Feld als Datenstruktur ist ein Beispiel einer impliziten Datenstruktur.

Prioritätswarteschlangen

- Prioritätswarteschlangen
- Bäume und Heaps
- Darstellung von Heaps
- Algorithmen für Heaps
- Einen Heap bauen
- Heapsort

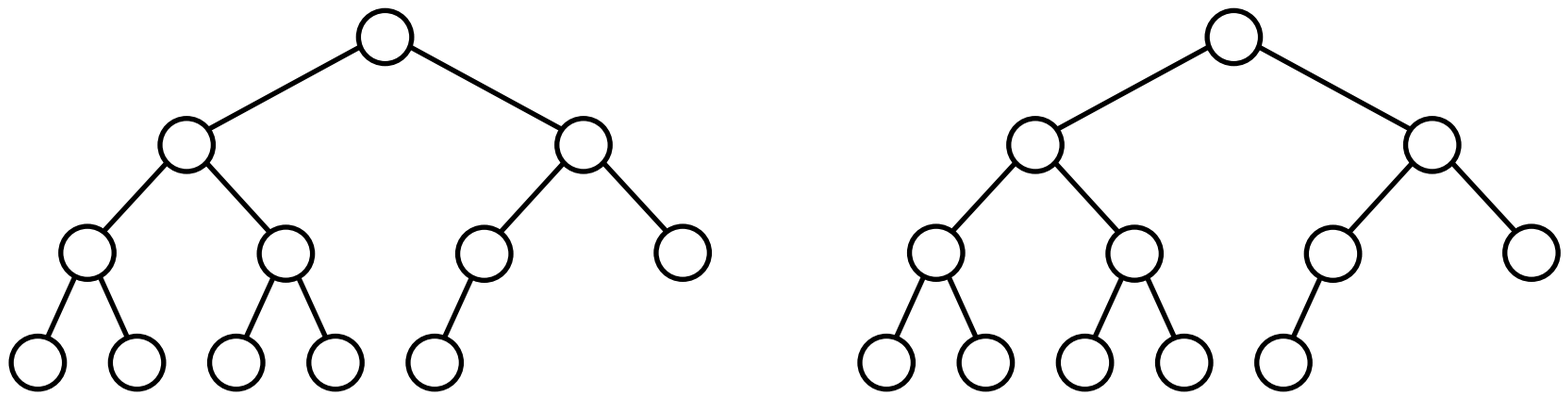
Einen Heap bauen: `heapify`

- Die Funktion **`heapify`**(H):
  Gegeben n Zahlen in unsortiertem Feld H[1..n], konvertiere Feld zu Heap.



Einen Heap bauen: `heapify`

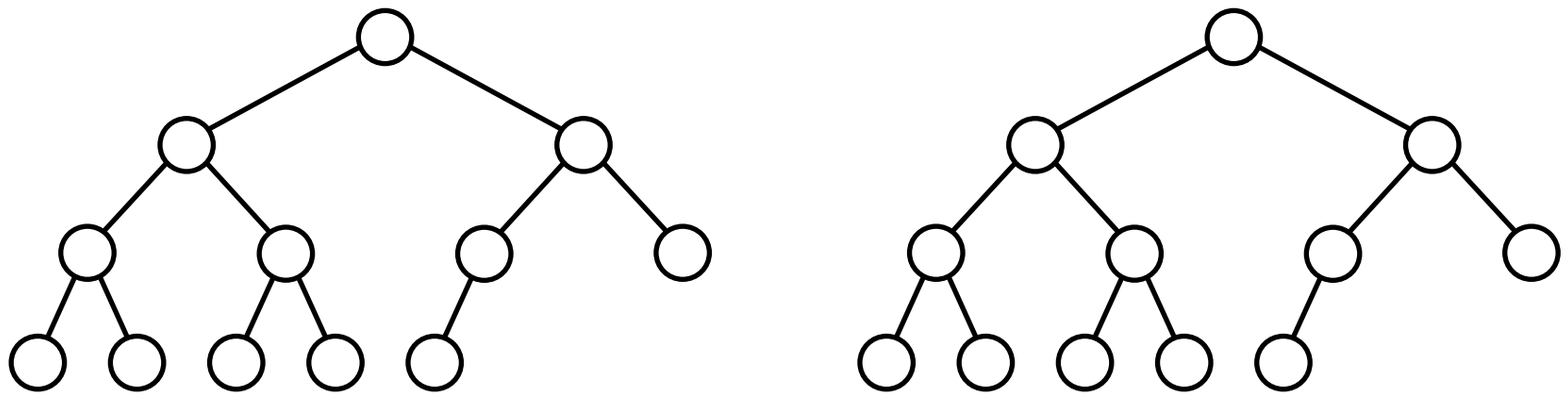
- Lösung 1: top-down Konstruktion
  - Für alle Knoten in *level order* (Ebene für Ebene von oben nach unten), wende BUBBLEUP an.



- Zeit.
  - Für jeden Knoten von Tiefe d brauchen wir O(d) Zeit.
  - 1 Knoten hat Tiefe 0,
    2 Knoten haben Tiefe 1,
    4 Knoten haben Tiefe 2, ...,
    ~n/2 Knoten haben Tiefe log n.
  - ⇒ Gesamtzeit ist O(n log n)
- Frage. Geht das besser?

Einen Heap bauen: `heapify`

- Lösung 2: bottom-up Konstruktion
  - Für alle Knoten in umgekehrter *level order*, wende BubbleDown an.



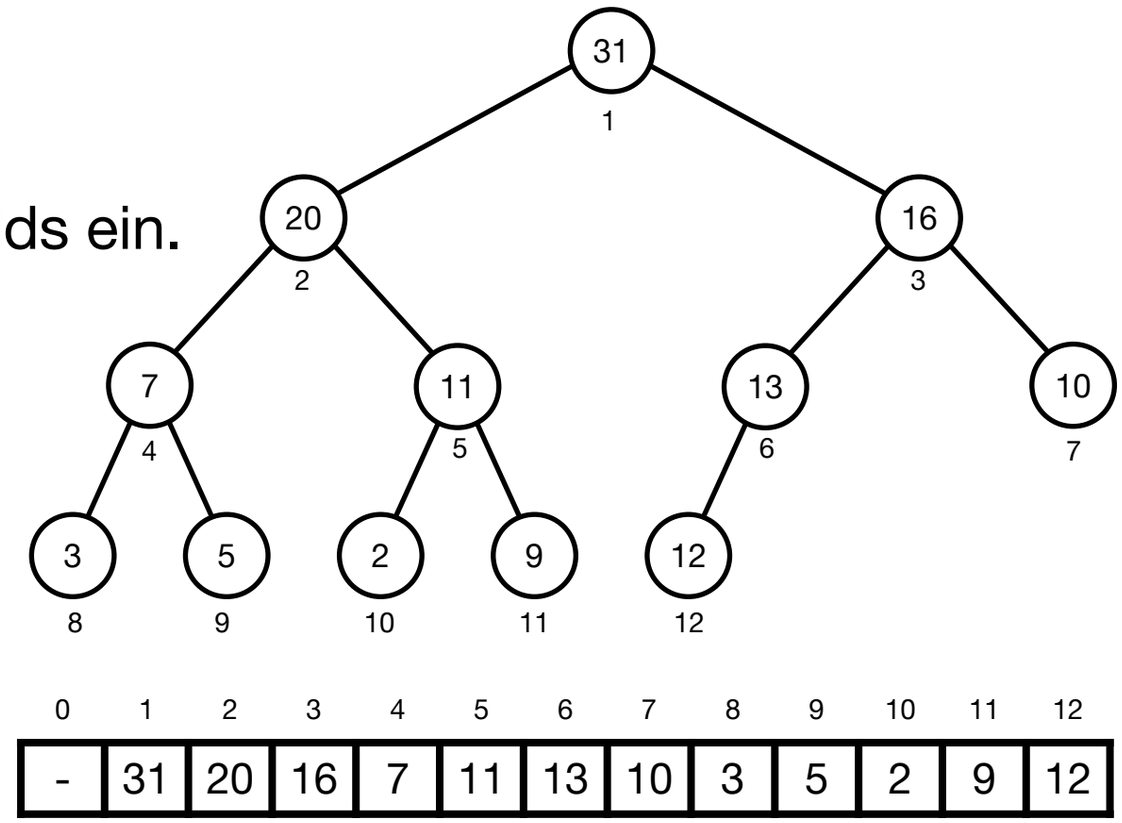
- Zeit.
  - Für jeden Knoten von Höhe h brauchen wir Zeit O(h).
  - 1 Knoten hat Höhe log n,
    2 Knoten haben Höhe log n - 1, ...,
    n/4 Knoten haben Höhe 1,
    n/2 Knoten haben Höhe 0.
  - ⇒ Gesamtzeit ist O(n) (siehe Übungen)

Prioritätswarteschlangen

- Prioritätswarteschlangen
- Bäume und Heaps
- Darstellung von Heaps
- Algorithmen für Heaps
- Einen Heap bauen
- Heapsort

Heapsort

- Sortieren. Wie können wir ein Feld H[1..n] mit einem Heap sortieren?
- Lösung.
  - Baue einen Heap für H.
  - Wende n mal ExtractMax an.
    - Füg die Ergebnisse am Ende des Felds ein.
  - Liefere H zurück.

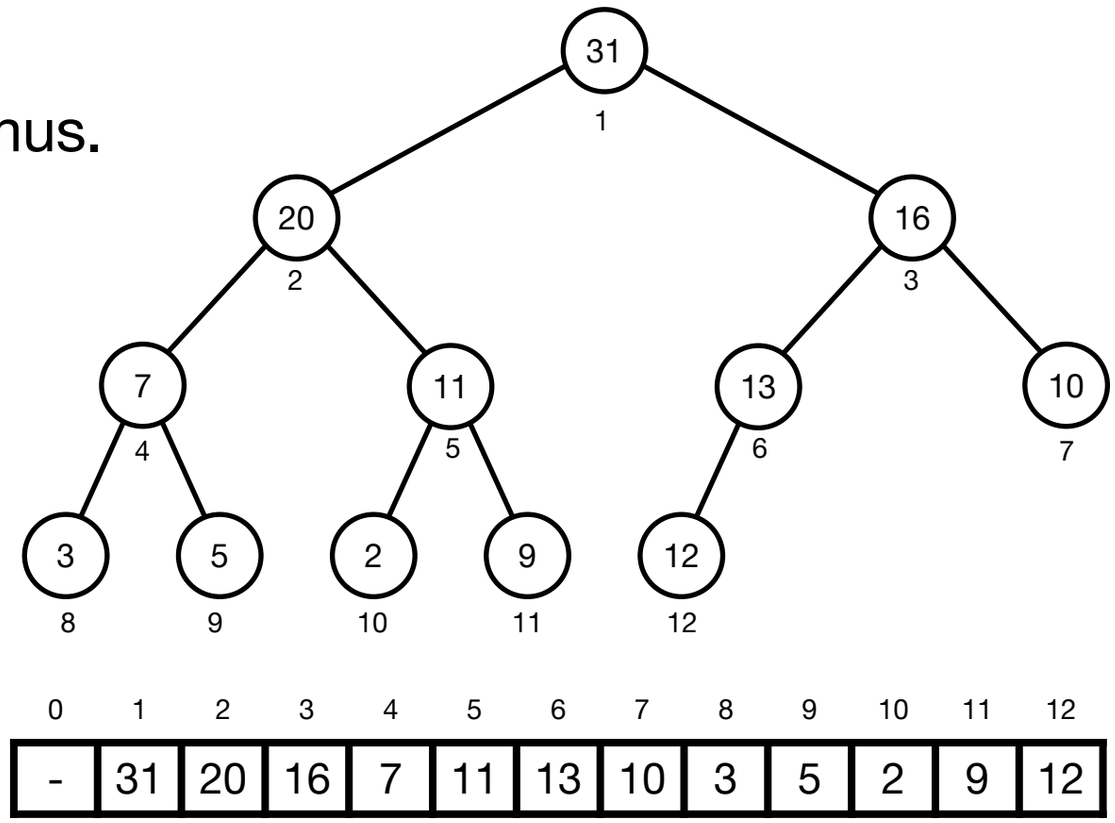


- Zeit.
  - Heap-Konstruktion in Zeit O(n).
  - n mal ExtractMax braucht insgesamt Zeit O(n log n).
  - ⇒ Gesamtzeit ist O(n log n).

Heapsort

- Satz. Wir können ein Feld in Zeit O(n log n) sortieren.

- Braucht nur O(1) zusätzlichen Platz.
- ⇒ Heapsort ist ein *in-place* Sortieralgorithmus.



Prioritätswarteschlangen

- Prioritätswarteschlangen
- Bäume und Heaps
- Darstellung von Heaps
- Algorithmen für Heaps
- Einen Heap bauen
- Heapsort