# Algorithmen und Datenstrukturen
# Kapitel 4
# Neue Datenstrukturen, besseres (?) Sortieren (Teil 3)

Frank Heitmann
heitmann@informatik.uni-hamburg.de

4. November 2015

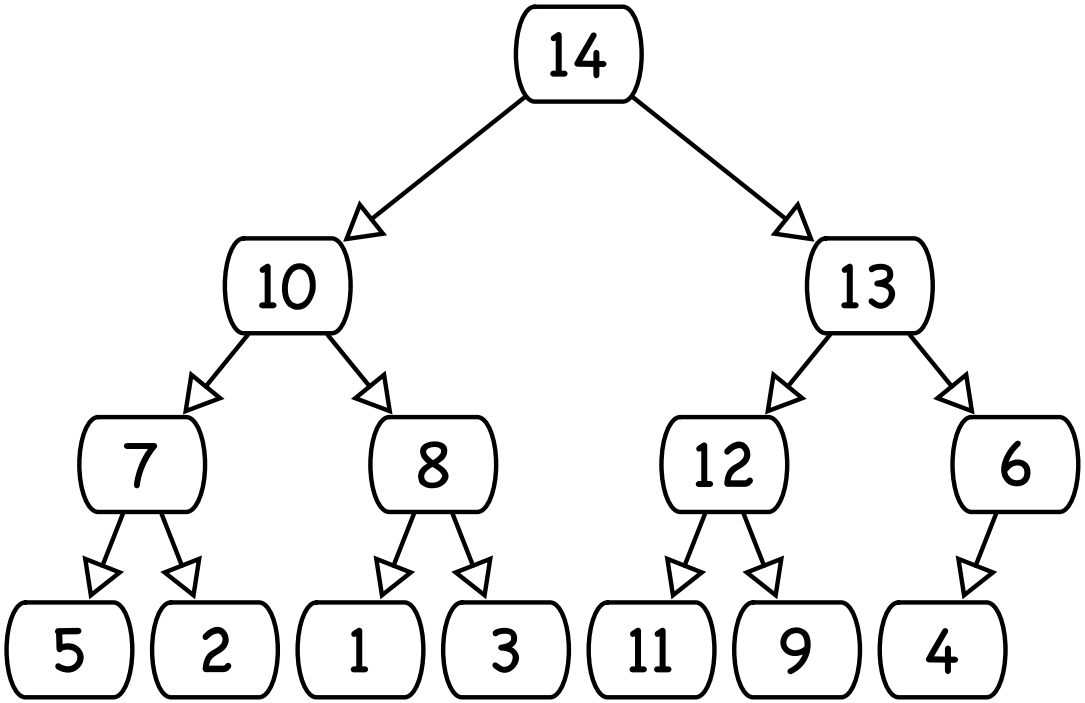
# Heaps als Warteschlangen

Heaps können genutzt werden, um effiziente Prioritätswarteschlangen zu implementieren. Das Ziel ist es die wichtigen Operationen

- Maximum($S$)
- ExtractMax($S$)
- Insert($S, x$)
- IncreaseKey($S, x, k$)

in $O(\log n)$ Zeit zu realisieren.

# Maximum und ExtractMax - Die Idee



# Maximum und ExtractMax - Pseudocode

**Algorithmus 1** HeapMaximum($A$)

1: **return** $A[1]$

**Algorithmus 2** HeapExtractMax($A$)

1: **if** heap-größe$[A] < 1$ **then** error
2: $max = A[1]$
3: $A[1] = A[\text{heap-größe}[A]]$
4: heap-größe$[A] =$ heap-größe$[A] - 1$
5: `MaxHeapify`$(A, 1)$
6: **return** $max$

Beispiel folgt später...

# Insert/Increase - Idee und Pseudocode

Die Idee von HeapInsert ist das neue Element einfach ganz rechts im Array (im Baum ganz rechts in der untersten Ebene) anzufügen und es dann im Baum nach 'oben' wandern zu lassen, bis es an der richtigen Stelle ist. (Dies ist sehr ähnlich zu der MaxHeapify Operation.)

**Algorithmus 3** HeapIncreaseKey($A, i, key$)

```
1: if key < A[i] then
2:     Error
3: end if
4: A[i] = key
5: while i > 1 && A[Vater(i)] < A[i] do
6:     swap(A[i], A[Vater(i)])
7:     i = Vater(i)
8: end while
```

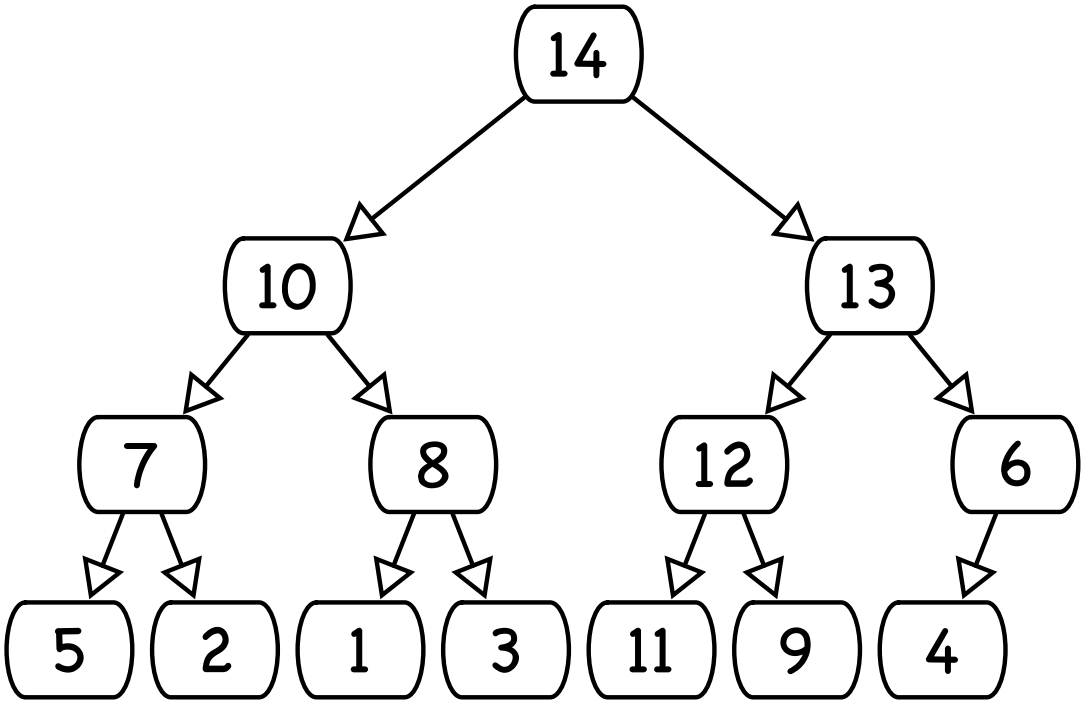
# Insert/Increase - Idee und Pseudocode

Die Idee von HeapInsert ist das neue Element einfach ganz rechts im Array (im Baum ganz rechts in der untersten Ebene) anzufügen und es dann im Baum nach 'oben' wandern zu lassen, bis es an der richtigen Stelle ist. (Dies ist sehr ähnlich zu der MaxHeapify Operation.)

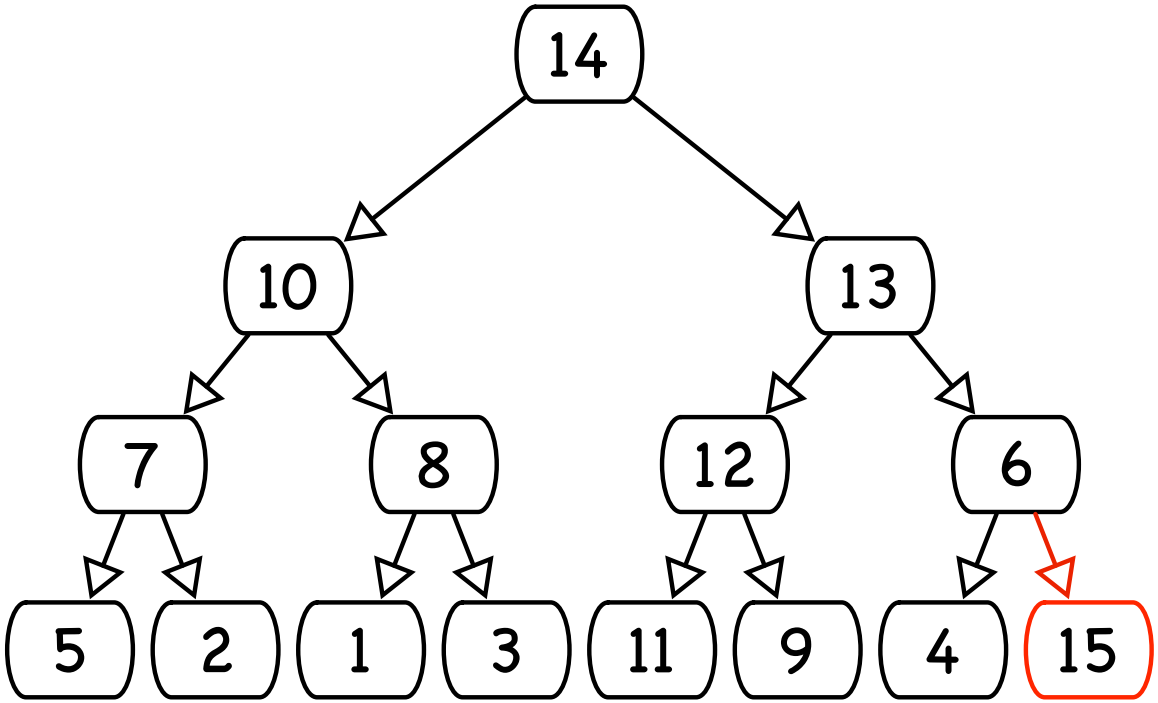
**Algorithmus 4** HeapInsert($A$, key)

1: heap-größe$[A]$ = heap-größe$[A] + 1$
2: $A$[heap-größe$[A]] = -\infty$
3: `HeapInscreaseKey`($A$, heap-größe$[A]$, key)

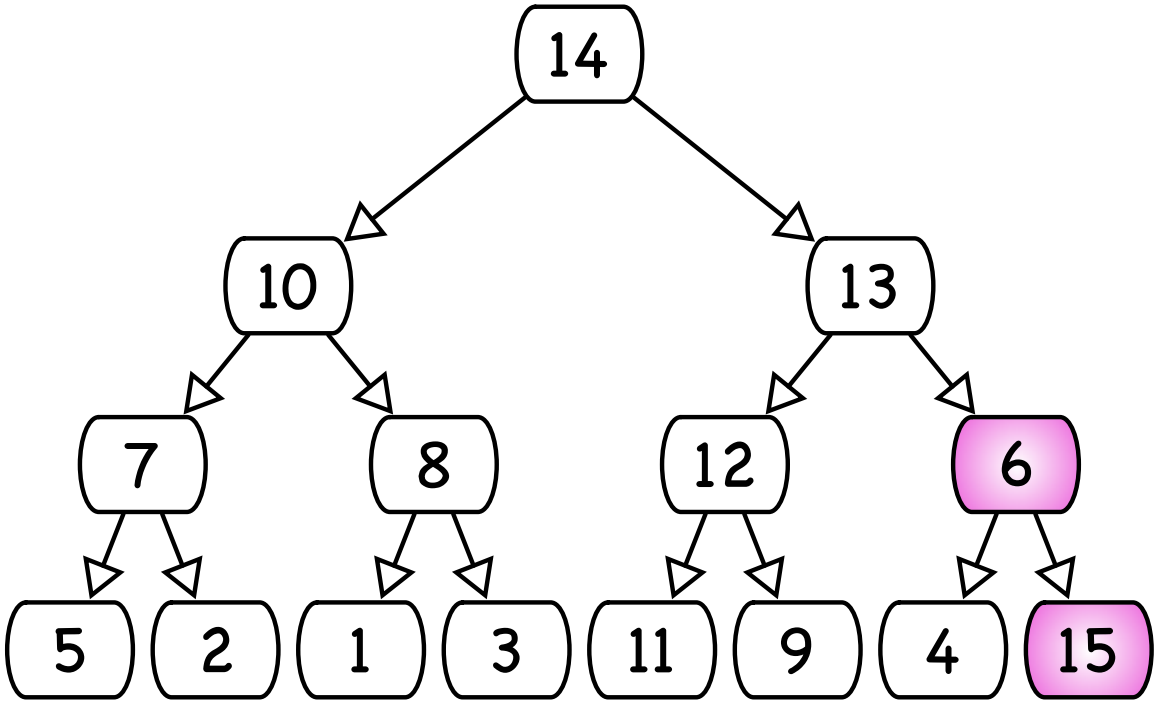
# MaxHeapInsert - Beispiel



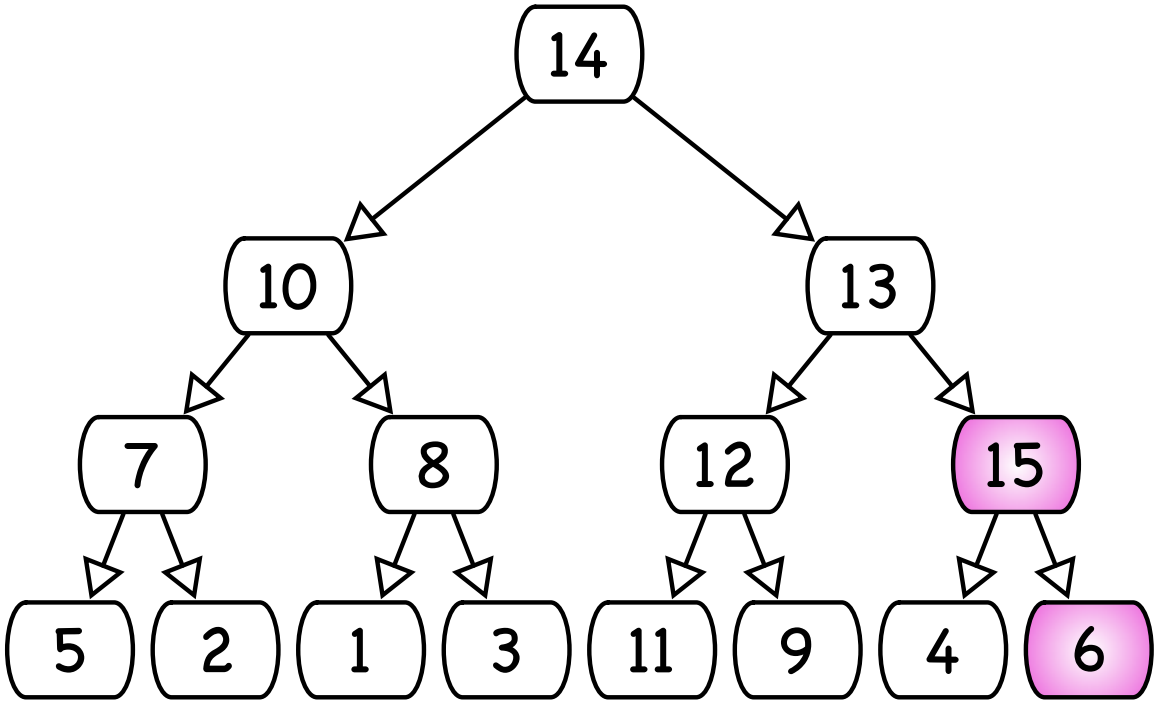
# MaxHeapInsert - Beispiel



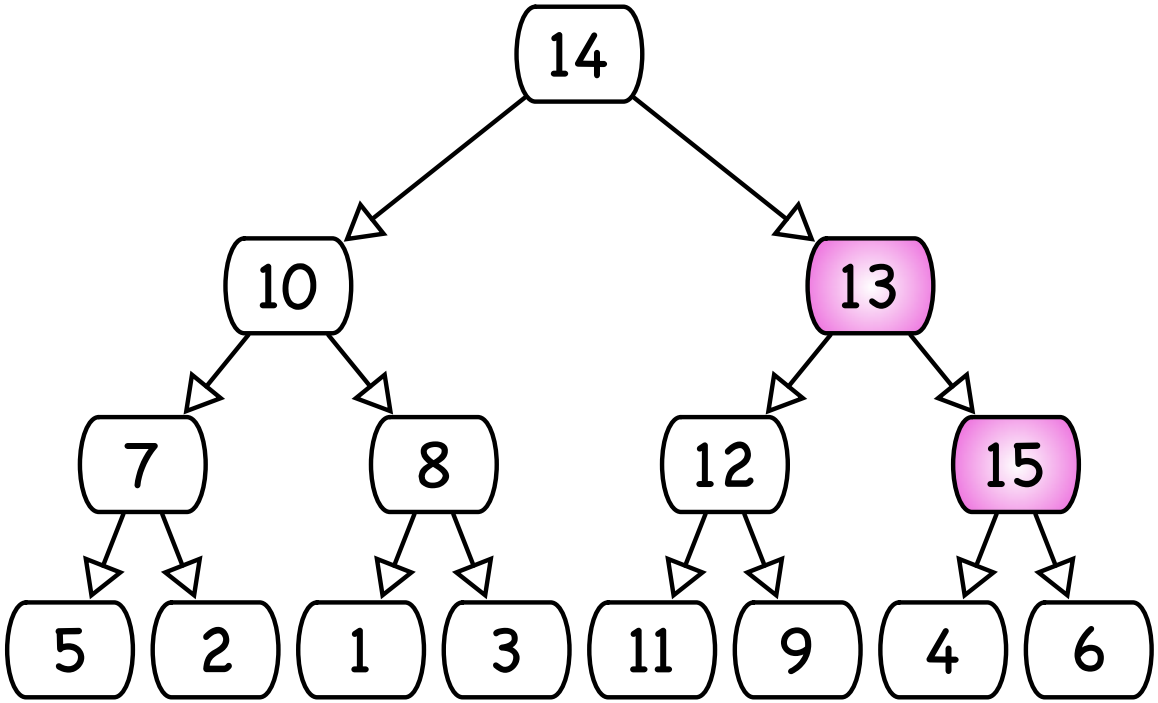
# MaxHeapInsert - Beispiel



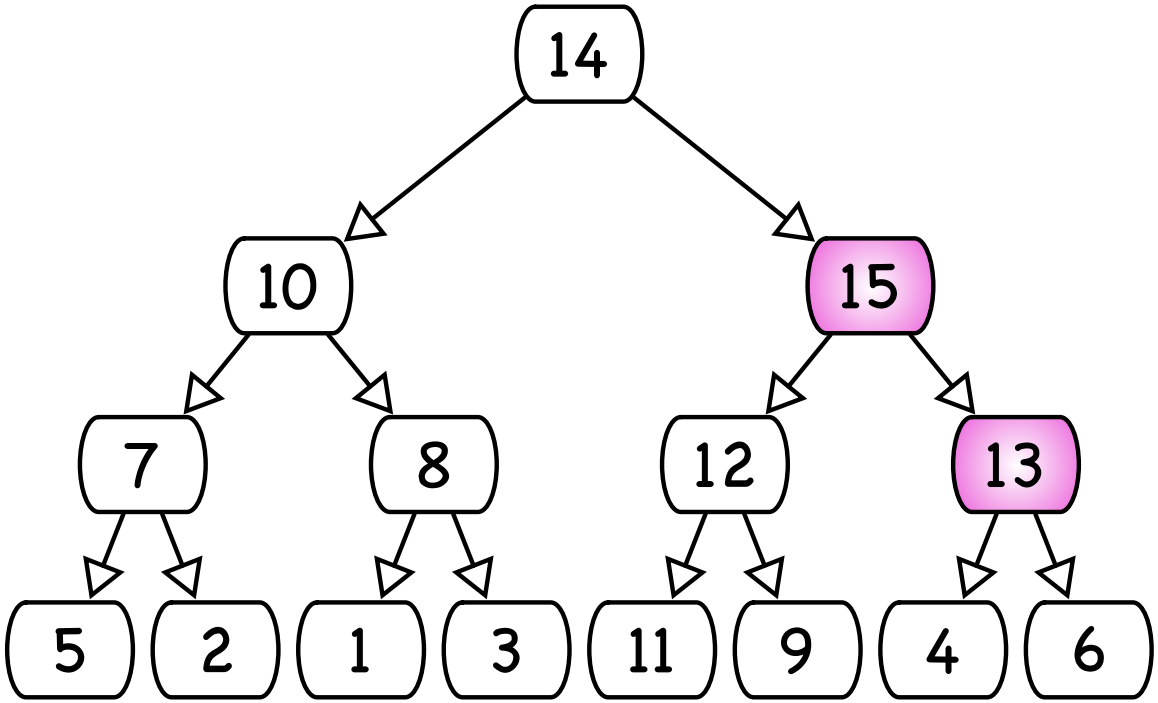
# MaxHeapInsert - Beispiel



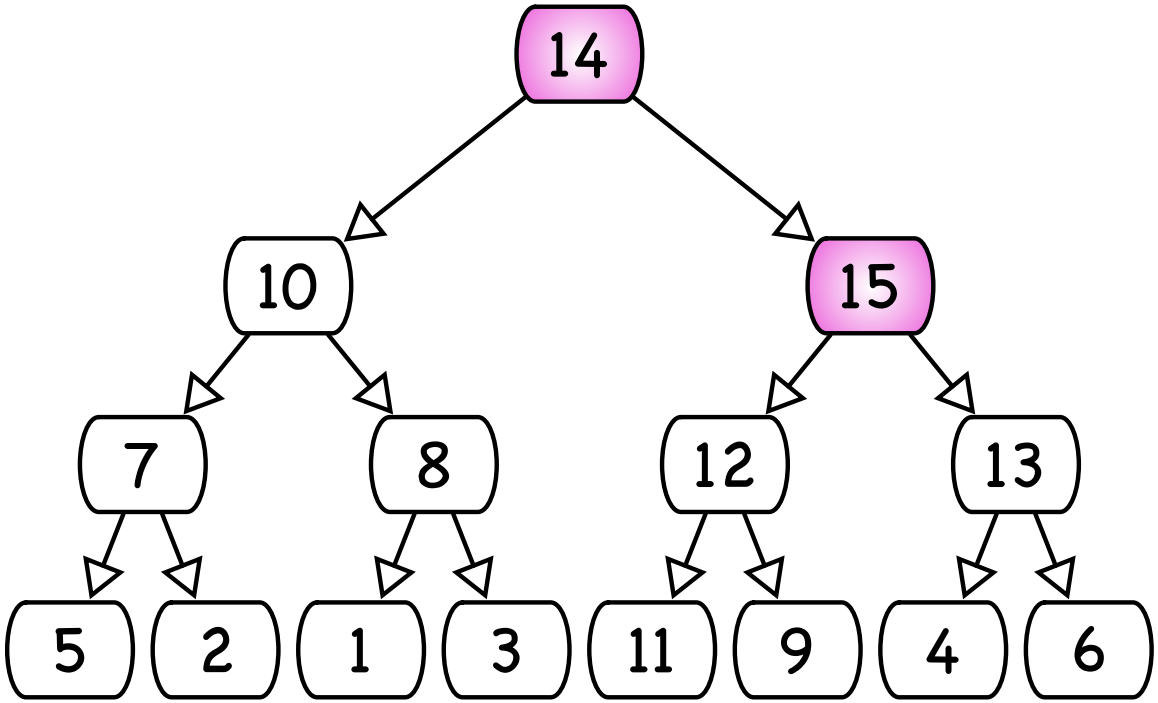
# MaxHeapInsert - Beispiel



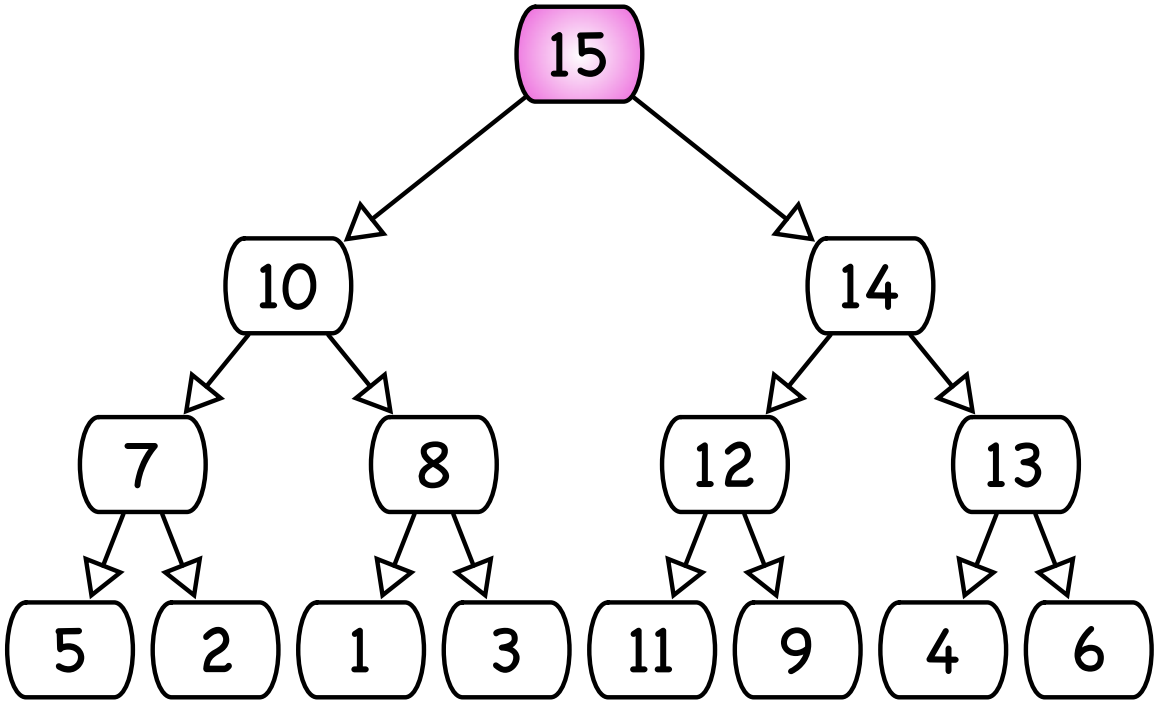
# MaxHeapInsert - Beispiel



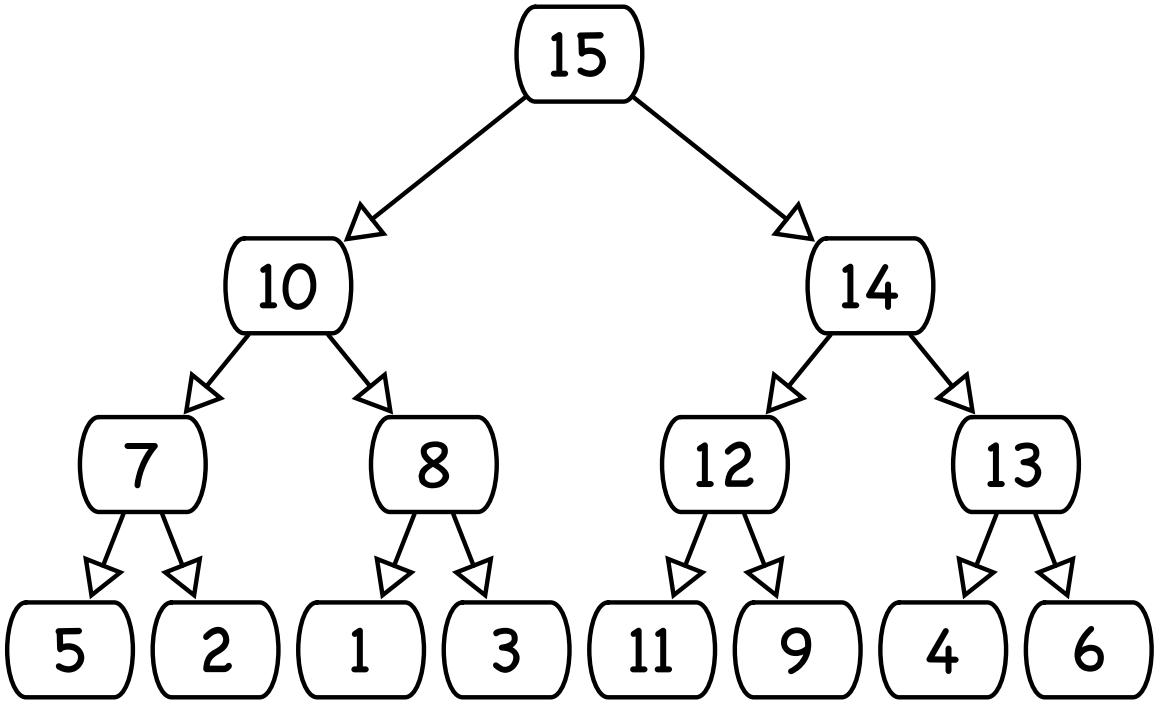
# MaxHeapInsert - Beispiel



# MaxHeapInsert - Beispiel



# MaxHeapInsert - Beispiel



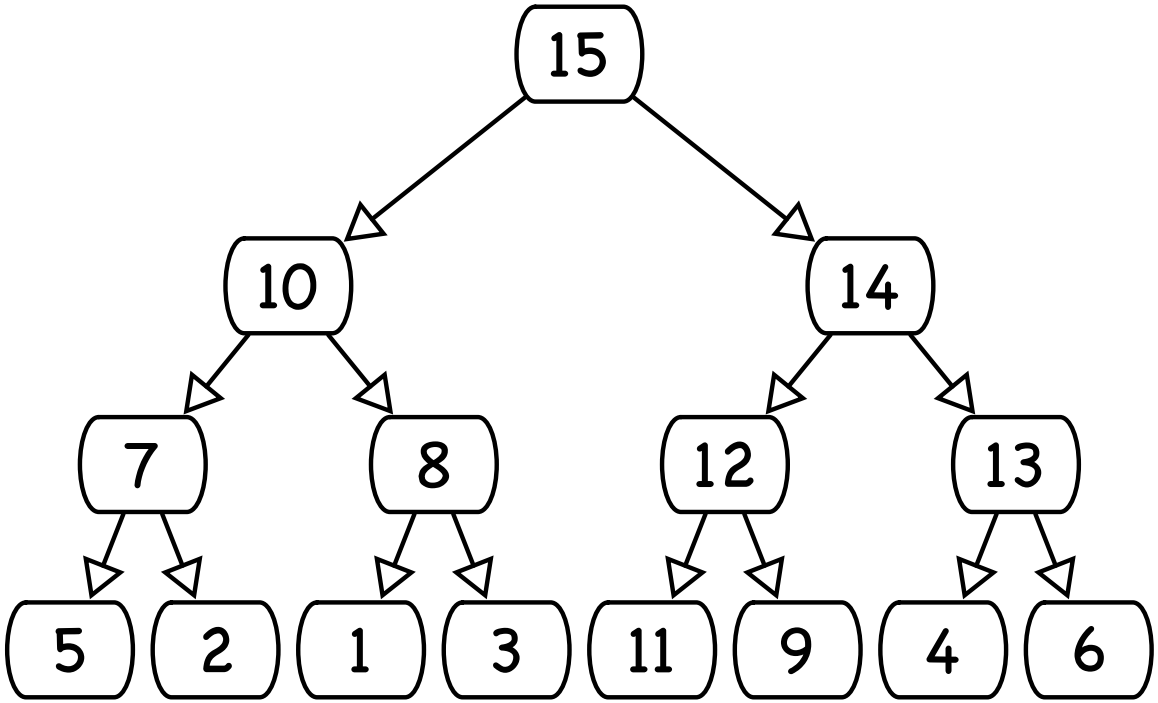
# ExtractMax

Und jetzt noch ein Beispiel für HeapExtractMax ...

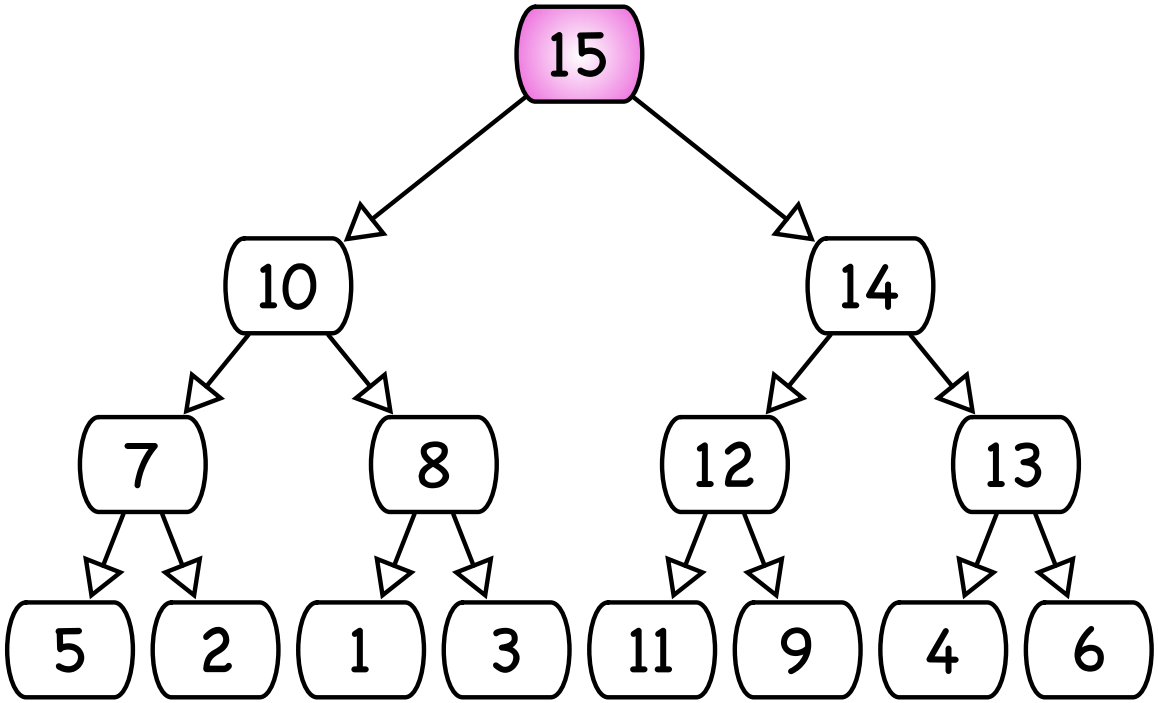
**Algorithmus 5** HeapExtractMax($A$)

1: **if** heap-größe$[A] < 1$ **then** error
2: $max = A[1]$
3: $A[1] = A[\text{heap-größe}[A]]$
4: heap-größe$[A] =$ heap-größe$[A] - 1$
5: `MaxHeapify`$(A, 1)$
6: **return** $max$

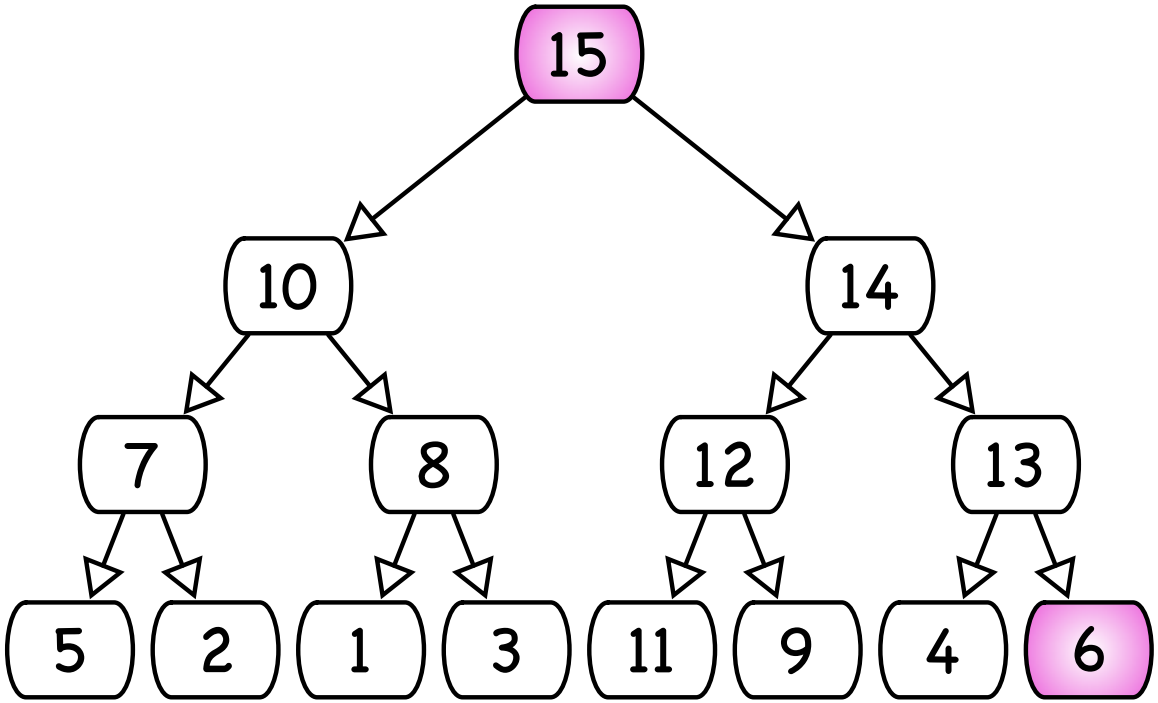
# HeapExtractMax - Beispiel



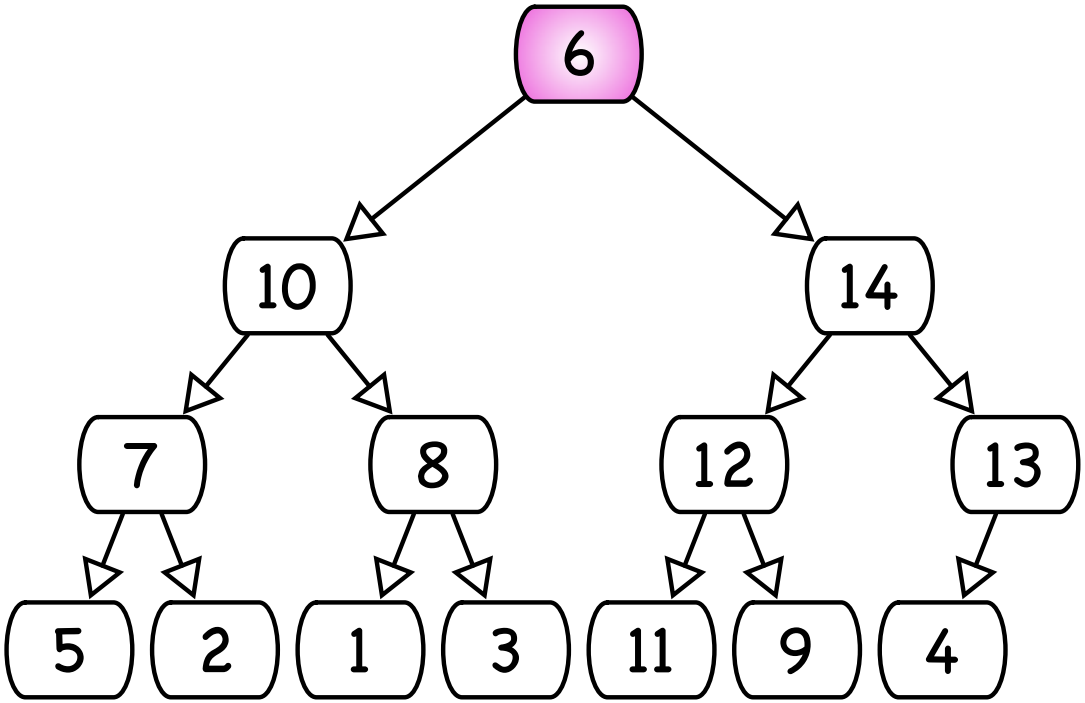
# HeapExtractMax - Beispiel



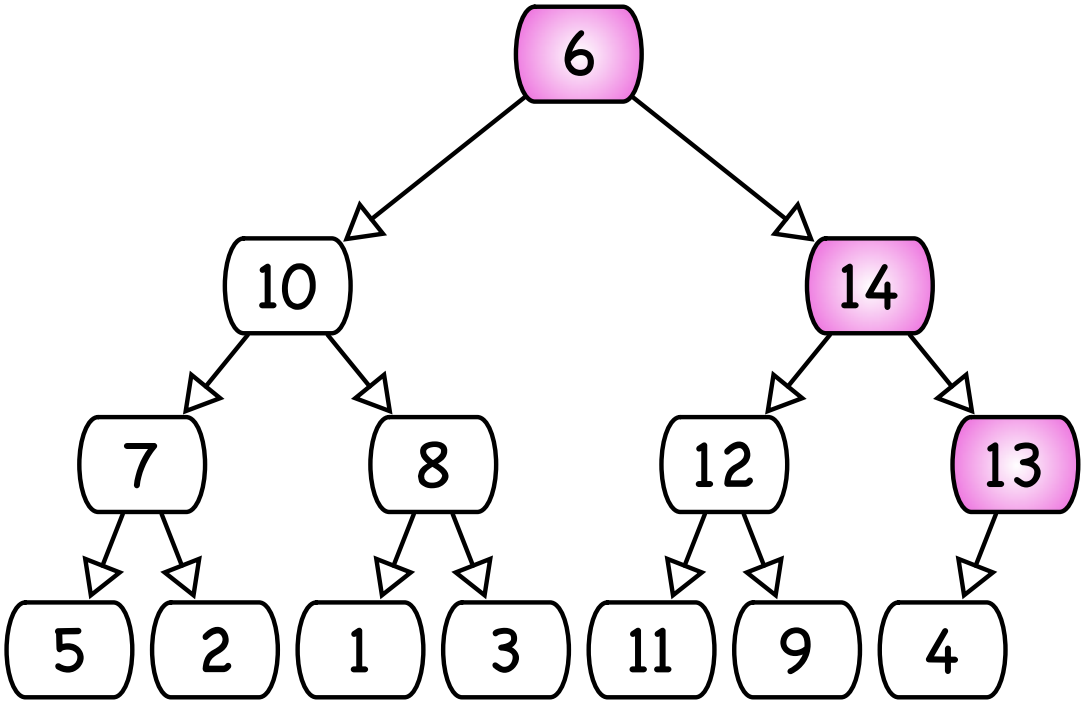
# HeapExtractMax - Beispiel



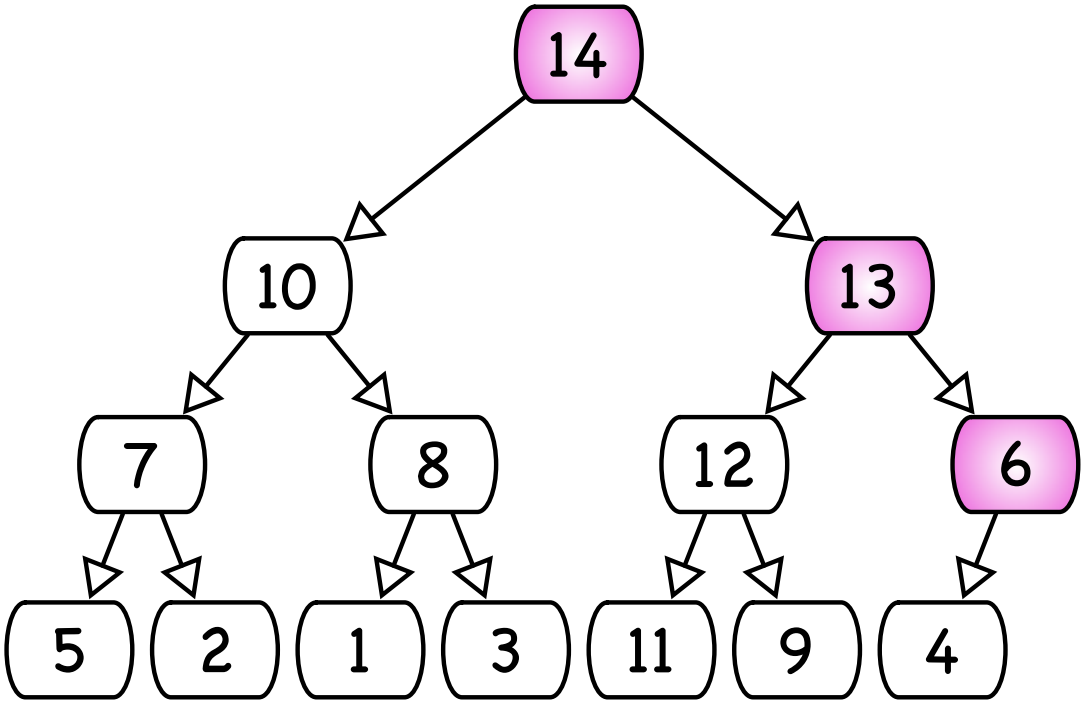
# HeapExtractMax - Beispiel



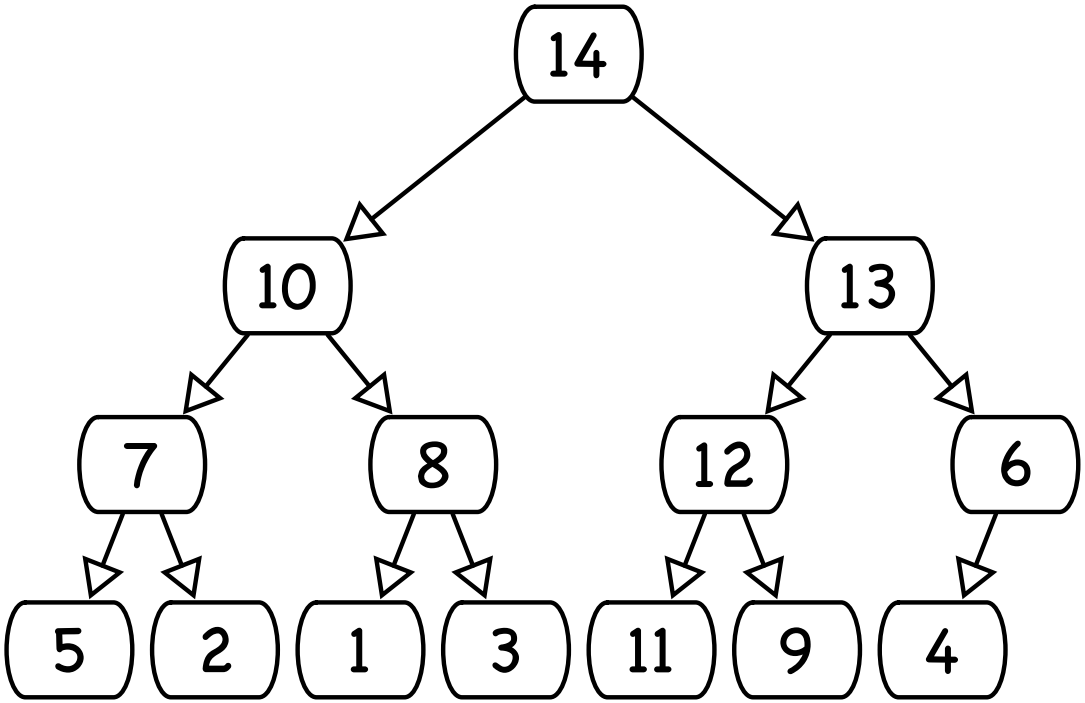
# HeapExtractMax - Beispiel



# HeapExtractMax - Beispiel



# HeapExtractMax - Beispiel



# Wiederholung: Prioritätswarteschlangen

- Heaps für Warteschlangen
  - HeapMaximum und HeapExtractMax (letztere nutzt wieder MaxHeapify)
  - HeapIncreaseKey (im Prinzip die 'Umkehrung' von MaxHeapify)
  - HeapInsert (nutzt HeapIncreaseKey)

# Heaps als Warteschlange: Die Idee

- Die Idee von HeapInsert ist das neue Element einfach ganz rechts im Array (im Baum ganz rechts in der untersten Ebene) anzufügen und es dann im Baum nach 'oben' wandern zu lassen, bis es an der richtigen Stelle ist. (Dies ist sehr ähnlich zu der MaxHeapify Operation.)
- HeapIncreaseKey arbeitet genauso, fängt aber nicht immer ganz unten/rechts an.
- Maximum und ExtractMax arbeiten im Prinzip wie der Schleifenrumpf von HeapSort. (ExtractMax muss die Heap-Eigenschaft wieder herstellen und nutzt MaxHeapify.)

# Warteschlangen: Maximum und ExtractMax

**Algorithmus 6** HeapMaximum($A$)

1: **return** $A[1]$

**Algorithmus 7** HeapExtractMax($A$)

1: **if** heap-größe$[A] < 1$ **then** error
2: $max = A[1]$
3: $A[1] = A[\text{heap-größe}[A]]$
4: heap-größe$[A] =$ heap-größe$[A] - 1$
5: `MaxHeapify`$(A, 1)$
6: **return** $max$

# Warteschlangen: Increase und Insert

**Algorithmus 8** HeapIncreaseKey($A, i, key$)

```
1: if key < A[i] then
2:     Error
3: end if
4: A[i] = key
5: while i > 1 && A[Vater(i)] < A[i] do
6:     swap(A[i], A[Vater(i)])
7:     i = Vater(i)
8: end while
```

**Algorithmus 9** HeapInsert($A$, key)

```
1: heap-größe[A] = heap-größe[A] + 1
2: A[heap-größe[A]] = −∞
3: HeapInscreaseKey(A, heap-größe[A], key)
```

# Analysen

## Satz

1. *Die Operationen HeapMaximum, HeapExtractMax, HeapIncreaseKey und HeapInsert sind alle korrekt und arbeiten in $O(\log n)$ Zeit. HeapMaximum sogar in $\Theta(1)$.*

## Zur Übung

Genaueres zu den Korrektheitsbeweisen wieder zur Übung oder bei Schwierigkeiten zum Nachlesen im [Cormen].