Algorithmen und Datenstrukturen Kapitel 4

Neue Datenstrukturen, besseres (?) Sortieren (Teil 2)

Frank Heitmann
heitmann@informatik.uni-hamburg.de

4. November 2015

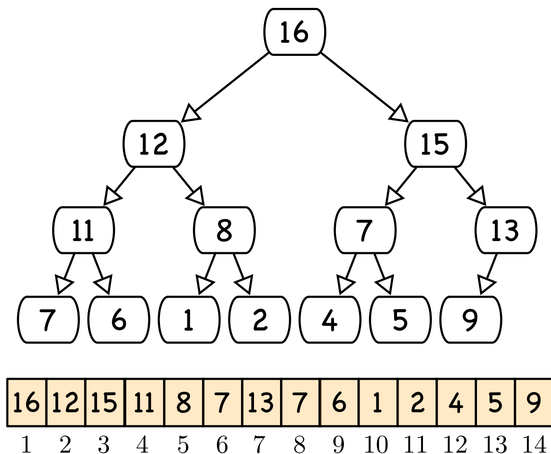
Heaps

Ein (binärer) *Heap* ist eine Datenstruktur, die als ein (fast) vollständiger binärer Baum angesehen werden kann, wobei der Baum noch die spezielle Heap-Eigenschaft erfüllt.

- Der Baum ist auf allen Ebenen vollständig gefüllt außer möglicherweise auf der letzten.
- Der Heap wird meist durch ein Array repräsentiert.
  - *länge*[A] ist die Anzahl der Elemente *des Feldes*
  - *heap-größe*[A] ist die Anzahl der Elemente *im Heap* (die in A gespeichert werden).
- *Heap-Eigenschaft*: Sei n ein Knoten des Baumes und p sein Vater, dann gilt
  - Max-Heap: $v(p) \geq v(n)$
  - Min-Heap: $v(p) \leq v(n)$

$\Rightarrow$ Gut für Sortieren und Warteschlangen.

Heap - Beispiel



Heaps und Arrays

Hat man ein Array, so werden die Elemente von links nach rechts gelesen und der binäre Baum von oben nach unten und von links nach rechts Ebene für Ebene aufgebaut.

Hat man einen Baum, so wird dieser von oben nach unten und von links nach rechts gelesen und dabei das Array von links nach rechts gefüllt.

Heaps und Arrays

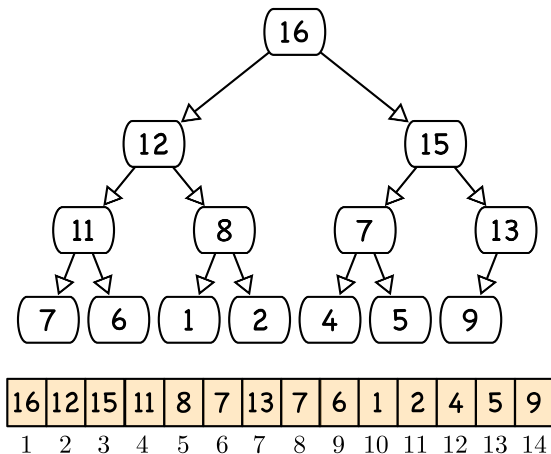
Damit sind einfache Operationen möglich. Sei i ein Index im Array, dann ist

$$\begin{aligned} Vater(i) &= \lfloor i/2 \rfloor \\ Left(i) &= 2i \\ Right(i) &= 2i+1 \end{aligned}$$

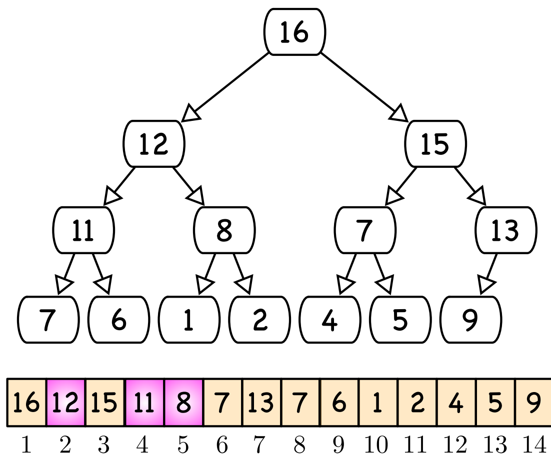
Die Max-Heap-Eigenschaft ist damit:

$$A[Vater(i)] \geq A[i]$$

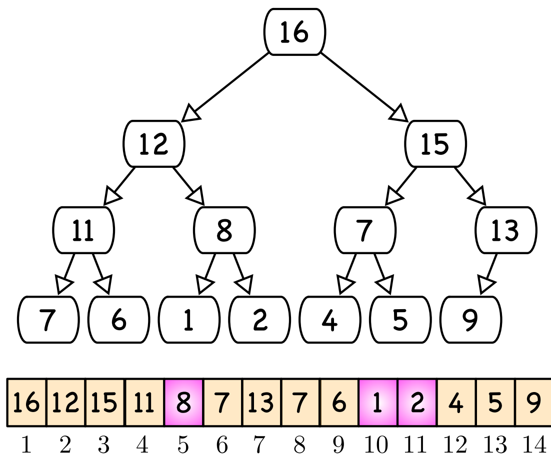
Heap - Beispiel



Heap - Beispiel (Vater/Kind)



Heap - Beispiel (Vater/Kind)



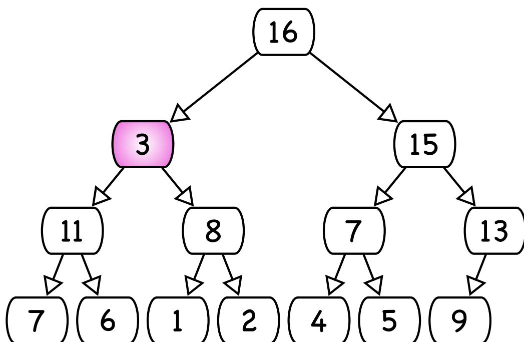
Aufrechterhaltung der Heap-Eigenschaft

Wie stellen wir aber gegeben ein Array (das noch kein Heap ist, sondern nur Werte enthält) einen Heap her? (Und das schnell!)

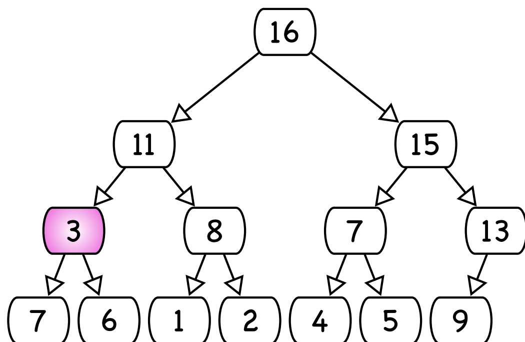
Eine wichtige Routine ist `MaxHeapify`(A, i).

Diese geht davon aus, dass $Left(i)$ und $Right(i)$ bereits Max-Heaps sind, dass jedoch $A[i]$ selbst die Heap-Eigenschaft verletzten kann. Die Routine lässt $A[i]$ 'absinken', so dass der bei $A[i]$ beginnende Teilbaum dann ein Max-Heap wird.

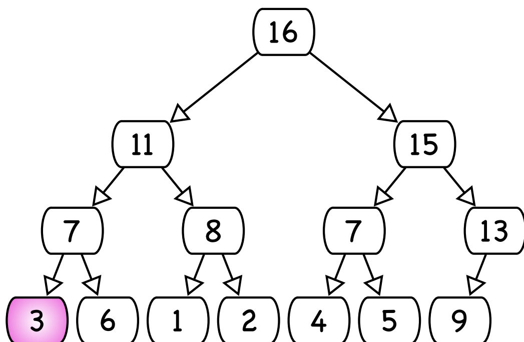
MaxHeapify - Beispiel



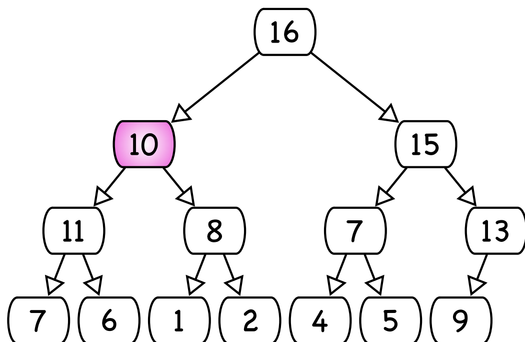
MaxHeapify - Beispiel



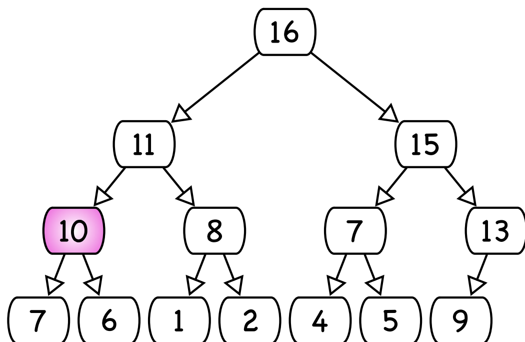
MaxHeapify - Beispiel



MaxHeapify - Beispiel 2



MaxHeapify - Beispiel 2



MaxHeapify - Die Idee

- Vergleiche $A[i]$ mit $A[Left(i)]$ und $A[Right(i)]$.
- Ist $A[i]$ am größten, so sind wir fertig.
- Sonst sei max der Index des größeren Elementes, tausche $A[i]$ und $A[max]$ (d.h. tausche $A[i]$ mit dem größeren Kind).
- Fahre nun mit $A[max]$ (hat den Wert von $A[i]$!) so wie eben mit $A[i]$ fort.

Anmerkung

Der Wert von $A[i]$ wandert im Baum nach unten. Bei allen Tauschoperationen ist dieser Wert beteiligt. Es finden sonst keine Tauschoperationen statt!

MaxHeapify - Analyse

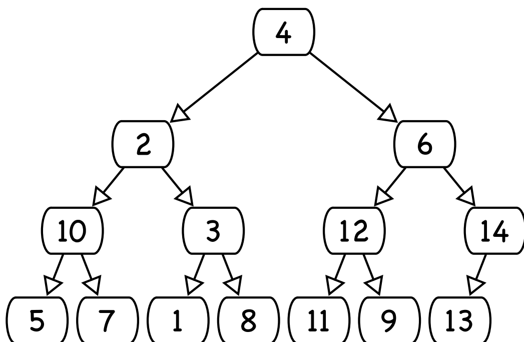
Satz

MaxHeapify ist korrekt und hat eine Laufzeit von $\Theta(\log n)$, wobei n die Anzahl der Knoten des Baumes ist.

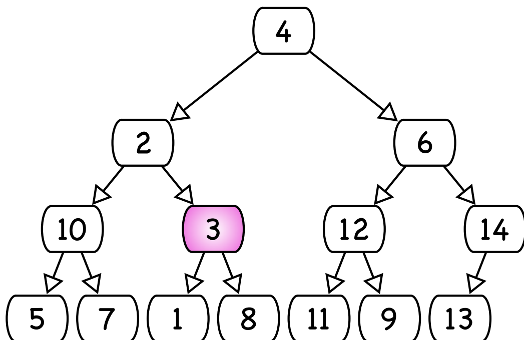
Diese Routine kann nun benutzt werden, um Schritt für Schritt einen korrekten MaxHeap aufzubauen.

Wie?

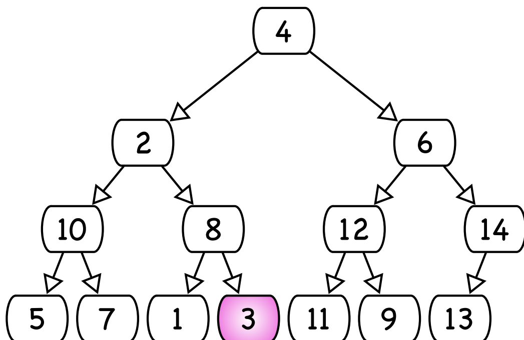
BuildMaxHeap - Beispiel



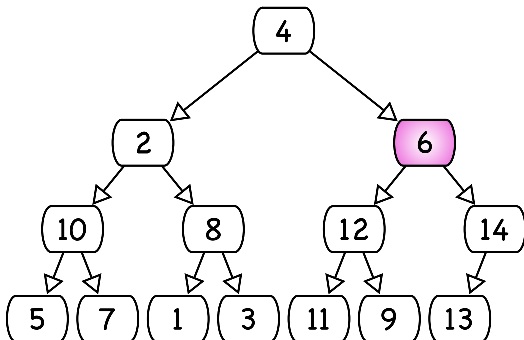
BuildMaxHeap - Beispiel



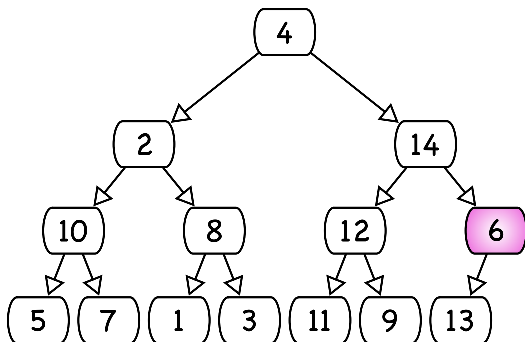
BuildMaxHeap - Beispiel



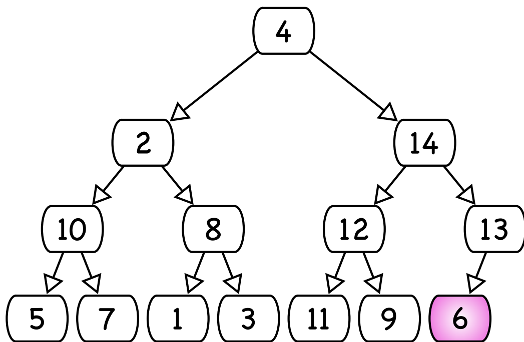
BuildMaxHeap - Beispiel



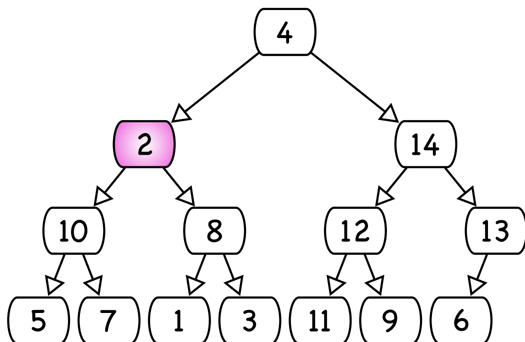
BuildMaxHeap - Beispiel



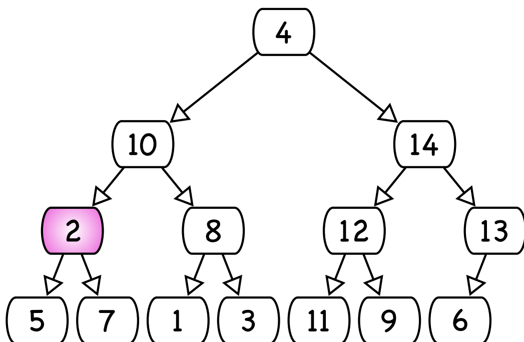
BuildMaxHeap - Beispiel



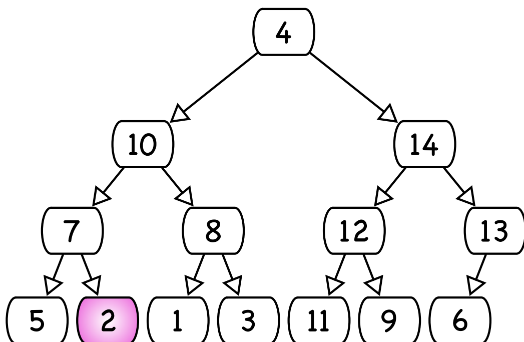
BuildMaxHeap - Beispiel



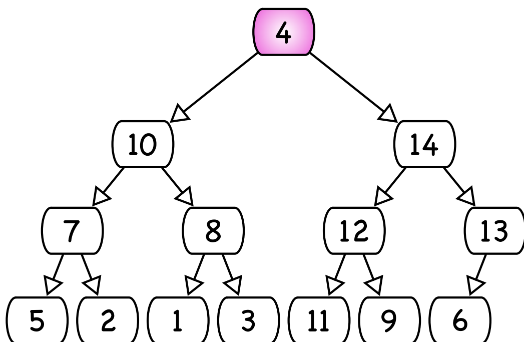
BuildMaxHeap - Beispiel



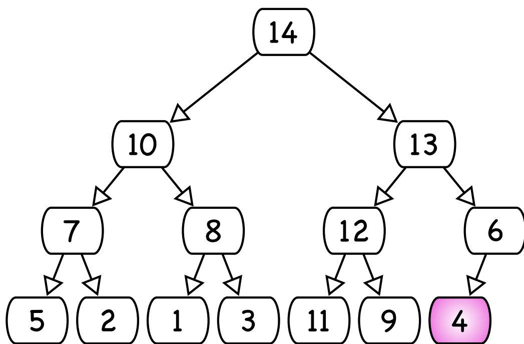
BuildMaxHeap - Beispiel



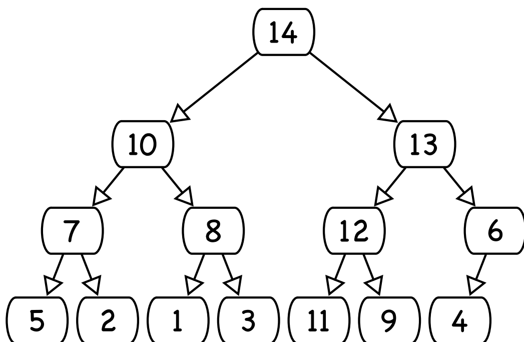
BuildMaxHeap - Beispiel



BuildMaxHeap - Beispiel



BuildMaxHeap - Beispiel



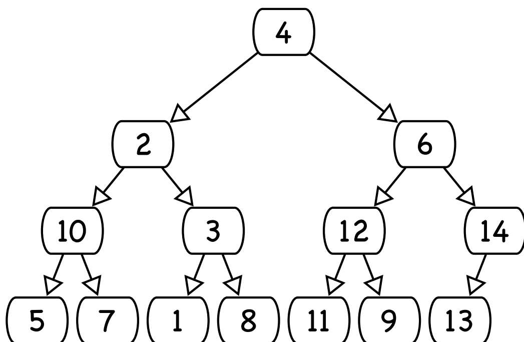
BuildMaxHeap - Die Idee und Pseudocode

1. Suche im Baum von unten nach oben und von rechts nach links (im Array also von rechts nach links) den ersten Knoten, der als Wurzel eines Teilbaumes betrachtet, kein Heap mehr ist.
2. Stelle die Heap-Eigenschaft her (mit einem Aufruf von `MaxHeapify`).
3. Fahre bei 1. fort.

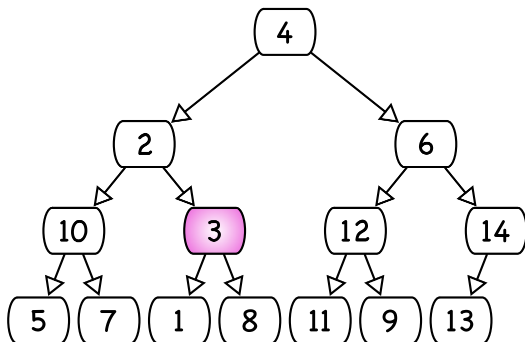
Algorithmus 1 BuildMaxHeap(A, i)

1: heap-größe[A] $=$ länge[A]
2: **for** $i = \lfloor \text{länge}[A]/2 \rfloor$ downto 1 **do**
3:     `MaxHeapify`(A, i)
4: **end for**

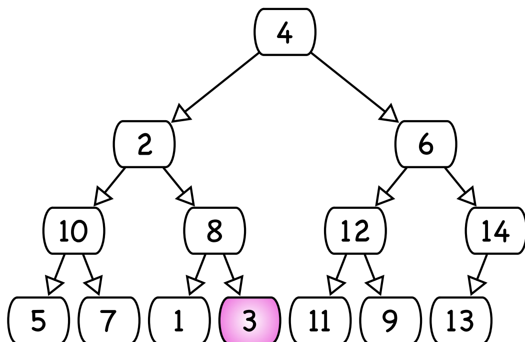
BuildMaxHeap - Beispiel (Wdh.)



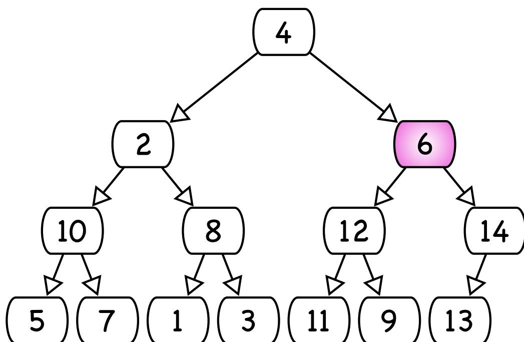
BuildMaxHeap - Beispiel (Wdh.)



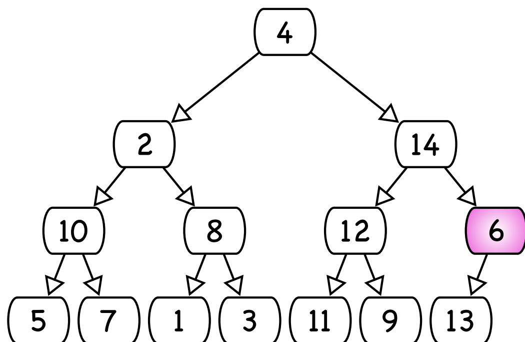
BuildMaxHeap - Beispiel (Wdh.)



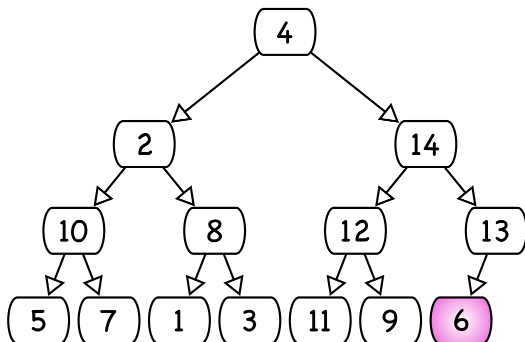
BuildMaxHeap - Beispiel (Wdh.)



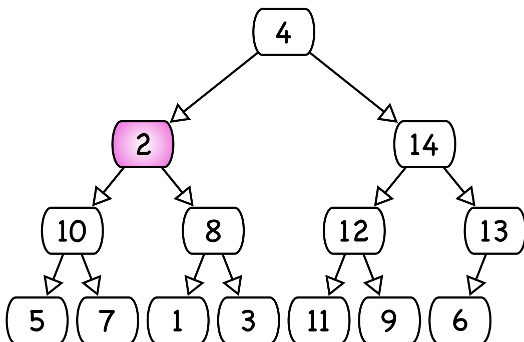
BuildMaxHeap - Beispiel (Wdh.)



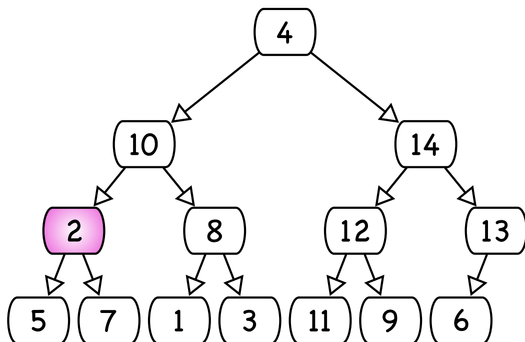
BuildMaxHeap - Beispiel (Wdh.)



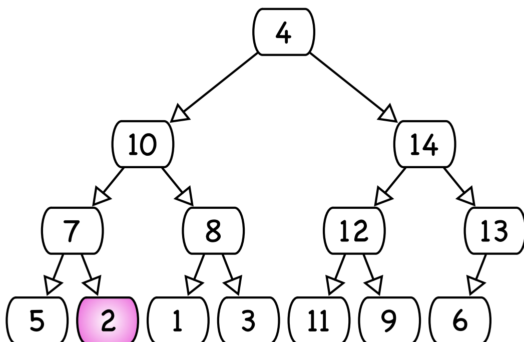
BuildMaxHeap - Beispiel (Wdh.)



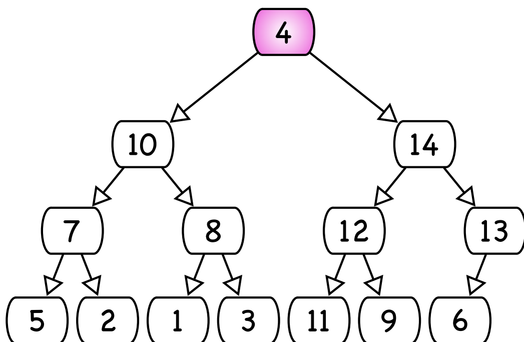
BuildMaxHeap - Beispiel (Wdh.)



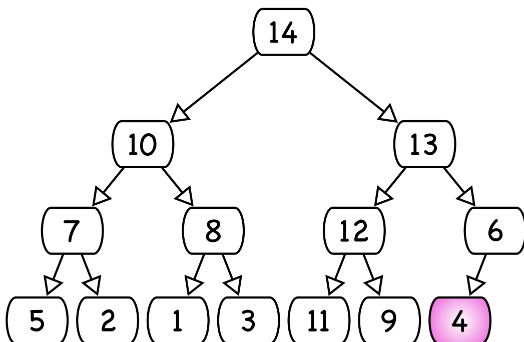
BuildMaxHeap - Beispiel (Wdh.)



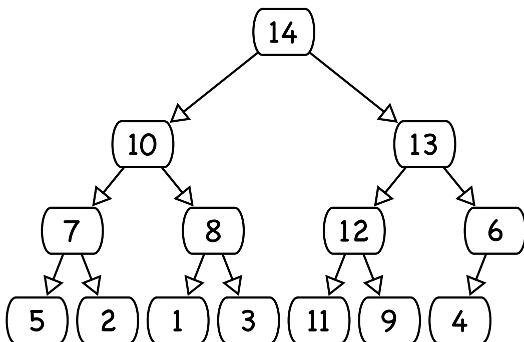
BuildMaxHeap - Beispiel (Wdh.)



BuildMaxHeap - Beispiel (Wdh.)



BuildMaxHeap - Beispiel (Wdh.)



BuildMaxHeap - Analyse

Algorithmus 2 BuildMaxHeap(A)

1: heap-größe[A] = länge[A]
2: **for** $i = \lfloor \text{länge}[A]/2 \rfloor$ downto 1 **do**
3: `MaxHeapify`(A, i)
4: **end for**

Zum Beweis der Korrektheit von BuildMaxHeap benutzt man die Korrektheit von MaxHeapify und folgende Schleifeninvariante:

> *Zu Beginn jeder Iteration der for-Schleife ist jeder Knoten $i+1, i+2, \ldots, n$ die Wurzel eines Max-Heap.*

BuildMaxHeap - Analyse

Satz

BuildMaxHeap ist korrekt und die Laufzeit ist durch $O(n \cdot \log n)$ *beschränkt. Eine genauere Analyse zeigt, dass die Laufzeit sogar durch* $O(n)$ *beschränkt ist. Wir können einen Heap also in Linearzeit herstellen.*

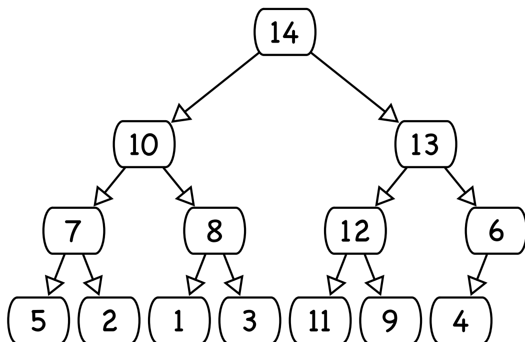
HeapSort

Wir wollen jetzt den Heap benutzen um einen effizienten Sortieralgorithmus zu entwickeln.

Gegeben ein Array von Zahlen, stellen wir zunächst den Heap her (mit BuildMaxHeap).

Und dann? ...

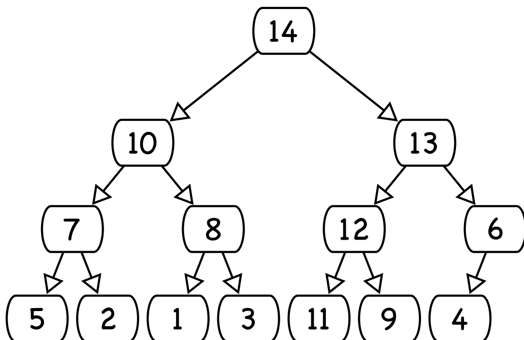
HeapSort - Die Idee ?



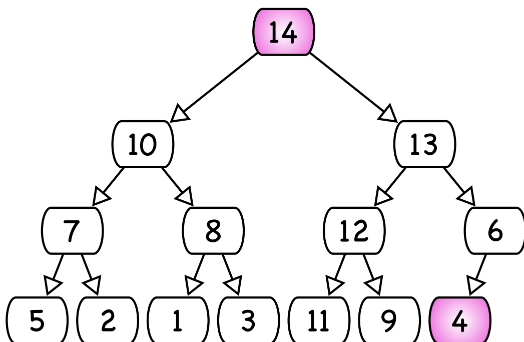
HeapSort - Die Idee

1. Gegeben ein Array, stelle zunächst einen Heap her (BuildMaxHeap).
2. Vertausche die Wurzel (maximales Element!) mit dem Element ganz rechts unten (das Element steht im Array ganz rechts! (Von den noch nicht behandelten Elementen)).
3. Verringere die Heap-Größe um 1 und führe MaxHeapify auf die Wurzel aus (die und nur die verletzt jetzt möglicherweise die Heap-Eigenschaft).
4. Fahre bei 2. fort.

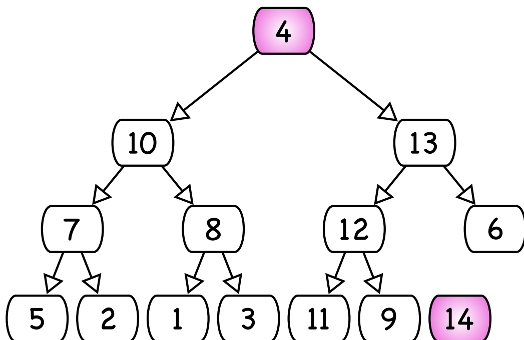
HeapSort - Beispiel



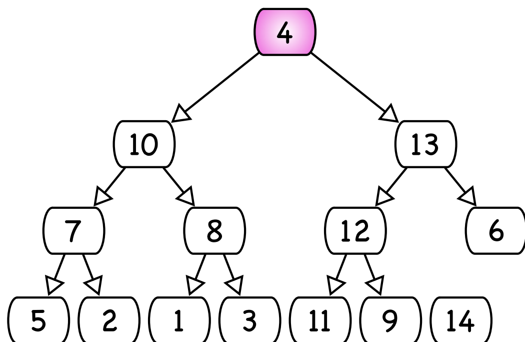
HeapSort - Beispiel



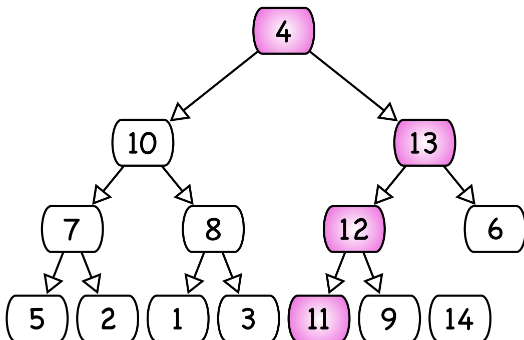
HeapSort - Beispiel



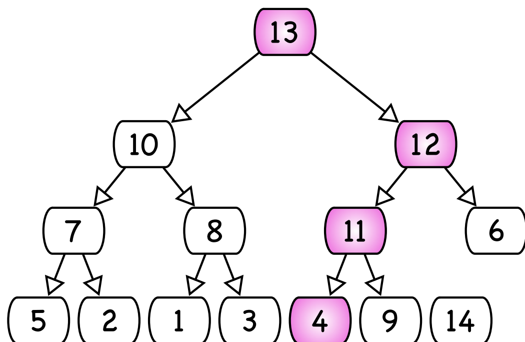
HeapSort - Beispiel



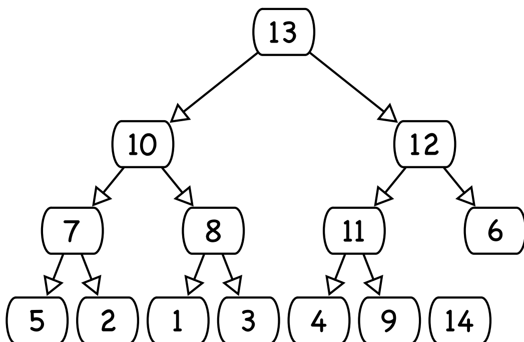
HeapSort - Beispiel



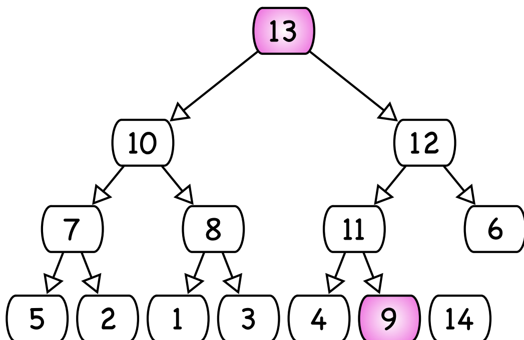
HeapSort - Beispiel



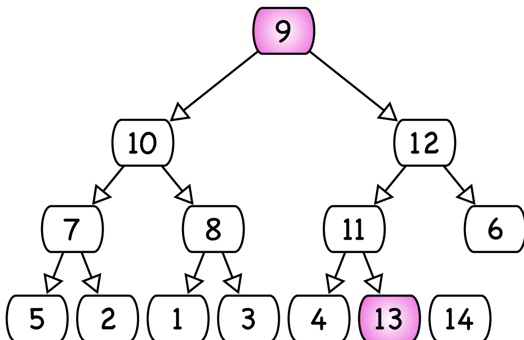
HeapSort - Beispiel



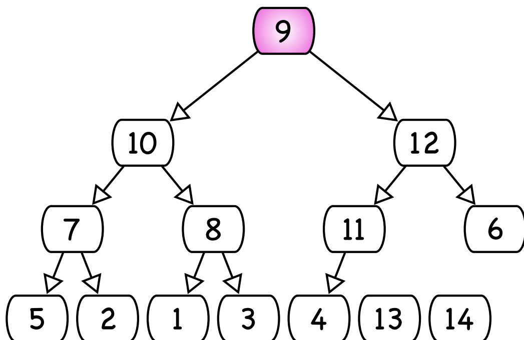
HeapSort - Beispiel



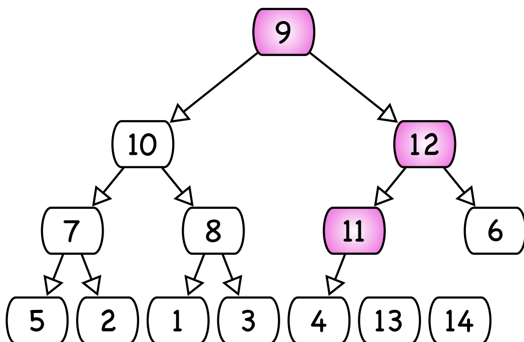
HeapSort - Beispiel



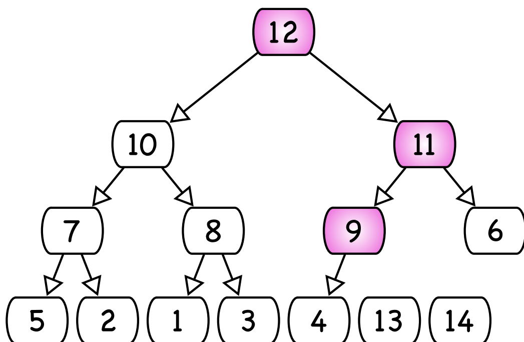
HeapSort - Beispiel



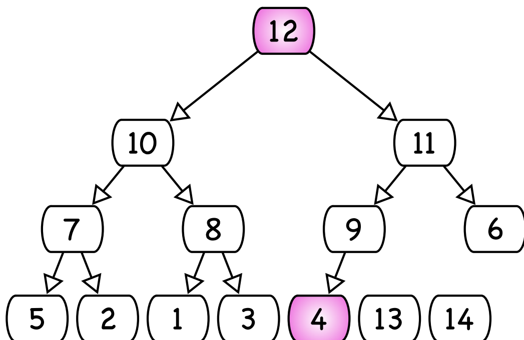
HeapSort - Beispiel



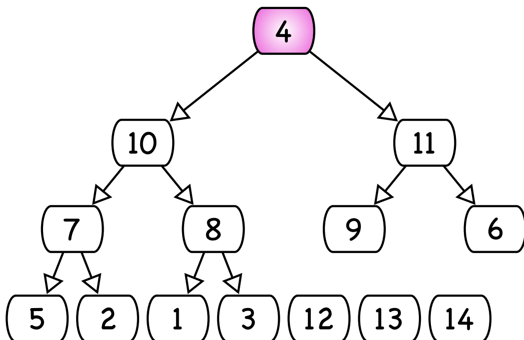
HeapSort - Beispiel



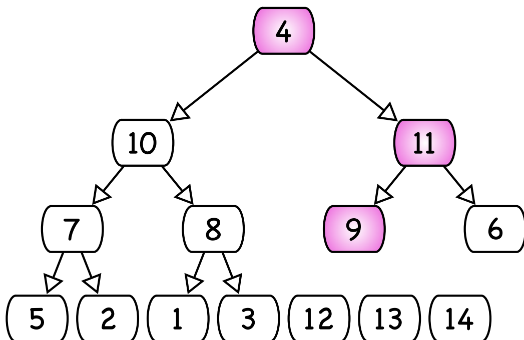
HeapSort - Beispiel



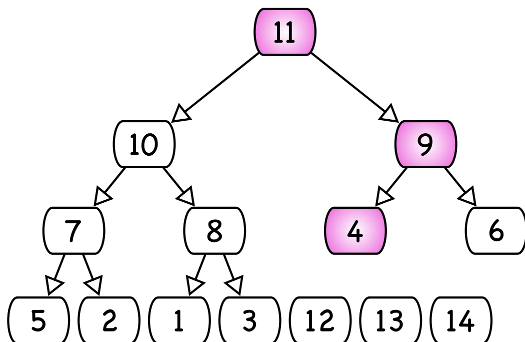
HeapSort - Beispiel



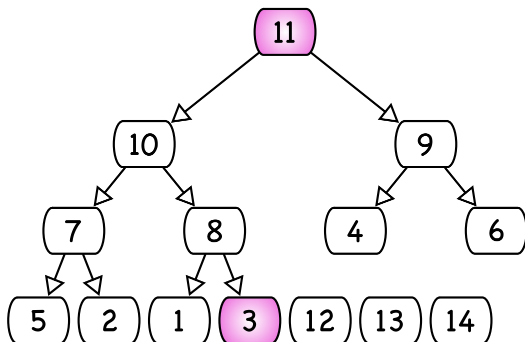
HeapSort - Beispiel



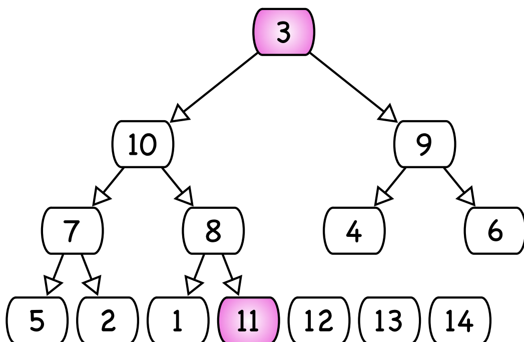
HeapSort - Beispiel



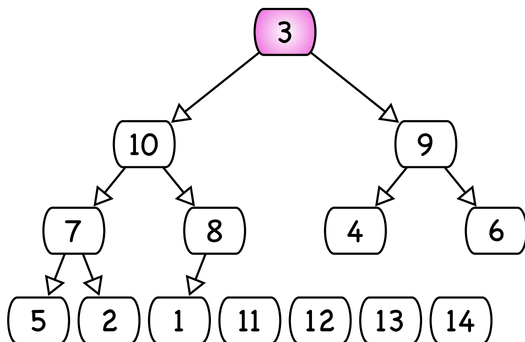
HeapSort - Beispiel



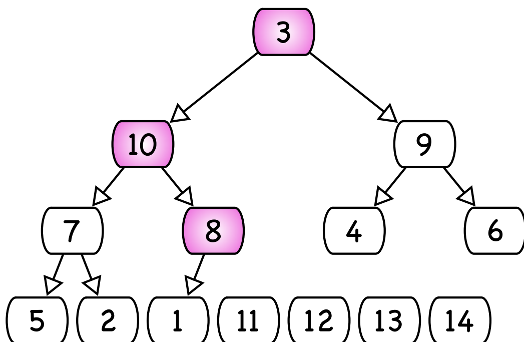
HeapSort - Beispiel



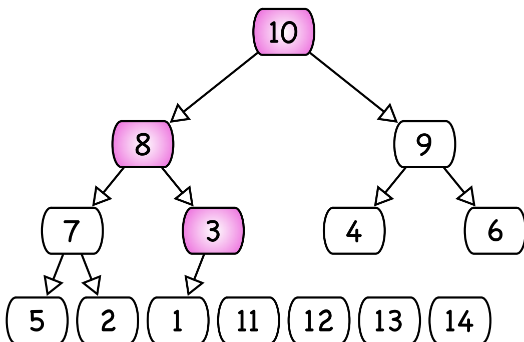
HeapSort - Beispiel



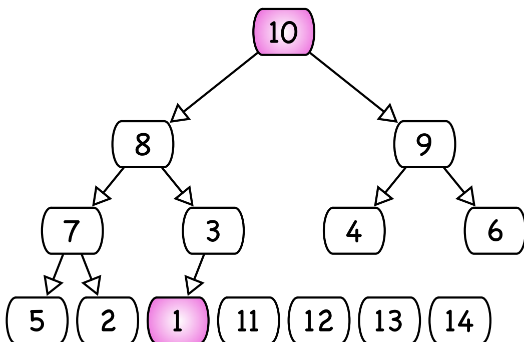
HeapSort - Beispiel



HeapSort - Beispiel



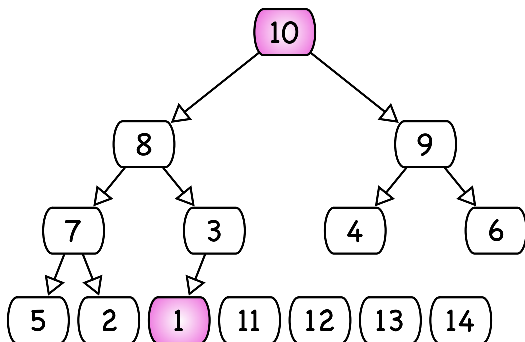
HeapSort - Beispiel



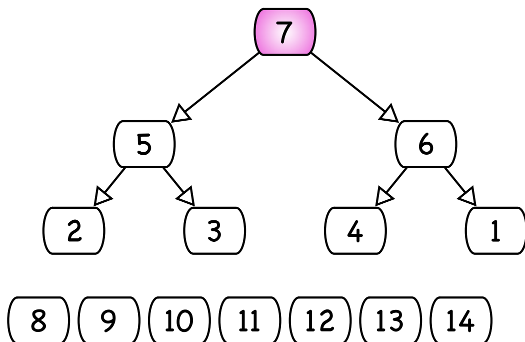
HeapSort - Beispiel

Wir überspringen ein paar Schritte...

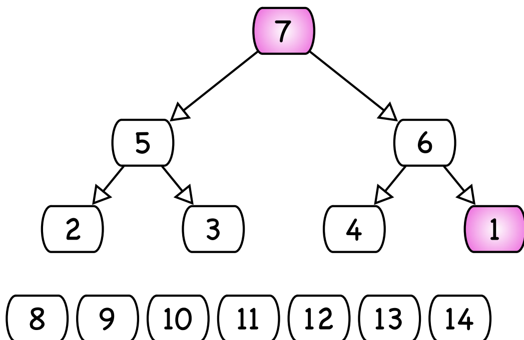
HeapSort - Beispiel



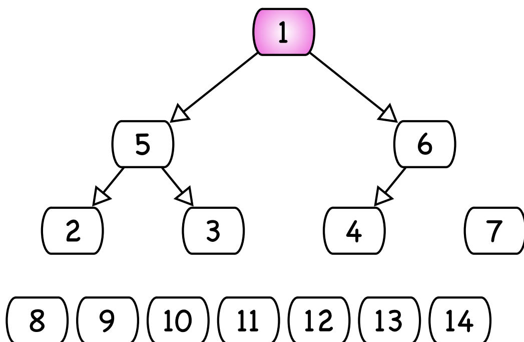
HeapSort - Beispiel



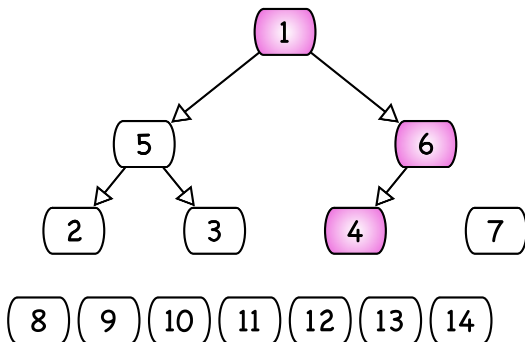
HeapSort - Beispiel



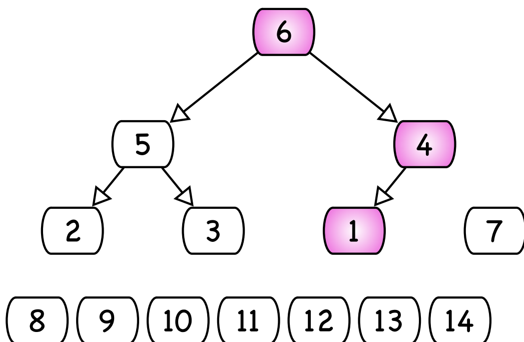
HeapSort - Beispiel



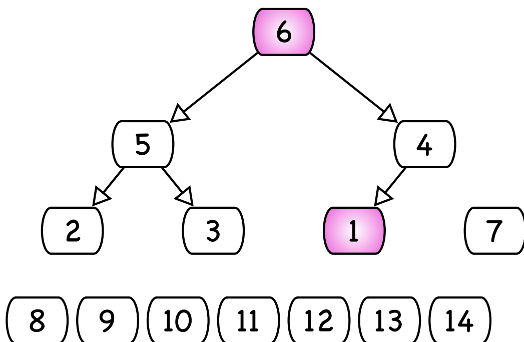
HeapSort - Beispiel



HeapSort - Beispiel



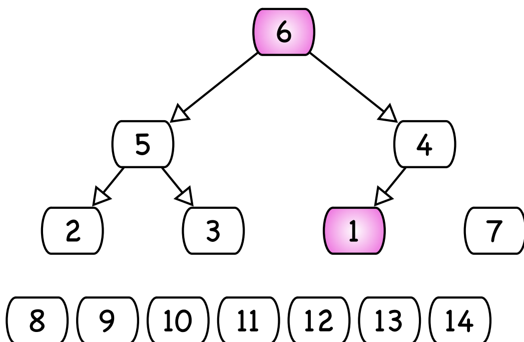
HeapSort - Beispiel



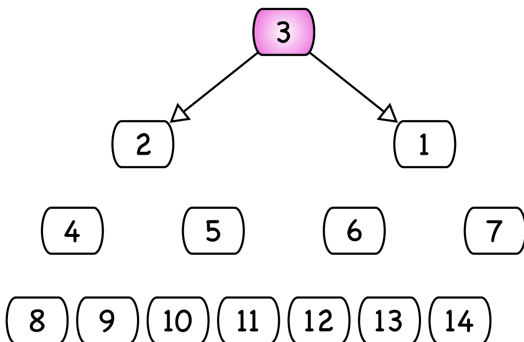
HeapSort - Beispiel

Wir überspringen nochmal ein paar Schritte...

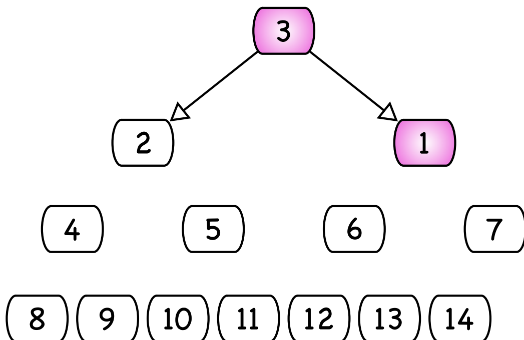
HeapSort - Beispiel



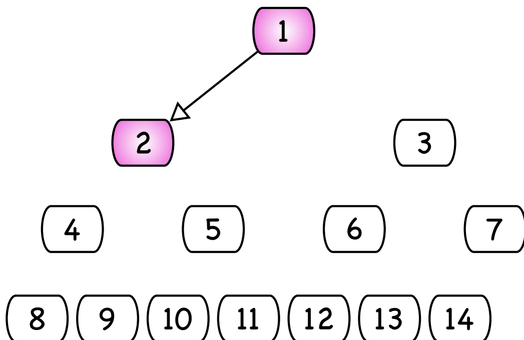
HeapSort - Beispiel



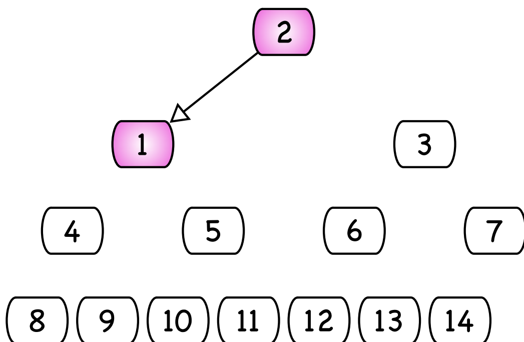
HeapSort - Beispiel



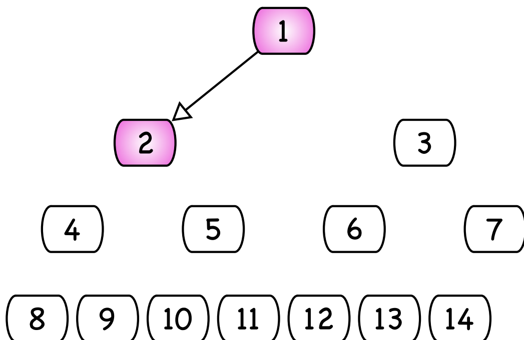
HeapSort - Beispiel



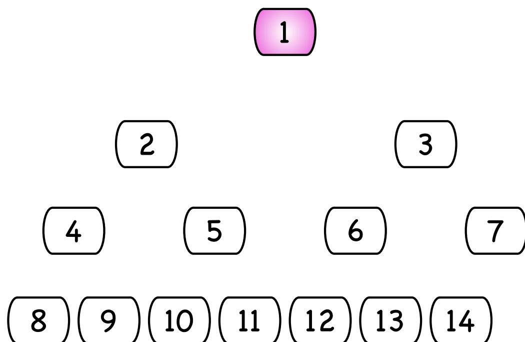
HeapSort - Beispiel



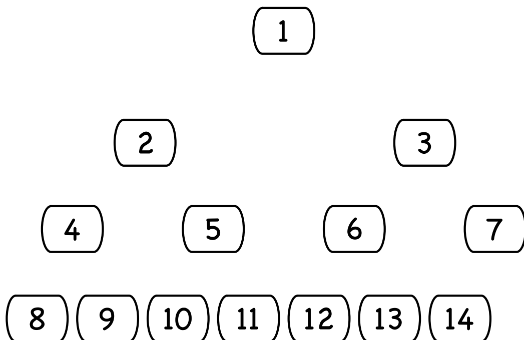
HeapSort - Beispiel



HeapSort - Beispiel



HeapSort - Beispiel



HeapSort - Die Idee

1. Gegeben ein Array, stelle zunächst einen Heap her (BuildMaxHeap).
2. Vertausche die Wurzel (maximales Element!) mit dem Element ganz rechts unten (das Element steht im Array ganz rechts! (Von den noch nicht behandelten Elementen)).
3. Verringere die Heap-Größe um 1 und führe MaxHeapify auf die Wurzel aus (die und nur die verletzt jetzt möglicherweise die Heap-Eigenschaft).
4. Fahre bei 2. fort.

HeapSort - Pseudocode

Algorithmus 3 HeapSort(A)

```
1: BuildMaxHeap(A)
2: for i = länge[A] downto 2 do
3:     swap(A[1], A[i])
4:     heap-größe[A] = heap-größe[A] − 1
5:     MaxHeapify(A, 1)
6: end for
```

Satz

HeapSort sortiert ein Array von n Zahlen in $\Theta(n \cdot \log n)$ Schritten.
HeapSort sortiert in-place.

Wiederholung: Heaps

Ein (binärer) *Heap* ist eine Datenstruktur, die als ein (fast) vollständiger binärer Baum angesehen werden kann, wobei der Baum noch die spezielle Heap-Eigenschaft erfüllt.

- Der Baum ist auf allen Ebenen vollständig gefüllt außer möglicherweise auf der letzten.
- Der Heap wird meist durch ein Array repräsentiert.
  - *länge*[*A*] ist die Anzahl der Elemente *des Feldes*
  - *heap-größe*[*A*] ist die Anzahl der Elemente *im Heap* (die in A gespeichert werden).
- *Heap-Eigenschaft*: Sei n ein Knoten des Baumes und p sein Vater, dann gilt
  - Max-Heap: $A[Vater(i)] \geq A[i]$
  - Min-Heap: $A[Vater(i)] \leq A[i]$

$\Rightarrow$ Gut für Sortieren und Warteschlangen.

Wiederholung: HeapSort

- Heaps zum Sortieren:
  - MaxHeapify zur Aufrechterhaltung der Heap-Eigenschaft
  - BuildMaxHeap zur Erstellung eines MaxHeaps (benutzt MaxHeapify)
  - HeapSort sortiert ein Array (benutzt MaxHeapify und BuildMaxHeap)

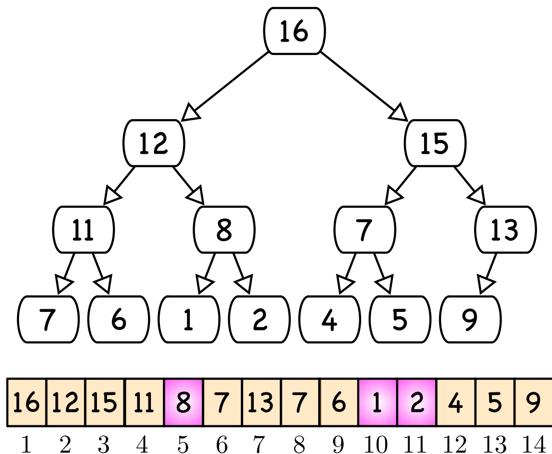
Wiederholung - Heaps und Arrays

Hat man ein Array, so werden die Elemente von links nach rechts gelesen und der binäre Baum von oben nach unten und von links nach rechts Ebene für Ebene aufgebaut.

Hat man einen Baum, so wird dieser von oben nach unten und von links nach rechts gelesen und dabei das Array von links nach rechts gefüllt.

$$\begin{aligned} Vater(i) &= \lfloor i/2 \rfloor \\ Left(i) &= 2i \\ Right(i) &= 2i+1 \end{aligned}$$

Heap - Beispiel



MaxHeapify - Die Idee

- Vergleiche $A[i]$ mit $A[Left(i)]$ und $A[Right(i)]$.
- Ist $A[i]$ am größten, so sind wir fertig.
- Sonst sei *max* der Index des größeren Elementes, tausche $A[i]$ und $A[max]$ (d.h. tausche $A[i]$ mit dem größeren Kind).
- Fahre nun mit $A[max]$ (hat den Wert von $A[i]$!) so wie eben mit $A[i]$ fort.

Anmerkung

Der Wert von $A[i]$ wandert im Baum nach unten. Bei allen Tauschoperationen ist dieser Wert beteiligt. Es finden sonst keine Tauschoperationen statt!

MaxHeapify - Pseudocode

Algorithmus 4 MaxHeapify(A, i)

1: $l = Left(i)$
2: $r = Right(i)$
3: **if** $l \leq$ heap-größe$[A]$ und $A[l] > A[i]$ **then**
4: $max = l$
5: **else**
6: $max = i$
7: **end if**
8: **if** $r \leq$ heap-größe$[A]$ und $A[r] > A[max]$ **then**
9: $max = r$
10: **end if**
11: **if** $max \neq i$ **then**
12: swap($A[i], A[max]$)
13: MaxHeapify(A, max)
14: **end if**

BuildMaxHeap - Die Idee und Pseudocode

1. Suche im Baum von unten nach oben und von rechts nach links (im Array also von rechts nach links) den ersten Knoten, der als Wurzel eines Teilbaumes betrachtet, kein Heap mehr ist.
2. Stelle die Heap-Eigenschaft her (mit einem Aufruf von `MaxHeapify`).
3. Fahre bei 1. fort.

Algorithmus 5 BuildMaxHeap(A, i)

1: heap-größe[A] $=$ länge[A]
2: **for** $i = \lfloor \text{länge}[A]/2 \rfloor$ downto 1 **do**
3: `MaxHeapify`(A, i)
4: **end for**

HeapSort - Die Idee

1. Gegeben ein Array, stelle zunächst einen Heap her (BuildMaxHeap).
2. Vertausche die Wurzel (maximales Element!) mit dem Element ganz rechts unten (das Element steht im Array ganz rechts! (Von den noch nicht behandelten Elementen)).
3. Verringere die Heap-Größe um 1 und führe MaxHeapify auf die Wurzel aus (die und nur die verletzt jetzt möglicherweise die Heap-Eigenschaft).
4. Fahre bei 2. fort.

HeapSort - Pseudocode

Algorithmus 6 HeapSort(A)

```
1: BuildMaxHeap(A)
2: for i = länge[A] downto 2 do
3:     swap(A[1], A[i])
4:     heap-größe[A] = heap-größe[A] − 1
5:     MaxHeapify(A, 1)
6: end for
```

Anmerkung

Ist das Array gar nicht vollständig gefüllt, sollte in der `for`-Schleife nicht länge[A] genutzt werden, sondern heap-größe[A] (die weiteren Plätze im Array werden ja gar nicht genutzt!).

Analysen

Satz

1. *MaxHeapify ist korrekt und hat eine Laufzeit von* $\Theta(\log n)$*, wobei* n *die Anzahl der Knoten des Baumes ist.*
2. *BuildMaxHeap ist korrekt und die Laufzeit ist durch* $O(n \cdot \log n)$ *beschränkt. Eine genauere Analyse zeigt, dass die Laufzeit sogar durch* $O(n)$ *beschränkt ist (also in* $\Theta(n)$ *ist). Wir können einen Heap also in Linearzeit herstellen.*
3. *HeapSort sortiert ein Array von* n *Zahlen in* $\Theta(n \cdot \log n)$ *Schritten. HeapSort sortiert in-place.*

Zur Übung

Wir haben nur kurz die Korrektheitsbeweise erwähnt oder angedeutet. Genaueres zur Übung oder bei Schwierigkeiten zum Nachlesen im [Cormen].