

Algorithmen und Datenstrukturen

Aufgabe 1 Arrays und Heaps (Beispiellösung)

- Kein Heap; fast Min-Heap bis auf $7 \leftrightarrow 6$.
- Min-Heap; Vaterknoten stets kleiner als Kindknoten.
- Kein Heap; beispielsweise Wurzel $4 < 5$ (linkes Kind), aber $4 > 3$ (rechtes Kind).
- Max-Heap; Vaterknoten stets größer als Kindknoten.

Aufgabe 2 Heap-Sort (Beispiellösung)

Array als fast vollständiger Binärbaum:

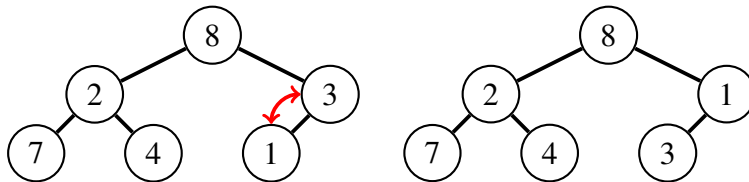


buildMinHeap: (**minHeapify** für alle Werte oder Positionen von hinten nach vorne)

- **minHeapify(1), minHeapify(4), minHeapify(7):**

↪ trivial bzw. überflüssig, da Blätter

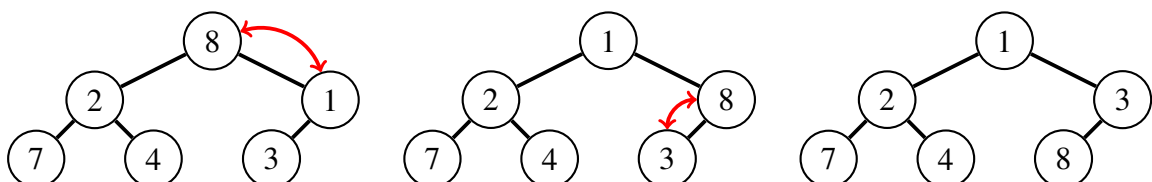
- **minHeapify(3):**



- **minHeapify(2):**

↪ nichts zu tun

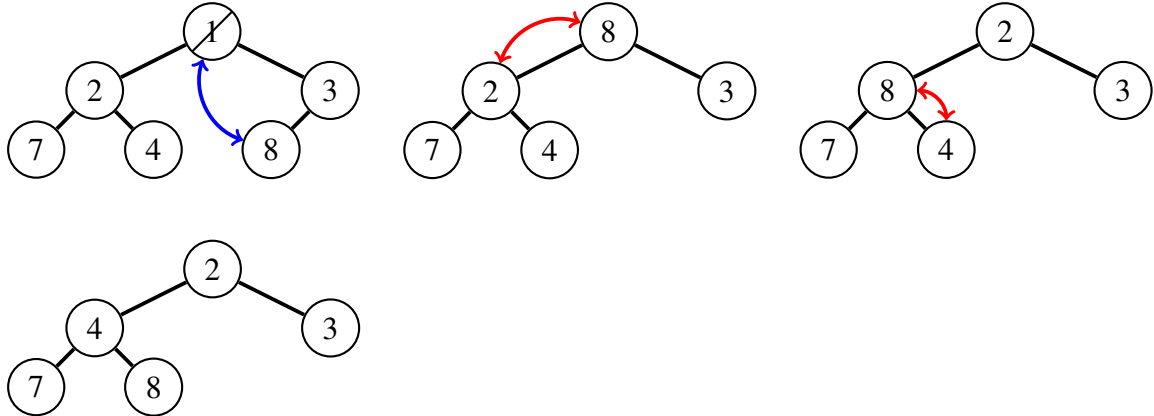
- **minHeapify(8):**



⇒ Heap nach **buildMinHeap**: (1,2,3,7,4,8)

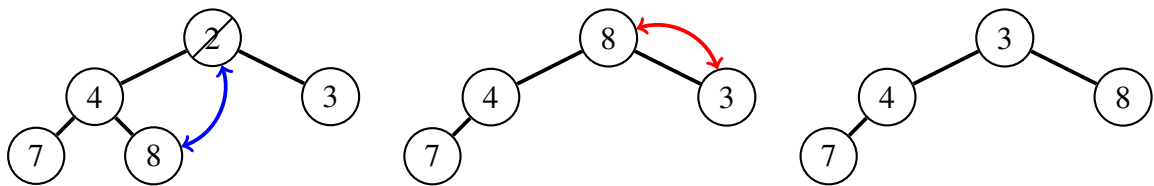
Sortierung: (**extractMin** bis Heap leer)

- **extractMin** = 1, dadurch 8 temporär neue Wurzel, folglich **minHeapify**(8)



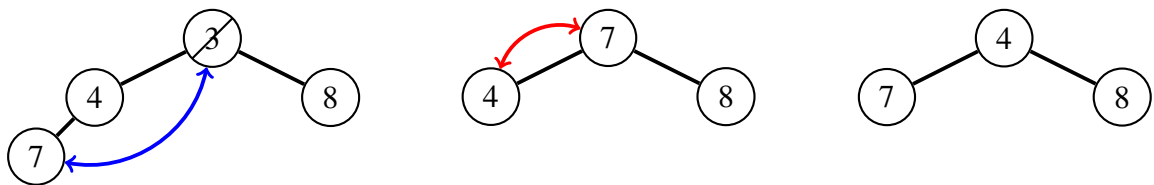
⇒ Heap nach **extractMin**: (2,4,3,7,8)

- **extractMin** = 2, dadurch 8 temporär neue Wurzel, folglich **minHeapify**(8)



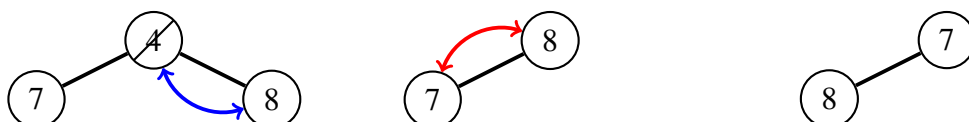
⇒ Heap nach **extractMin**: (3,4,8,7)

- **extractMin** = 3, dadurch 7 temporär neue Wurzel, folglich **minHeapify**(7)



⇒ Heap nach **extractMin**: (4,7,8)

- **extractMin** = 4, dadurch 8 temporär neue Wurzel, folglich **minHeapify**(8)



⇒ Heap nach **extractMin**: (7,8)

- **extractMin** = 7, dadurch 8 neue Wurzel



⇒ Heap nach **extractMin**: (8)

- **extractMin** = 8

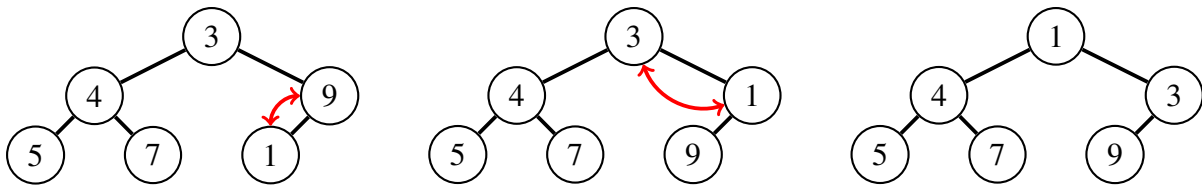
⇒ Heap nach **extractMin**: \emptyset

Aufgabe 3 Min-Priority Queue (Beispiellösung)

Array als fast vollständiger Binärbaum:

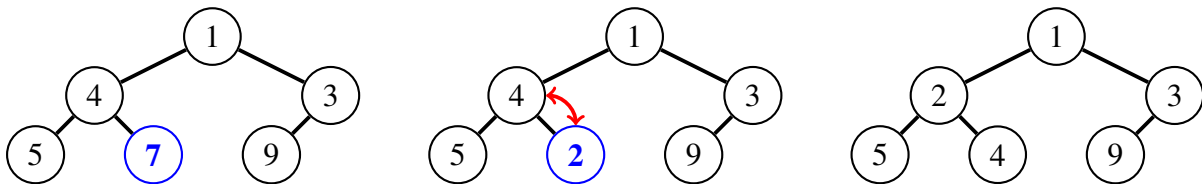


insert(1): (Einfügen am Ende, und Propagation aufwärts)



⇒ Heap nach **insert(1)**: (1, 4, 3, 5, 7, 9)

decreaseKey(7, 2): (Propagation aufwärts)



⇒ Heap nach **insert(1)** und **decreaseKey(7, 2)**: (1, 2, 3, 5, 4, 9)

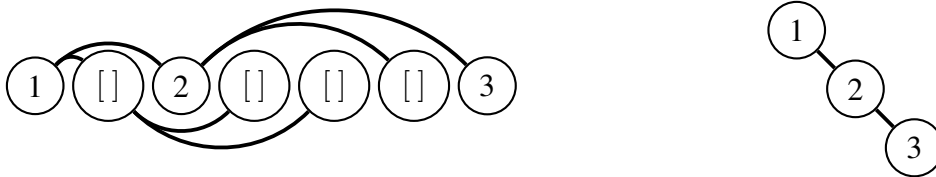
Aufgabe 4 Binäre Suchbäume (Beispiellösung)

a) Ja, es handelt sich um einen binären Suchbaum (V, E) , da für jeden inneren Knoten $v \in V$ gilt:

- für alle Knoten x im linken Teilbaum $v.left$ gilt: $key(x) \leq key(v)$
- für alle Knoten y im rechten Teilbaum $v.right$ gilt: $key(y) \geq key(v)$

Darüber hinaus gilt für jeden inneren Knoten $v \in V$, dass die Höhe des linken und des rechten Teilbaums gleich sind. Daher ist die AVL-Bedingung erfüllt und es handelt sich um einen AVL-Baum.

- b) Bei dem Baum handelt es sich erneut um einen binären Suchbaum. Allerdings ist bei dem Knoten 4 die AVL-Bedingung verletzt (Höhe des linken Teilbaums ist 2, während die Höhe des rechten Teilbaums 0 beträgt). Daher stellt der Baum keinen AVL-Baum dar.
- c) Es sich bei dem dargestellten Baum nicht um einen binären Suchbaum. Für den Knoten 8 ist das rechte Kind 7 kleiner. Da jeder AVL-Baum auch ein binärer Suchbaum sein muss, kann es sich somit auch nicht um einen AVL-Baum handeln.
- d) Zuerst erzeugen wir aus der Liste den binären Baum entsprechend der Vorlesung:

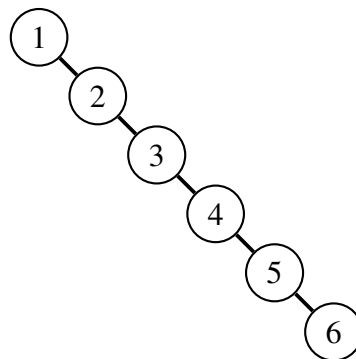


Wir stellen fest, dass es sich um einen binären Suchbaum handelt. Jedoch ist die AVL-Bedingung in der Wurzel verletzt, daher handelt es sich nicht um einen AVL-Baum.

- e) Der Grund, warum es nicht sinnvoll ist, Suchbäume als Liste zu speichern ist der, dass diese nicht (zwangsläufig) vollständig / fast vollständig sind. Daher können die entsprechenden Listen grösstenteils leer sein. Im Beispiel der vorherigen Teilaufgabe benötigen wir für den binären Suchbaum mit 3 Knoten eine Liste der Länge 7, wovon 4 Einträge leer sind.

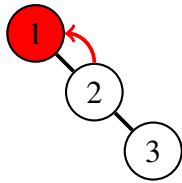
Aufgabe 5 Binärer Suchbaum vs. AVL-Baum (Beispiellösung)

- a) Fügen wir die Werte von 1 bis 6 nacheinander in einen binären Suchbaum ein, so erhalten wir einen entarteten Baum:

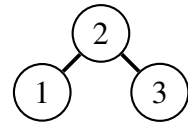


Um nun, ausgehend von der Wurzel, den Knoten 6 zu erreichen, benötigen wir 5 Schritte.

- b) Der Einfachheit halber geben wir nur die Schritte an, bei welchen eine Rotation benötigt wird um die AVL-Bedingung wieder herzustellen.
 - ...
 - **insert(3)**

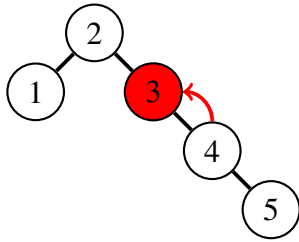


— Einfache Rotation —→

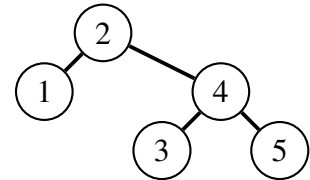


• ...

• **insert(5)**

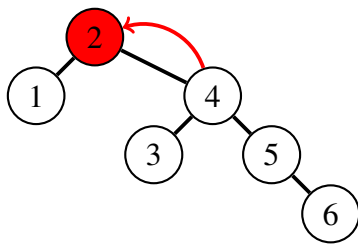


— Einfache Rotation —→

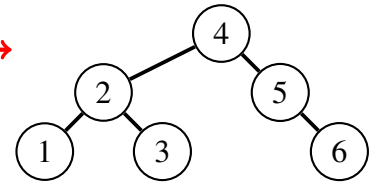


• ...

• **insert(6)**



— Einfache Rotation —→



Dieses Mal benötigen wir also 2 Schritte um zu dem Knoten 6.