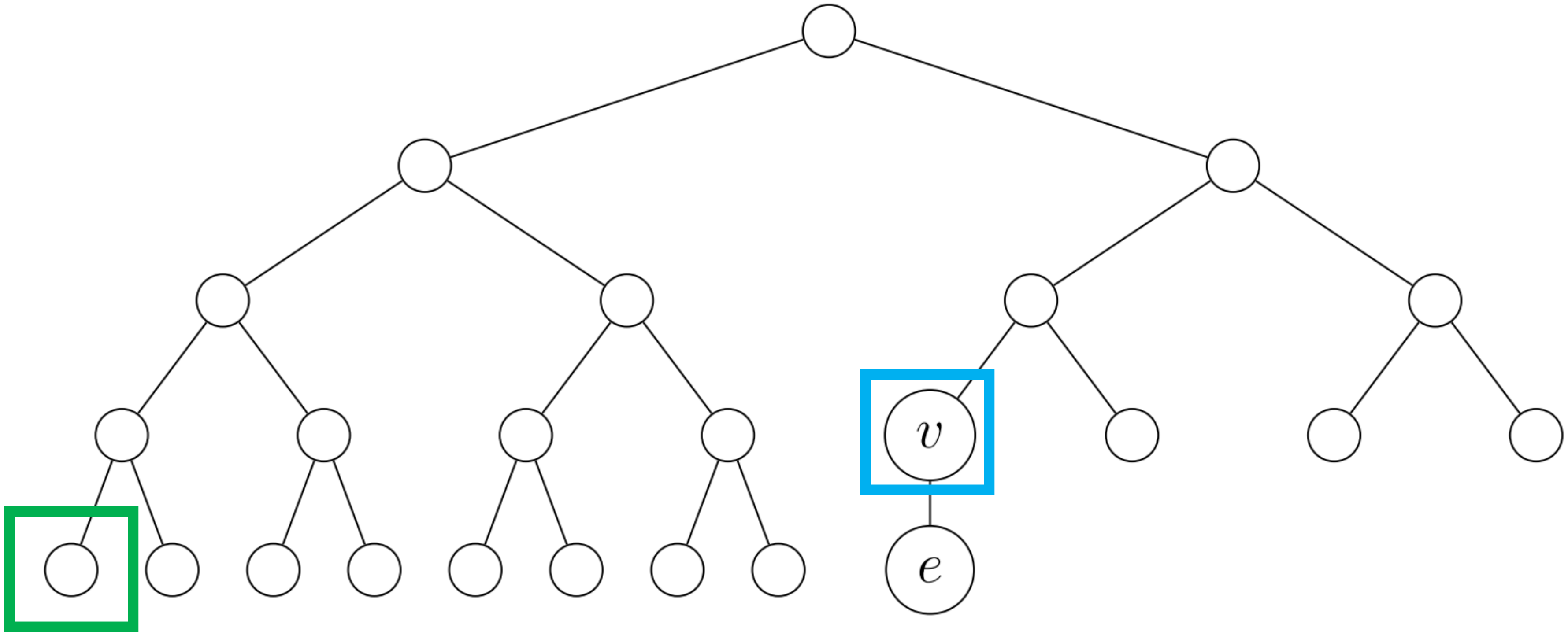
Tutorium Grundlagen: Algorithmen und Datenstrukturen

Übungsblatt Woche 6

Wiederholung: Definitionen

Keine Kinder = Blatt (für uns auch einzelne Wurzel)

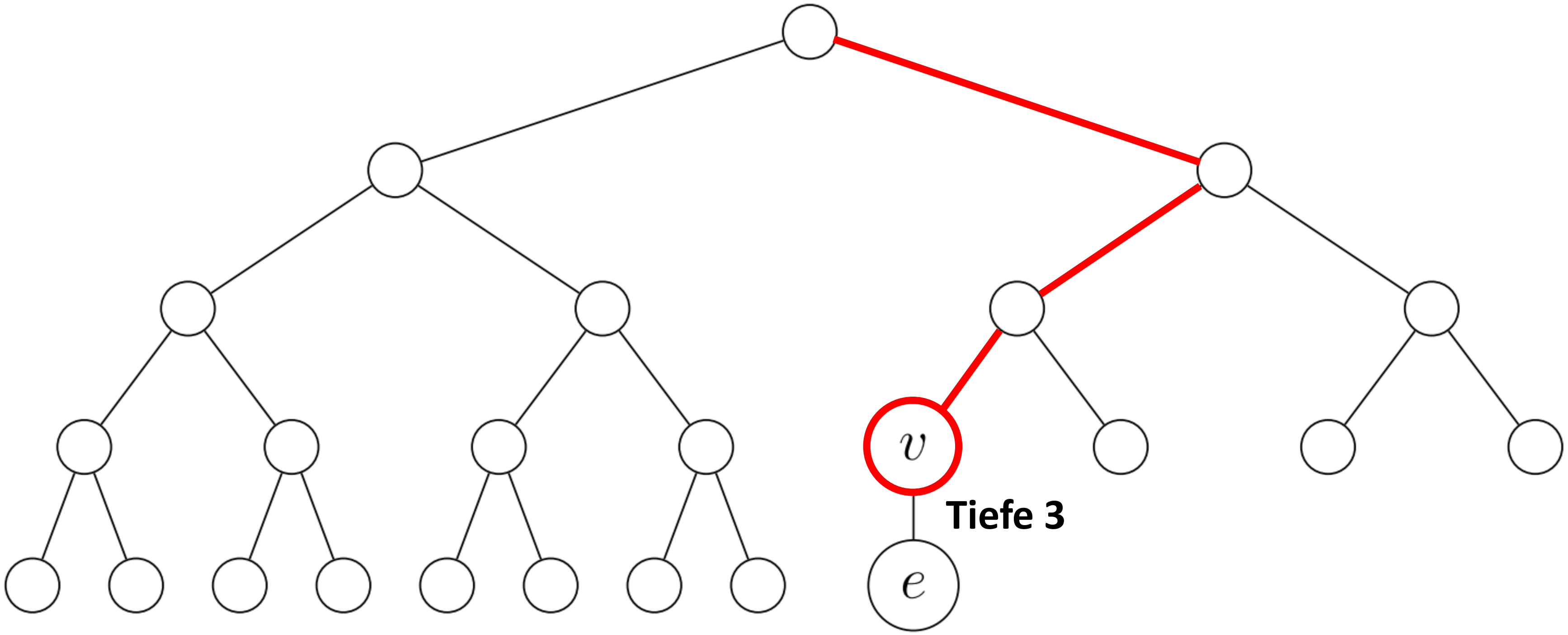
Sonst = innerer Knoten



Wiederholung: Definitionen

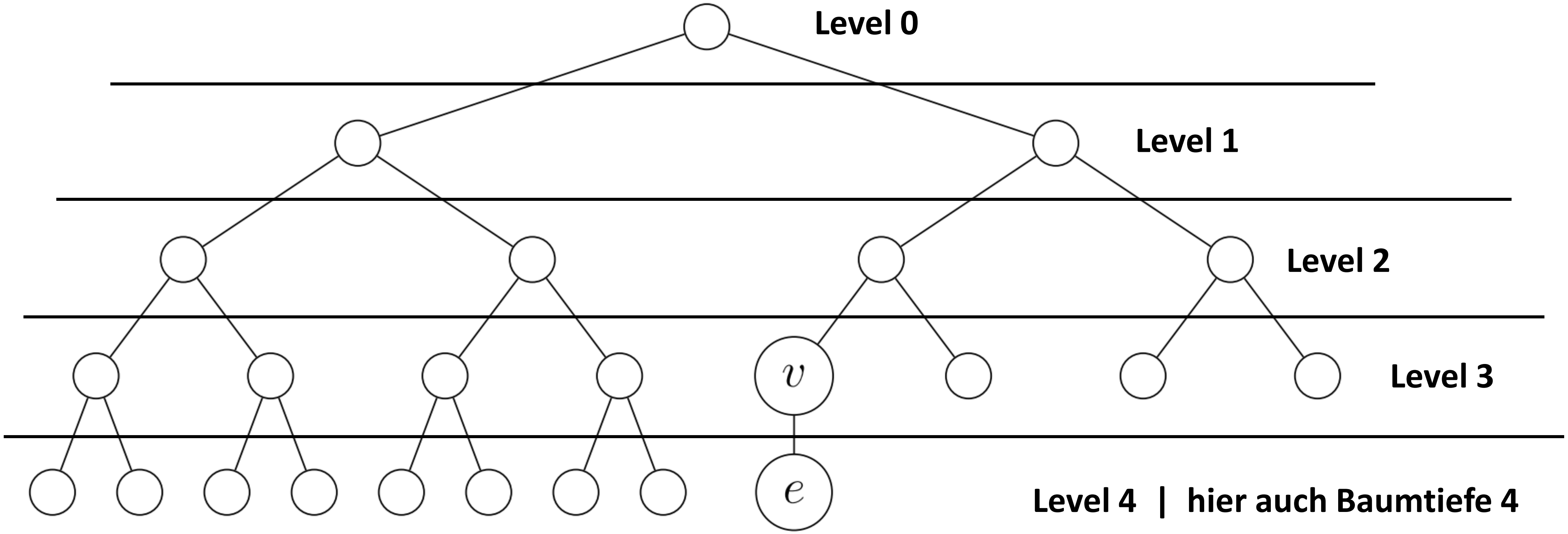
Tiefe eines Knotens = Entfernung von Wurzel gemessen in Kanten
(insbesondere hat die Wurzel Tiefe 0)

Achtung: Unterschiede in Literatur!!!



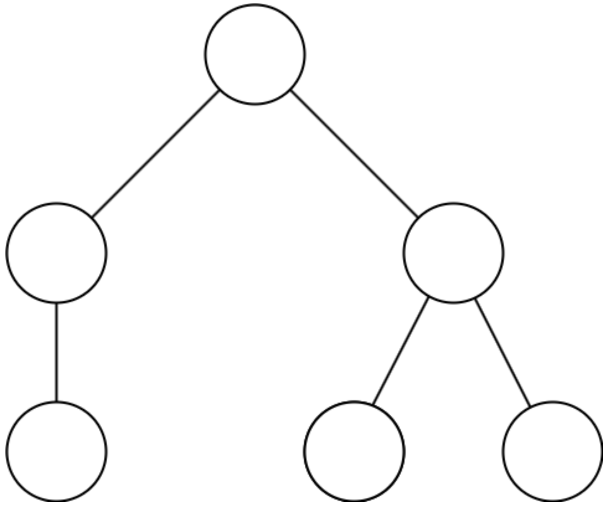
Wiederholung: Definitionen

Tiefe eines Baumes = Höchste Knotentiefe im Baum
i-tes Level eines Baumes = Menge aller Knoten mit Tiefe i

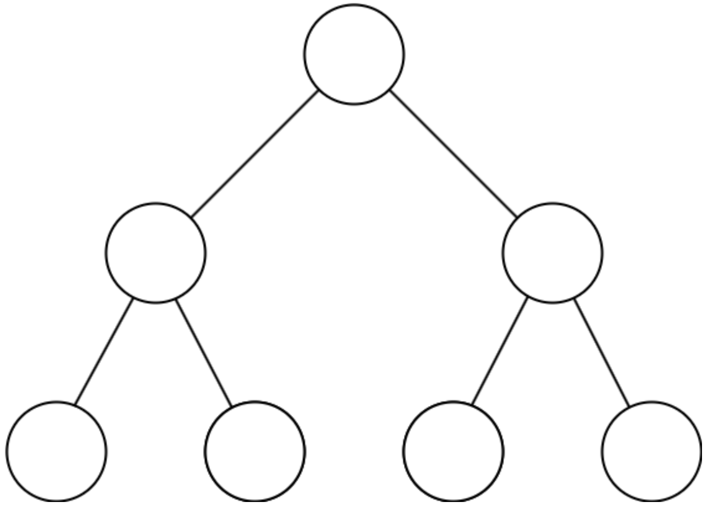


Wiederholung: Definitionen

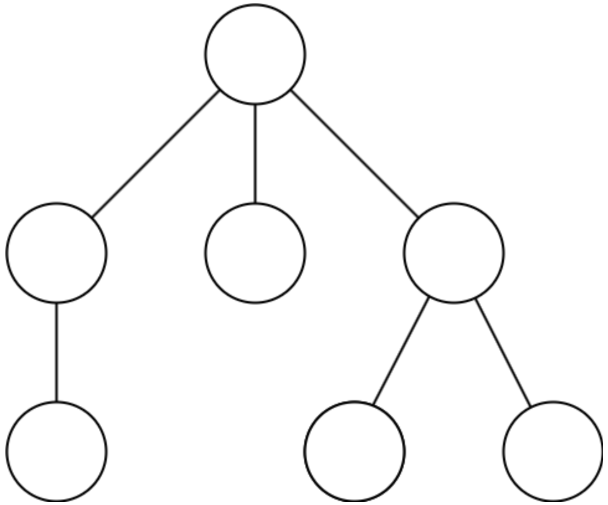
Binärbaum = Jeder innere Knoten hat ***maximal*** 2 Kinder
Echter Binärbaum = Jeder innere Knoten hat ***genau*** 2 Kinder



Binärbaum



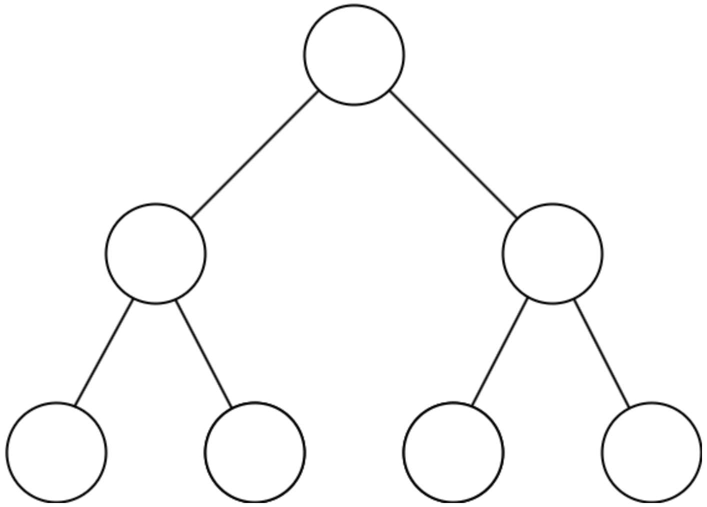
Echter Binärbaum



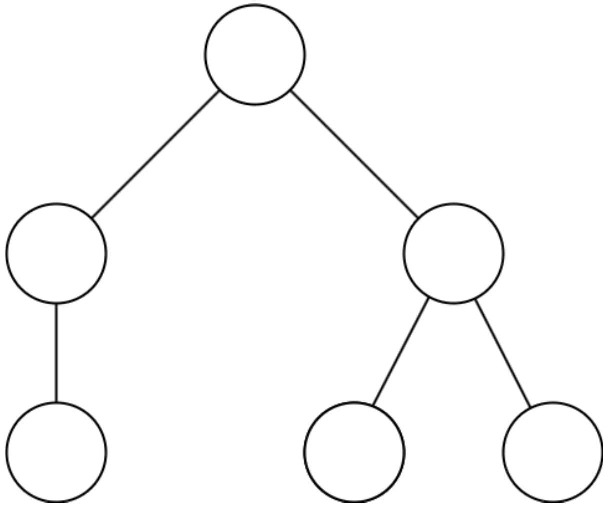
Kein Binärbaum

Wiederholung: Definitionen

Vollständiger Binärbaum = Alle Level des Baumes sind vollständig gefüllt
Fast vollständiger Binärbaum = Alle bis auf das tiefste Level des Baumes sind vollständig gefüllt und Umordnung ist möglich



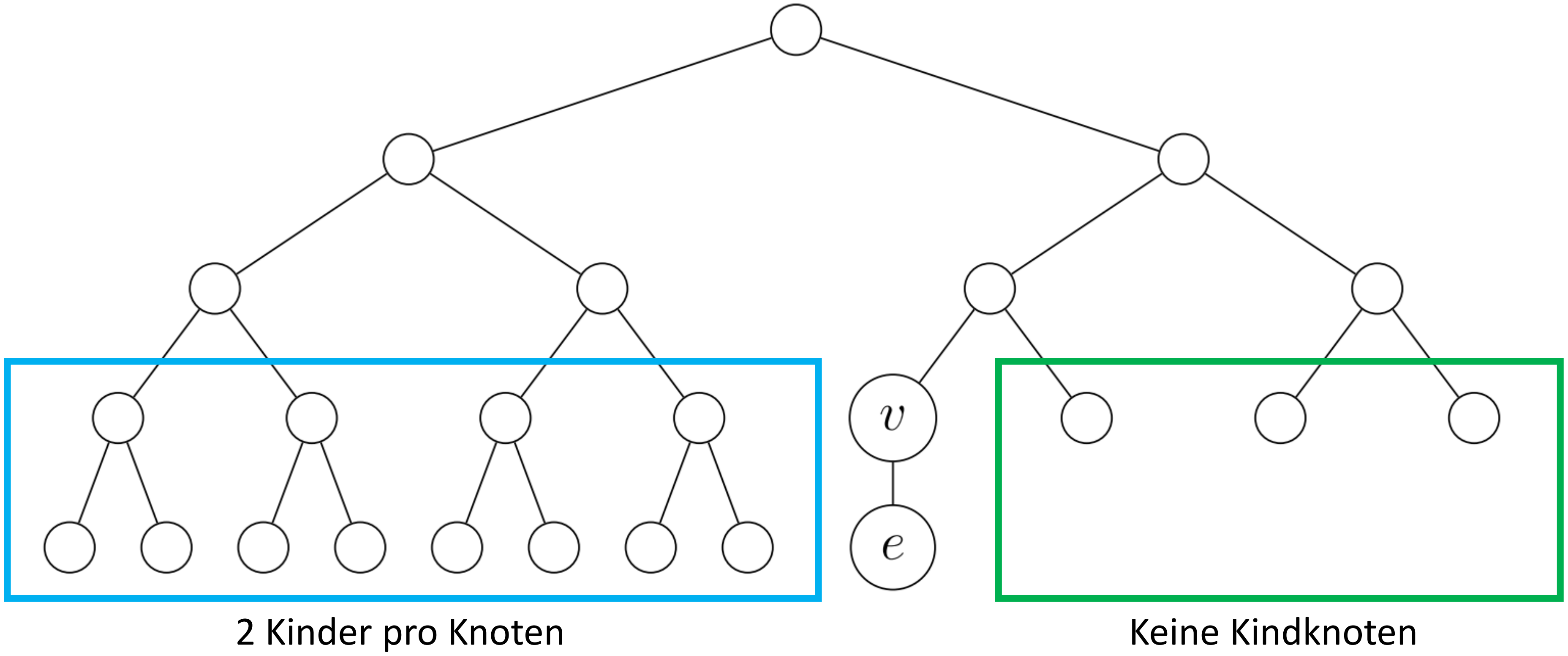
Vollständiger Binärbaum



Fast vollständiger Binärbaum

Wiederholung: Definitionen

Vollständiger Binärbaum = Alle Level des Baumes sind vollständig gefüllt
Fast vollständiger Binärbaum = Alle bis auf das tiefste Level des Baumes sind vollständig gefüllt und Umordnung ist möglich



Aufgabe 6.1
Vollständig

Aufgabe 6.1

In der Vorlesung wurde die folgende Aussage verwendet: Jeder fast vollständige Binärbaum mit $n \geq 1$ Knoten und Baumtiefe $t \geq 0$ erfüllt $2^t \leq n \leq 2^{t+1} - 1$. Beweisen Sie diese Aussage.

Aufgabe 6.2
Binärbaum mit b-Blättern

Aufgabe 6.2

Zeigen Sie, dass es für jedes $b \geq 1$ einen fast vollständigen echten Binärbaum mit b Blättern gibt.

Aufgabe 6.3
Binäre Heaps

Aufgabe 6.3

Führen Sie auf einem anfangs leeren Binären Heap (hier Min-Heap) folgende Operationen aus und stellen Sie die Zwischenergebnisse graphisch dar:

- build({15, 20, 9, 1, 11, 8, 4, 13, 17}),
- insert(7),
- delMin(),
- replaceKey(20, 3),
- delete(8).

Aufgabe 6.3

- build({15, 20, 9, 1, 11, 8, 4, 13, 17})

Aufgabe 6.3

- insert(7)

Aufgabe 6.3

- delMin()

Aufgabe 6.3

- replaceKey(20, 3)

Aufgabe 6.3

- delete(8)

Aufgabe 6.4
Binäre Heaps

Aufgabe 6.4

Prioritätswarteschlangen, wie beispielsweise durch Heaps implementiert, lassen sich auch problemlos zum Sortieren von Sequenzen verwenden. Insbesondere haben wir in der Vorlesung gelernt, dass sich binäre Heaps ohne zusätzlichen Speicherbedarf gut auf einem Feld umsetzen lassen. Dabei muss allerdings beachtet werden, dass die Funktion *deleteMin()* das jeweils kleinste Element zuerst mit dem letzten vertauscht, und dann aus dem Heap entfernt (bevor die Heap-Invariante per *siftDown* wiederhergestellt wird). Das sorgt dafür, dass man eine Sequenz, die in einem Feld gespeichert ist, mit einem normalen min-Heap absteigend sortiert, statt wie sonst aufsteigend.
Gegeben ist die folgende Sequenz, die bereits so in einem Feld im Speicher vorliegt: [94, 34, 55, 14, 37, 74]
Sie können davon ausgehen, dass jede Zahl immer genau ein Speicherfeld belegt. Sortieren Sie das Feld nun wie gewohnt aufsteigend (also mit dem kleinsten Element an erster Stelle), indem Sie Heapsort auf Basis eines Max-Heaps ausführen. Ein Max-Heap verhält sich genauso wie ein Min-Heap, mit dem Unterschied, dass die Priorität eines Knotens immer größer oder gleich der Priorität seiner Kindknoten sein muss.

a) Führen Sie zunächst die Operation *build* auf dem Feld aus, und stellen Sie die MaxHeap-Invariante sicher, indem Sie passende *siftDown*-Operationen ausführen.
b) Führen Sie nun so lange *deleteMax* aus, bis kein Element mehr im Heap ist. Vertauschen Sie dazu wie gelernt das erste (hier: maximale!) Element mit dem letzten im Heap. Anstatt nun das letzte Element zu löschen, ignorieren Sie für alle weiteren Operationen den Speicher von diesem und allen nachfolgenden Elementen. Mit dieser Einschränkung können Sie nun wie gewohnt *siftDown* auf dem obersten Element ausfuhren, bis die Heap-Invariante wiederhergestellt wurde.

Aufgabe 6.4 (a)

- build({94, 34, 55, 14, 37, 74})

Aufgabe 6.4 (a)

- 1. deleteMax()

Aufgabe 6.4 (a)

- 2. deleteMax()

Aufgabe 6.4 (a)

- 3. deleteMax()

Aufgabe 6.4 (a)

- 4. deleteMax()

Aufgabe 6.4 (a)

- 5. deleteMax()

Aufgabe 6.4 (a)

- 6. deleteMax()