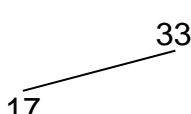
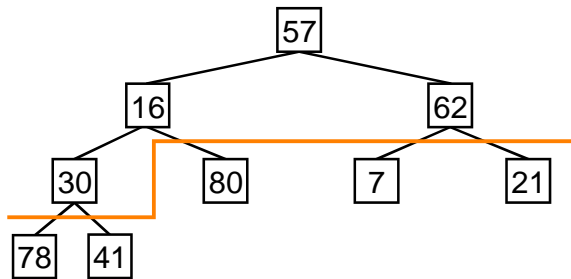


VL10, Lösung 1

- a) Heaps sind ausgeglichene binäre Bäume, bei denen jeder Knoten einen größeren Wert enthält als alle Kind-Knoten. Dadurch befindet sich das größte Element immer in der Wurzel. Es existiert eine Abbildung zwischen einem Array beliebiger Größe und dem zugehörigen Heap, worauf Heapsort basiert.
- b) Bei den ersten beiden Bäumen handelt es sich um Heaps, beim letzten Baum jedoch nicht: die 50 ist größer als die 33, und müsste daher an der Wurzel stehen.
- c)
- 

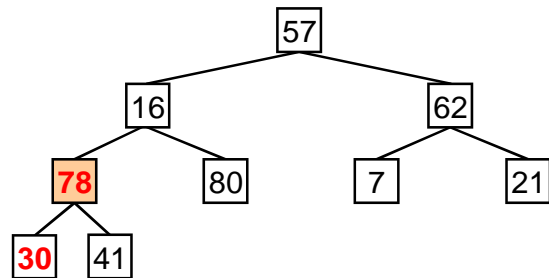
VL10, Lösung 2

Phase 1:



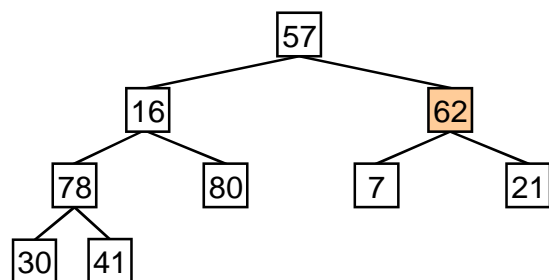
Array-Darstellung:

57	16	62	30	80	7	21	78	41
----	----	----	----	----	---	----	----	----



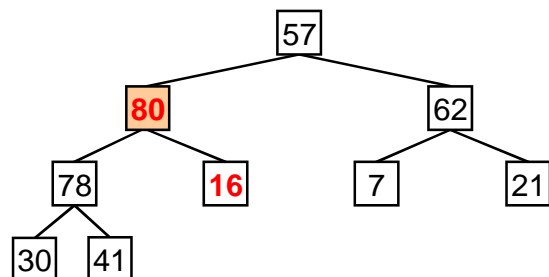
Array-Darstellung:

57	16	62	78	80	7	21	30	41
----	----	----	----	----	---	----	----	----



Array-Darstellung:

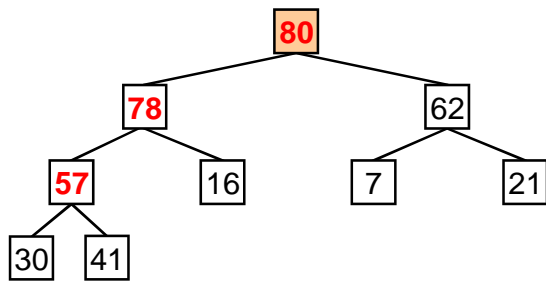
57	16	62	78	80	7	21	30	41
----	----	----	----	----	---	----	----	----



Array-Darstellung:

57	80	62	78	16	7	21	30	41
----	----	----	----	----	---	----	----	----

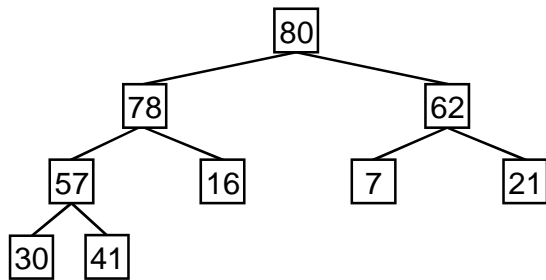
Algorithmen und Datenstrukturen



Musterlösung, SS2021

Array-Darstellung:

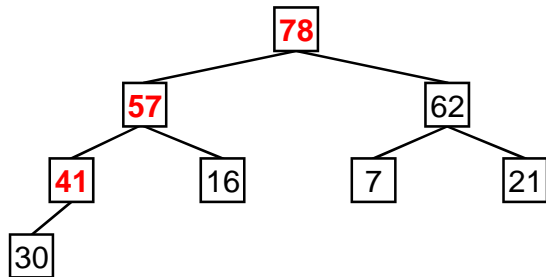
80	78	62	57	16	7	21	30	41
----	----	----	----	----	---	----	----	----



Array-Darstellung:

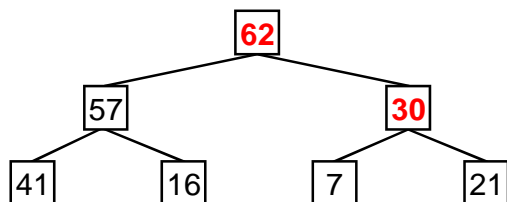
80	78	62	57	16	7	21	30	41
----	----	----	----	----	---	----	----	----

Phase 2:



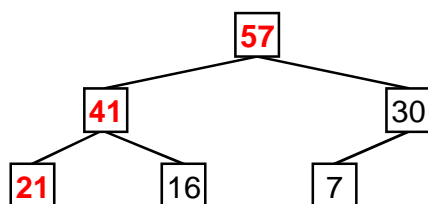
Array-Darstellung:

78	57	62	41	16	7	21	30	80
----	----	----	----	----	---	----	----	----



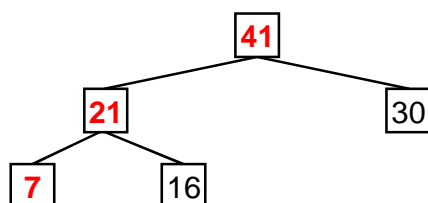
Array-Darstellung:

62	57	30	41	16	7	21	78	80
----	----	----	----	----	---	----	----	----



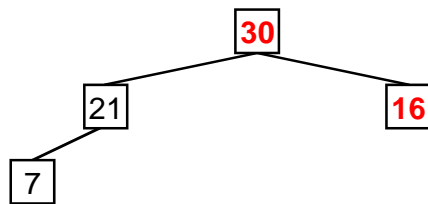
Array-Darstellung:

57	41	30	21	16	7	62	78	80
----	----	----	----	----	---	----	----	----



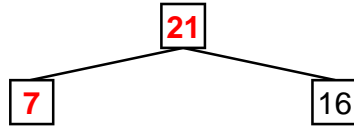
Array-Darstellung:

41	21	30	7	16	57	62	78	80
----	----	----	---	----	----	----	----	----



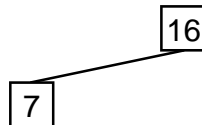
Array-Darstellung:

30	21	16	7	41	57	62	78	80
----	----	----	---	----	----	----	----	----



Array-Darstellung:

21	7	16	30	41	57	62	78	80
----	---	----	----	----	----	----	----	----



Array-Darstellung:

16	7	21	30	41	57	62	78	80
----	---	----	----	----	----	----	----	----



Array-Darstellung:

7	16	21	30	41	57	62	78	80
---	----	----	----	----	----	----	----	----

Array-Darstellung:

7	16	21	30	41	57	62	78	80
---	----	----	----	----	----	----	----	----

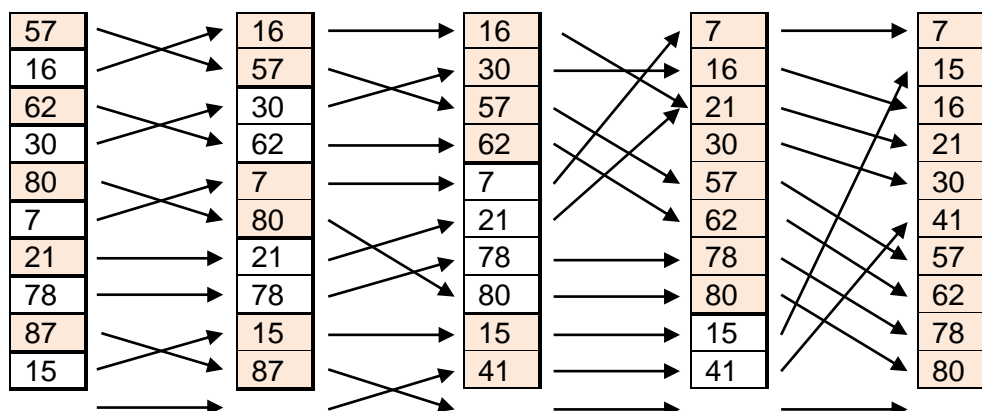
VL10, Lösung 3

Beim Versickern müssen an jedem Knoten zwei Vergleiche durchgeführt werden: einmal zwischen den beiden Kind-Knoten, und zwischen dem größten Kind-Knoten und dem zu versickernden Element.

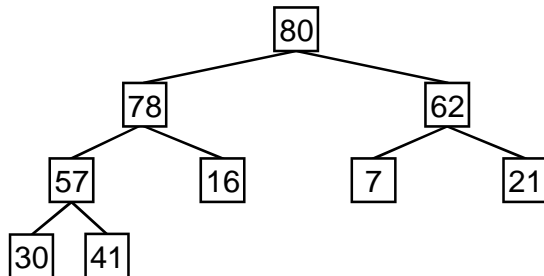
Sollen sehr viele Elemente sortiert werden und sind Vergleiche zudem teuer (also komplexe `compareTo`-Methode, die ggf. mehrere Attribute wie Strings berücksichtigt), so ist es günstiger zunächst den Pfad entlang der größten Elemente zum Blatt zu verfolgen (1 Vergleich je Knoten), und danach von unten entlang des Pfades die richtige Stelle zum Einfügen zu finden (ebenfalls 1 Vergleich je Knoten).

VL10, Lösung 4

Die unterschiedlichen Bereiche sind abwechselnd orange und weiß eingefärbt:



VL10, Lösung 5 a)

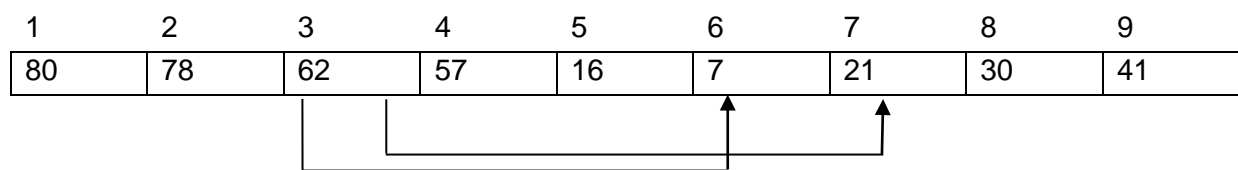


Array-Darstellung:

80	78	62	57	16	7	21	30	41
----	----	----	----	----	---	----	----	----

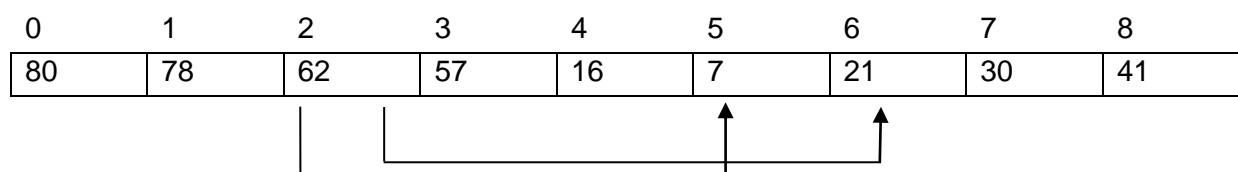
Im Pseudocode beginnen Felder mit dem Index 1. Die Formeln für die Position des linken und rechten Kinds eines Knotens mit dem Index i lauten entsprechend der Vorlesung:

- Linkes Kind: $2 \cdot i$
- Rechtes Kind: $2 \cdot i + 1$

Beispiel: $i = 3$ 

In Java beginnen Felder mit dem Index 0. Um die Formel anzupassen, kann man wie folgt vorgehen. Sei i der Java-Index des Knotens. Zu diesem muss **1** addiert werden, um den entsprechenden Index im Pseudocode zu erhalten. Dann kann die Formel für den Pseudocode angewendet werden. Abschließend muss dann noch **1** subtrahiert werden, um wieder einen Java-Index zu erhalten:

- Linkes Kind: $2 \cdot (i + 1) - 1 = 2 \cdot i + 1$
- Rechtes Kind: $2 \cdot (i + 1) + 1 - 1 = 2 \cdot i + 2$

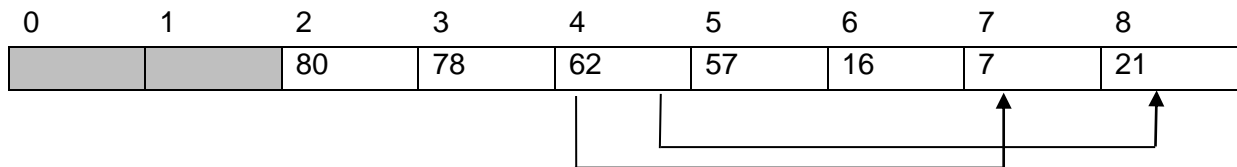
Beispiel: $i = 2$ 

Die Wurzel des Heaps befindet sich nun am Index links eines Java-Feldes. Um die Formel anzupassen, kann man wie folgt vorgehen. Sei i der Java-Index des Knotens. Von diesem muss **links** subtrahiert werden, um einen Java-Index beginnend ab 0 zu

erhalten. Dann kann die Formel für Java angewendet werden. Abschließend muss dann noch **links** addiert werden, um wieder einen passenden Index zu erhalten:

- Linkes Kind: $2 \cdot (i - \text{links}) + 1 + \text{links}$
- Rechtes Kind: $2 \cdot (i - \text{links}) + 2 + \text{links}$

Beispiel: $\text{links} = 2, i = 4$



VL10, Lösung 5 und 6

siehe Datei ML09-10.zip