FernUniversität in Hagen

Lösungsvorschläge zur Hauptklausur „1661 Datenstrukturen I“

4.8.2012

Aufgabe 1

(a)

1. $T_1(n) = 3n + 10 = O(n)$.
2. $T_2(n) = \log_2 7 \cdot n^2 + 16 = O(n^2)$.
3. $T_3(n) = kn^3 + 5 \cdot 2^n = O(2^n)$.
4. $T_4(n) = 0{,}1n \cdot \log n + 3n = O(n \cdot \log n)$.
5. $T_5(n) = n^2(4n + \log n) = O(n^3)$.
6. $T_6(n) = T_2(n) + T_3(n) = O(2^n)$.
7. $T_7(n) = T_1(n) + T_4(n) = O(n \cdot \log n)$.
8. $T_8(n) = T_3(n) + T_4(n) = O(2^n)$.
9. $T_9(n) = T_1(n) + n \cdot T_2(n) + n^2 \cdot T_4(n) = O(n^3 \cdot \log n)$.
10. $T_{10}(n) = (T_4(n))^2 \cdot T_2(n) = O(n^4 \cdot (\log n)^2)$.

(b)

1. Die Aussage ist wahr, denn $\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$. (L'Hospital)
2. Die Aussage ist falsch, denn $\lim_{n\to\infty} \frac{3^{n-1}}{2^n} = \lim_{n\to\infty} \frac{1}{3} \cdot \left(\frac{3}{2}\right)^n = \infty$.
3. Die Aussage ist falsch, denn mit $f(n) = 0$, $g(n) = n$ folgt $f(n) + g(n) = n \notin O(0) = O(f(g(n)))$.
4. Die Aussage ist wahr, denn $\lim_{n\to\infty} \frac{2^{n+1}}{2^n} = \lim_{n\to\infty} 2 \cdot \left(\frac{2}{2}\right)^n = 2$.

(c)

Die Aussage folgt aus der Transitivität der Relation ‚≤'. Es gilt nach Voraussetzung sowohl

$$\exists\, n_1 \in \mathbb{N}, c_1 \in \mathbb{R}^+ : \forall\, n \geq n_1 : f(n) \leq c_1 \cdot g(n)$$

als auch

$$\exists\, n_2 \in \mathbb{N}, c_2 \in \mathbb{R}^+ : \forall\, n \geq n_2 : g(n) \leq c_2 \cdot h(n).$$

Daraus folgt sofort

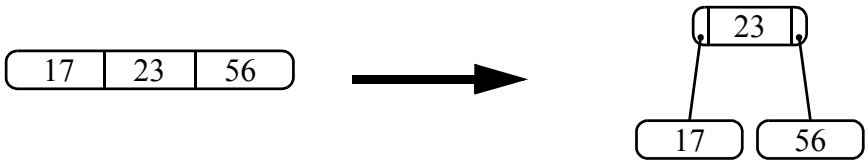
$$\exists\, n_3 \in \mathbb{N}, c_3 \in \mathbb{R}^+ : \forall\, n \geq n_3 : f(n) \leq c_3 \cdot h(n),$$

und damit $f \in O(h)$, wenn man $c_3 = c_1 \cdot c_2$ und $n_3 = \max(n_1, n_2)$ wählt.

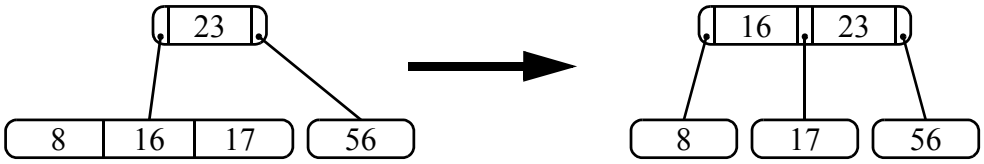
Aufgabe 2

(a)

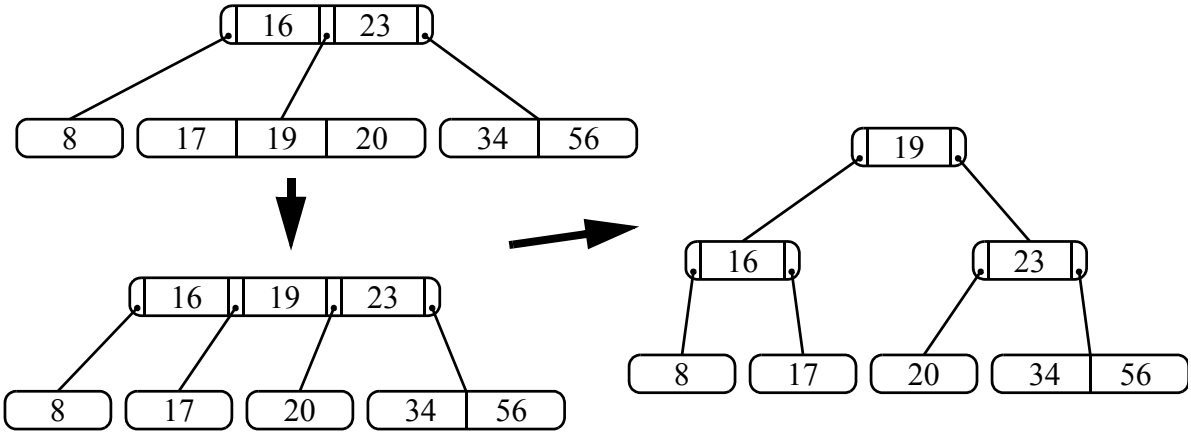
Der erste Überlauf tritt beim Einfügen der 56 ein:



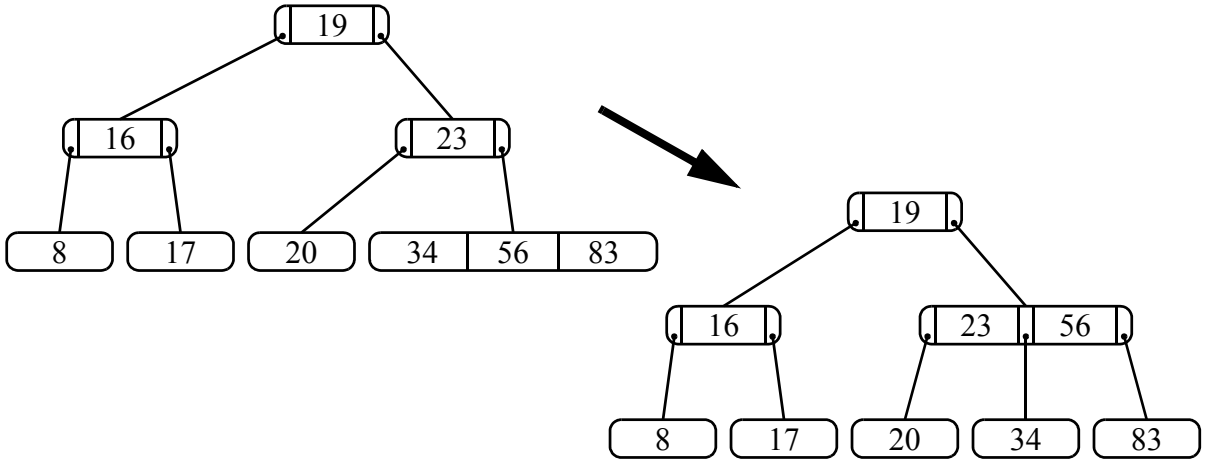
Beim Einfügen der 16 tritt der nächste Überlauf auf:



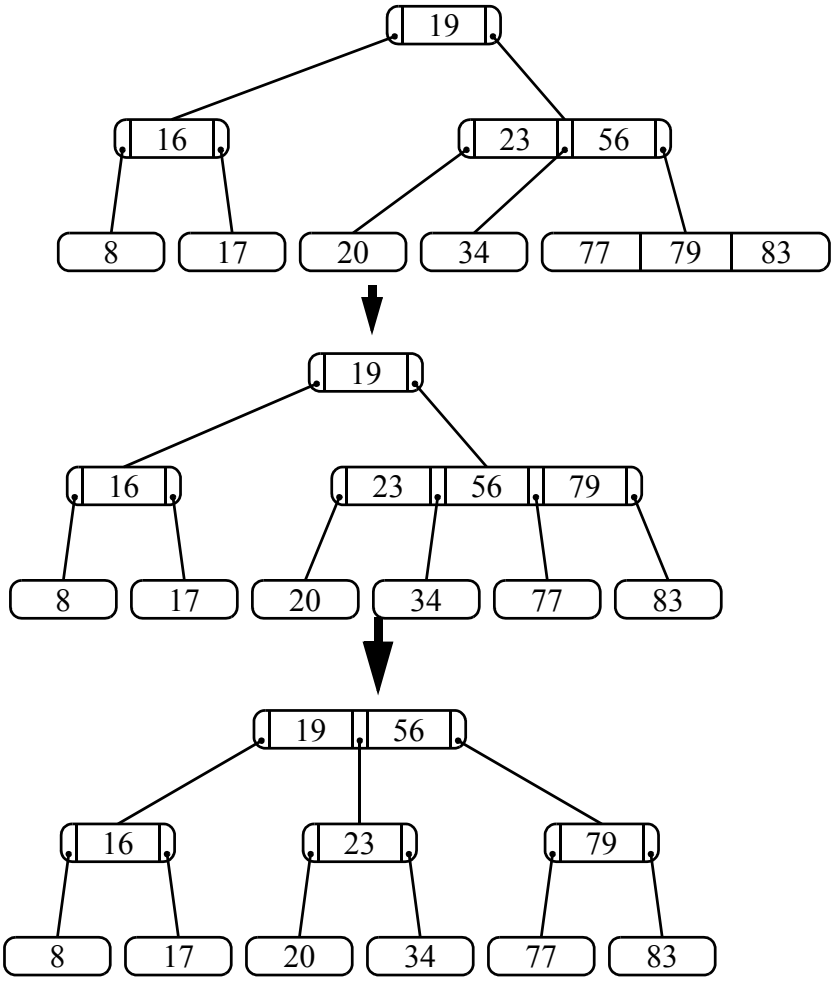
Beim Einfügen der 19 ist erneut eine Überlaufbehandlung notwendig. Durch den Split muss ein Überlauf im Vaterknoten behandelt werden:



Auch beim Einfügen der 83 tritt ein Überlauf ein:

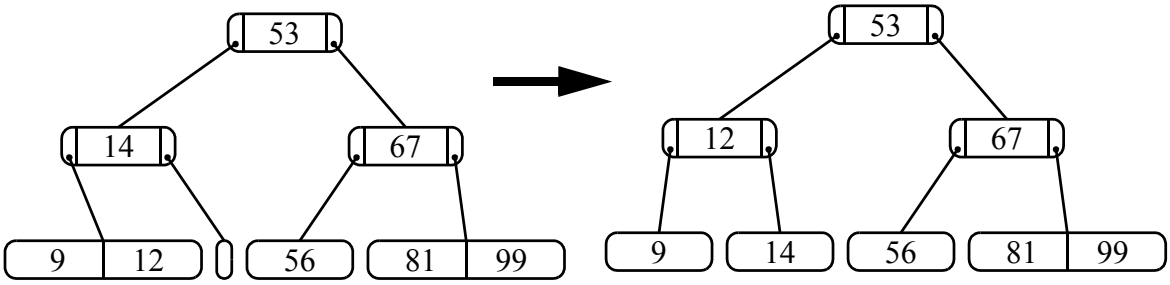


Beim Einfügen der 79 tritt erneut ein Überlauf ein, der sich bis zur nächsten Ebene fortsetzt:



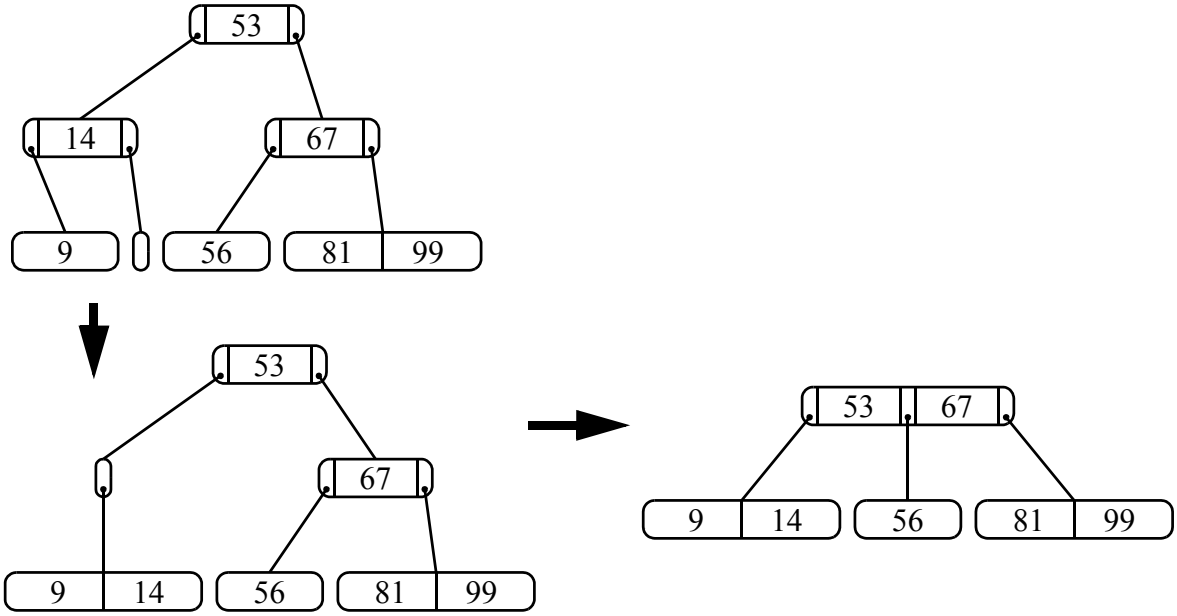
(b)

Beim Löschen der 18 gibt es einen Unterlauf, der mittels Balance behoben wird.

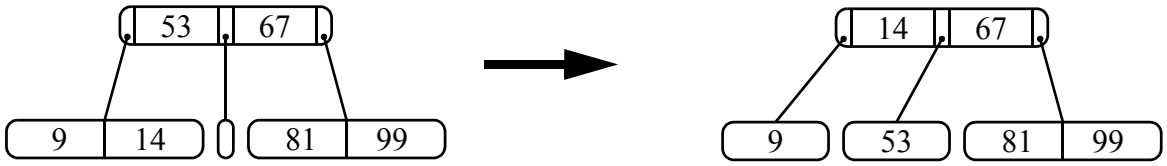


Beim Löschen der 12 wird diese zunächst mit der 14 getauscht. Die anschließende Löschung führt zu einem Unterlauf im entsprechenden Knoten. Mangels Nachbarn mit genügend vielen

Elementen wird ein Merge durchgeführt, der erneut zu einem Unterlauf führt. Auch dieser wird durch einen Merge ausgeglichen:



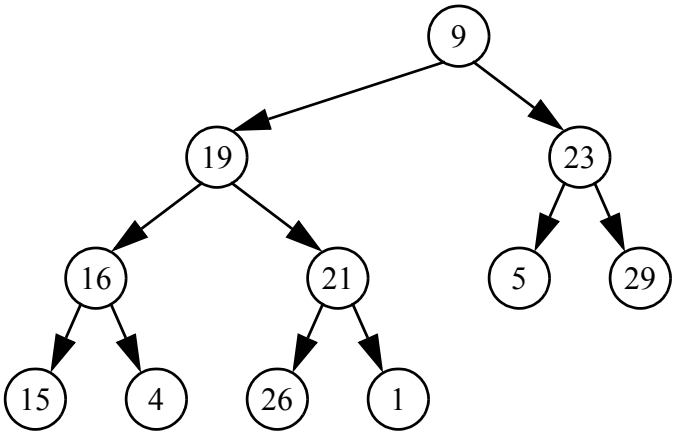
Der Unterlauf beim Löschen der 56 wird wieder mittels Merge ausgeglichen. Beide Nachbarn verfügen dazu über genügend Elemente, daher wird (wie in der Aufgabenstellung gefordert) mit dem linken Nachbarn ausgeglichen:



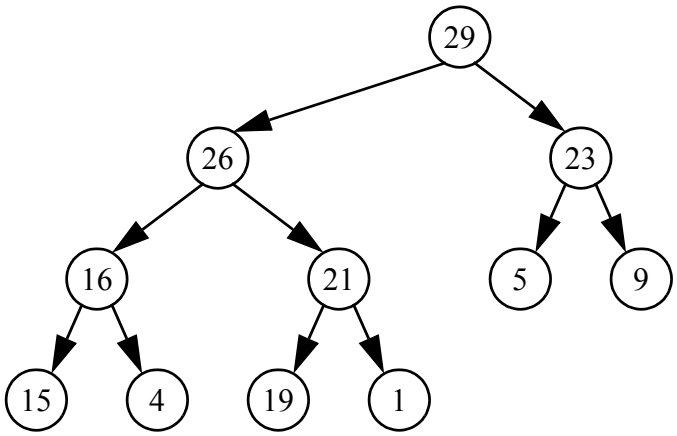
Aufgabe 3

(a)

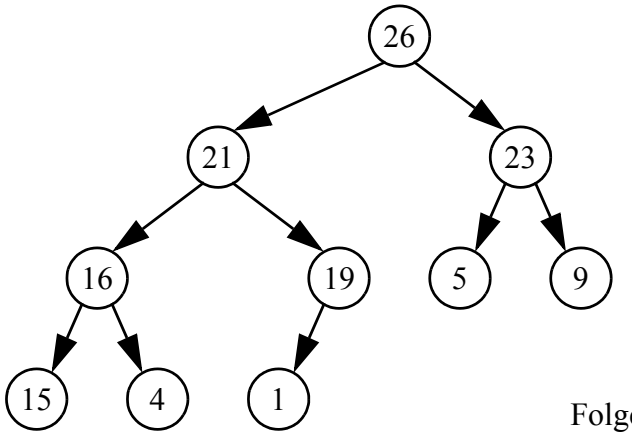
Die Baumeinbettung der Folge ist:



Nach der initialen Erstellung des Maximum-Heaps haben wir:

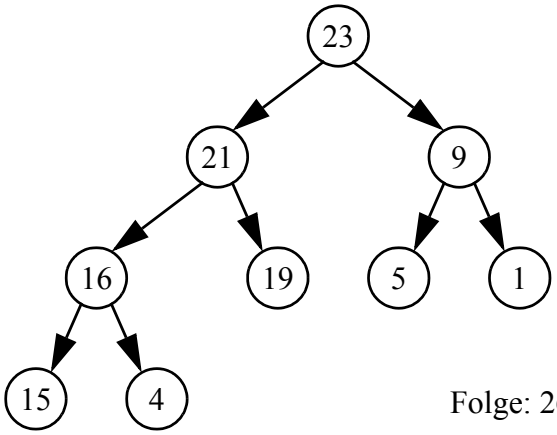


Nach Verarbeitung der 29 erhalten wir:



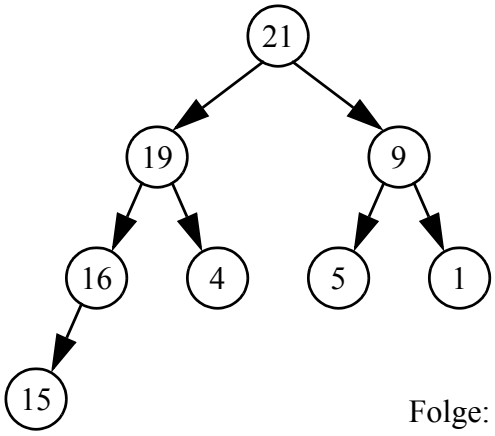
Folge: 29

Als nächstes wird die 26 zur Folge hinzugefügt, und die 23 sinkt ein:



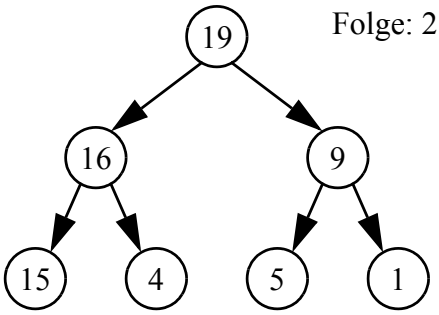
Folge: 26 29

Nachdem die 23 zur Folge hinzugefügt wurde, ergibt sich:



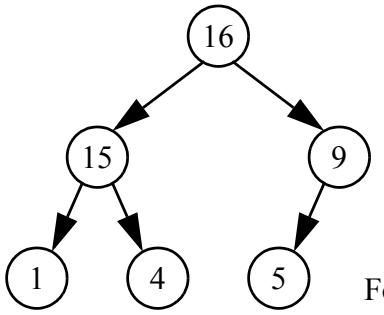
Folge: 23 26 29

Danach erhalten wir:



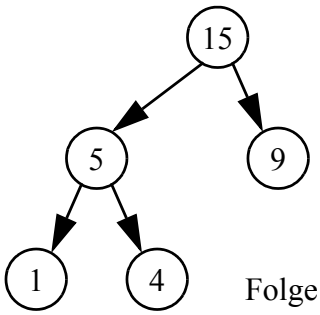
Folge: 21 23 26 29

Die 16 sinkt ein, und es ergibt sich:



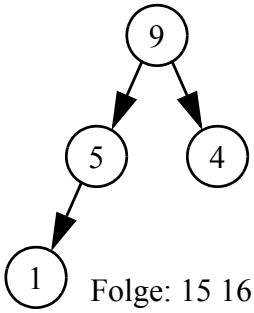
Folge: 19 21 23 26 29

Die 16 wird als nächste Zahl der sortierten Folge hinzugefügt:



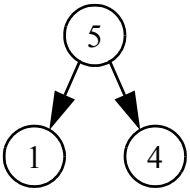
Folge: 16 19 21 23 26 29

Im nächsten Schritt erhalten wir:



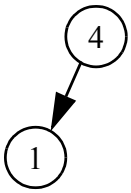
Folge: 15 16 19 21 23 26 29

Nach Verarbeitung der 9 haben wir folgendes Bild:



Folge: 9 15 16 19 21 23 26 29

Die 5 wird am Anfang der sortierten Folge eingefügt:



Folge: 5 9 15 16 19 21 23 26 29

Im nächsten Schritt terminiert das Verfahren:

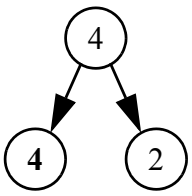
Folge: 4 5 9 15 16 19 21 23 26 29

(b)

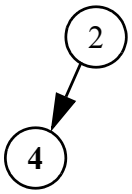
Ausgehend von der Wurzel durchlaufen wir den Heap nach unten und bestimmen dabei den Pfad minimaler bzw. maximaler (je nach Sortierrichtung) Kinder für das einsinkende Element *e*. Anschließend betrachten wir diesen Pfad in entgegengesetzter Richtung und stoppen, sobald ein Schlüssel $q < e$ gefunden wird. Wir entfernen nun *e*, schieben jedes Element des Pfades bis einschließlich *q* um eine Position nach oben und fügen *e* auf dem ehemaligen Platz von *q* ein.

(c)

Wir verwenden Heapsort, um die Folge 4 **4** 2 aufsteigend zu sortieren. Im ersten Schritt betten wir die Folge in einen Baum ein:



Die Maximum-Heap-Eigenschaft ist bereits erfüllt, also können wir das Maximum (4) entfernen und das letzte Blattelement (2) in die Wurzel eintragen:



Nun vertauschen wir die beiden Elemente, entfernen **4** aus dem Heap und setzen noch 2 an den Anfang der sortierten Folge, so dass diese schließlich 2 **4** 4 lautet

Die Vertauschungen beim Wiederherstellen der Heap-Eigenschaft sorgen also dafür, dass gleiche Elemente nach dem Sortieren in anderer Reihenfolge ausgegeben werden können, somit ist Heapsort instabil.

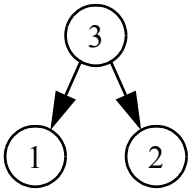
(d)

Die j-te Ebene eines Heaps enthält maximal 2^j Elemente. Ein Heap der Höhe h besitzt h voll besetzte Ebenen mit insgesamt $2^0 + 2^1 + \ldots + 2^{h-1} = 2^h - 1$ Elementen. Hinzu kommen mindestens ein Element (wenn die h-te Ebene nur aus einem Blatt besteht) und höchstens 2^h Elemente (wenn die h-te Ebene voll besetzt ist). Ein Heap der Höhe h enthält also mindestens 2^h und höchstens $2^h + 2^h - 1 = 2^{h+1} - 1$ Elemente.

(e)

Ein absteigend sortiertes Array repräsentiert stets einen gültigen Maximum-Heap, denn die Sortierung gewährleistet, dass alle Kindknoten kleiner oder gleich ihrem Vaterknoten sind, der ja im Array vor ihnen steht. Umgekehrt wird aber nicht jeder gültige Maximum-Heap durch ein

absteigend sortiertes Array dargestellt, da Elemente einer Heap-Ebene unsortiert sein können. So ist z.B.



ein gültiger Maximum-Heap mit der unsortierten Array-Darstellung [3, 1, 2].

Aufgabe 4

(a)

Für die Namen ergeben sich die folgenden Hashwerte:

Fabio: 6 + 1 + 2 = 9,
Hartmut: 8 + 1 + 18 = 27,
Anne: 1 + 14 + 14 = 29,
Wolfgang: 23 + 15 + 12 = 50,
Simone: 19 + 9 + 13 = 41,
Thomas: 20 + 8 + 15 = 43,
Sara: 19 + 1 + 18 = 38
Mahmoud: 13 + 1 + 8 = 22

(b)

Wir erhalten als finale Hashtabelle:

Behälternr.	Name	Behälternr.	Name
0	Wolfgang	5	
1	Simone	6	
2	Mahmoud	7	Hartmut
3	Thomas	8	Sara
4		9	Fabio, Anne

(c)

Für die Namen ergeben sich folgende Behälternummern:

Fabio	9		
Hartmut	7		
Anne	9 besetzt	0	
Wolfgang	0 besetzt	1	
Simone	1 besetzt	2	
Thomas	3		
Sara	8		
Mahmoud	2 besetzt	3 besetzt	6

Wir erhalten als finale Hashtabelle:

Behälternr.	Name	Behälternr.	Name	Behälternr.	Name
0	Anne	4		8	Sara
1	Wolfgang	5		9	Fabio
2	Simone	6	Mahmoud		
3	Thomas	7	Hartmut		

(d)

Die Hashtabelle bietet keinerlei Sortierung. Vor der Ausgabe müssen die Werte deshalb aus den Behältern der Hashtabelle entnommen und alphabetisch sortiert werden. Eine Möglichkeit ist das Einfügen der Werte in einen AVL-Baum gefolgt von der Ausgabe des inorder-Durchlaufs durch den AVL-Baum.

Statt des Einfügens in einen AVL-Baum und der anschließenden Ausgabe des inorder-Durchlaufs durch den AVL-Baum kann natürlich auch jedes andere „optimale“ Sortierverfahren mit entsprechender Ausgabemöglichkeit genutzt werden.

```
algorithm SortedOuput(Hashtable H)
begin
T : empty AVL-Tree
for each Bucket B of H do
  if (not(isempty(B)) then
     for each Name N of B do
        insert N into T
     od for
  fi
od for
output result of inorder(T)
end
```

(e)

Unabhängig von der Füllung müssen alle m Behälter der Hashtabelle untersucht werden. Für jeden Behälter müssen alle darin gespeicherten Werte in den AVL-Baum eingefügt werden. Abschließend müssen die im AVL-Baum gespeicherten Werte noch sortiert ausgegeben werden. Sei n die Gesamtanzahl der in der Hashtabelle gespeicherten Werte, dann beträgt der Aufwand für das Auslesen der Werte aus der Hashtabelle zusammen mit dem Einfügen der Werte in den AVL-Baum und der Ausgabe $O(m + n \log n + n) = O(m + n \log n)$.

(f)

Nein. Wenn eine Menge öfter sortiert ausgegeben werden muss, sollte für die Speicherung eine Datenstruktur verwendet werden, die die Menge direkt sortiert ablegt und dementsprechend die Menge ohne Zusatzaufwand sortiert ausgeben kann. Dies ist beim Hashing nicht der Fall. Die in der Hashtabelle abgespeicherte Menge muss für die sortierte Ausgabe jedesmal erst aufwändig sortiert werden. Außerdem müssen für die Sortierung immer alle Behälter der Hashtabelle durchlaufen werden, auch wenn nur wenige davon benötigt werden.

Aufgabe 5 Deckblatt

Hier erhalten Sie den Punkt, wenn Sie beide Klausurdeckblätter korrekt und vollständig ausgefüllt haben, also Namen, Matrikelnummer und Adresse korrekt eingetragen und genau diejenigen Aufgaben markiert haben, die Sie auch bearbeitet haben.