Prof. Dr. Ir. Joost-Pieter Katoen

Sebastian Junges, Benjamin Kaminski, David Korzeniewski, Tim Quatmann

Präsenzübung Datenstrukturen und Algorithmen SS 2018

– Musterlösung –

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	**15**	
Aufgabe 2	**15**	
Aufgabe 3	**15**	
Aufgabe 4	**18**	
Aufgabe 5	**12**	
Aufgabe 6	**15**	
Summe	**90**	

Hinweise:

- Auf **alle Blätter** (inklusive Zusatzblätter) müssen Sie **Ihre Matrikelnummer** schreiben.
- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- Schreiben Sie mit **dokumentenechten** Stiften, nicht mit roten oder grünen Stiften und nicht mit Bleistiften.
- Bitte beantworten Sie die Aufgaben auf den Aufgabenblättern.
- Geben Sie für jede Aufgabe **maximal eine** Lösung an. Streichen Sie alles andere durch. Andernfalls werden alle Lösungen der Aufgabe mit **0 Punkten** bewertet.
- Werden **Täuschungsversuche** beobachtet, so wird die Übung mit **0 Punkten** bewertet.
- Geben Sie am Ende der Übung **alle Blätter zusammen mit den Aufgabenblättern ab**.

Aufgabe 1 ($\mathcal{O}$-Notation): (3 + 7 + 5 = 15 Punkte)

Rufen Sie sich die Quasiordnung $\sqsubseteq$, definiert als

$$f \sqsubseteq g \quad \text{genau dann wenn} \quad f \in \mathcal{O}(g) \ ,$$

in Erinnerung.

a) Beweisen oder widerlegen Sie, dass $\sqsubseteq$ anti-symmetrisch ist.

b) Sortieren Sie die folgenden 15 Funktionen

$$\log(n^n), \quad \sum_{i=0}^{n} \frac{2i^3}{1+i}, \quad 2^n, \quad n^2, \quad 3^n, \quad 2^{9000 \cdot n}, \quad n^2 \cdot \log(n), \quad n^n,$$

$$n \cdot \sqrt[3]{n}, \quad n \cdot \log(n), \quad n!, \quad \log(n^2), \quad 0, \quad n \cdot \sqrt{n}, \quad \frac{n^2}{2000}$$

in aufsteigender Reihenfolge bezüglich der Quasiordnung $\sqsubseteq$. Zeigen Sie auch durch $f \approx g$ an, wenn sowohl $f \sqsubseteq g$ als auch $g \sqsubseteq f$ gilt. Schreiben Sie also beispielsweise

$$f \sqsubseteq g \approx h \sqsubseteq i \qquad \text{falls} \qquad f \in \mathcal{O}(g) \ , \quad g \in \mathcal{O}(h) \text{ und } h \in \mathcal{O}(g) \ , \quad \text{und} \quad h \in \mathcal{O}(i) \ .$$

c) Beweisen oder widerlegen Sie: $o(f) \cap \Theta(f) = \emptyset$.

Lösung:

a) $\sqsubseteq$ ist nicht anti-symmetrisch. Wähle als Gegenbeispiel $f(n) = 1$ und $g(n) = 2$. Dann gilt wegen

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{1}{2} < \infty \qquad \text{und} \qquad \lim_{n \to \infty} \frac{g(n)}{f(n)} = 2 < \infty$$

nach alternativer Definition von $\mathcal{O}$ sowohl

$$f \in \mathcal{O}(g) \qquad \text{als auch} \qquad g \in \mathcal{O}(f) \ .$$

Damit gilt nach Definition von $\sqsubseteq$ auch sowohl

$$f \sqsubseteq g \qquad \text{als auch} \qquad g \sqsubseteq f \ ,$$

aber *nicht*

$$f = g,$$

und damit ist Anti-symmetrie von $\sqsubseteq$ widerlegt.

b)

$$0 \sqsubseteq \log(n^2) \sqsubseteq \log(n^n) \approx n \cdot \log(n) \sqsubseteq n \cdot \sqrt[3]{n} \sqsubseteq n \cdot \sqrt{n} \sqsubseteq n^2 \approx \frac{n^2}{2000}$$

$$\sqsubseteq n^2 \cdot \log(n) \sqsubseteq \sum_{i=0}^{n} \frac{2i^3}{1+i} \sqsubseteq 2^n \sqsubseteq 3^n \sqsubseteq 2^{9000 \cdot n} \sqsubseteq n! \sqsubseteq n^n$$

c) Wir beweisen $o(f) \cap \Theta(f) = \emptyset$. Wir wissen aus der Vorlesung, dass $g \in \Theta(f)$ gilt, genau dann wenn

$$\exists c_1, c_2 > 0 \; \exists n_0 \; \forall n \geq n_0 : \quad c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n) \; .$$

Außerdem gilt $g \in o(f)$ gilt, genau dann wenn

$$\forall c > 0 \; \exists n_0 \; \forall n \geq n_0 : \quad 0 \leq g(n) < c \cdot f(n) \; .$$

Nehmen wir nun an, dass $g \in \Theta(f)$ und $g \in o(f)$ gilt. Dann müsste wegen $g \in \Theta(f)$ gelten:

$$\begin{aligned} & \exists c_1, c_2 > 0 \; \exists n_0 \; \forall n \geq n_0 : \quad c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n) \\ \Longrightarrow \; & \exists c_1 > 0 \; \exists n_0 \; \forall n \geq n_0 : \quad c_1 \cdot f(n) \leq g(n) \end{aligned}$$

Nun gilt die Aussage

$$0 \leq g(n) < c \cdot f(n) \; .$$

in der Definition von $g \in o(f)$ insbesondere für $c = c_1$, da sie ja für alle c gelten muss. Damit ergibt sich folgender Widerspruch: Es müsste nämlich gelten

$$\begin{aligned} & \exists c_1 > 0 \; \exists n_0 \; \forall n \geq n_0 : \quad c_1 \cdot f(n) \leq g(n) < c_1 \cdot f(n) \\ \Longrightarrow \; & \exists c_1 > 0 \; \exists n_0 \; \forall n \geq n_0 : \quad c_1 \cdot f(n) < c_1 \cdot f(n) \\ \Longrightarrow \; & \exists n_0 \; \forall n \geq n_0 : \quad f(n) < f(n) \qquad \text{(Widerspruch!)} \end{aligned}$$

Aufgabe 2 (Sortieren): (4 + 5 + 6 = 15 Punkte)

a) Sortieren Sie das folgende Array mithilfe von Mergesort. Geben Sie dazu das Array nach jeder Merge-Operation an. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

5	23	7	2	8	17	9	3

b) Quicksort ist im allgemeinen einer der effizientesten Sortieralgorithmen. Leider benötigt er aber quadratische Laufzeit, wenn immer das größte oder kleinste Element als Pivot ausgewählt wird. Wie kann Quicksort so angepasst werden, dass der Algorithmus auf fast sortierten Eingaben dennoch effizient ist? Begründen Sie ihre Antwort.

Wir betrachten eine Eingabe als fast sortiert, wenn nur sehr wenige Elemente an zufälliger Position nicht korrekt einsortiert sind.

c) Ein naiver Algorithmus um das häufigste Element in einem Array der Länge $n \geq 1$ zu finden, benötigt $\Theta(n^2)$ Zeit und konstant viele Array-Indizes als Speicherplatz. Der naive Algorithmus zählt für jedes Element im Array, wie häufig es vorkommt und speichert sich jeweils die Position des bisher häufigsten Elements und seine Häufigkeit.

Beschreiben Sie einen Algorithmus, der das häufigste Element in einem Array der Länge $n > 1$ findet und dessen asymptotische Laufzeit in $O(n \log n)$ liegt. Außerdem darf der Algorithmus nur konstant mehr Speicher benötigen als ein naiver Algorithmus. Begründen Sie ihre Antwort!

Lösung:

a)

5	23	7	2	8	17	9	3
5	23	7	2	8	17	9	3
5	23	2	7	8	17	9	3
2	5	7	23	8	17	9	3
2	5	7	23	8	17	9	3
2	5	7	23	8	17	3	9
2	5	7	23	3	8	9	17
2	3	5	7	8	9	17	23

b) Anstatt das letzte Element im Array als Pivot zu nutzen, wird das mittlere Element als Pivot genutzt. Da das Array fast sortiert ist, wird es bei Auswahl des mittleren Elements als Pivot wahrscheinlich in zwei fast gleich große Teile aufgeteilt. Die Höhe des Rekursionsbaums ist somit wahrscheinlich minimal. Damit erreicht Quicksort die Laufzeit $O(n \log n)$.

Realisiert werden kann diese Anpassung, indem man in jedem Schritt zunächst das mittlere Element mit dem letzten tauscht und anschließend wie in der Vorlesung vorgeht.

c) Sortiere das Array zunächst mit Heapsort. Dieser Algoritmus hat laut Vorlesung eine Worst Case Laufzeit in $O(n \log n)$ und konstanten Speicherbedarf.

In einem sortierten Array kann das häufigste Element in Linearzeit gefunden werden, da gleiche Elemente hintereinander stehen. Im Pseudocode sieht der Algorithmus wie folgt aus:

```
input: arr

arr := heapsort(arr)

h_anzahl := −∞
haeufigstes := 0

anzahl := 0

for aktuell := 1 ... n do
  anzahl := anzahl + 1
  if arr[aktuell] < arr[aktuell + 1] and aktuell < n then
    if anzahl >= h_anzahl then
      h_anzahl := anzahl
      haeufigstes := aktuell
    anzahl := 0

output: arr[haeufigstes]
```

Aufgabe 3 (Sortieren): (4 + 5 + 6 = 15 Punkte)

a) Sortieren Sie das folgende Array mit Hilfe von Insertionsort. Geben Sie dazu das Array nach jeder Iteration der äußeren Schleife an. Gehen Sie strikt nach dem Verfahren aus der Vorlesung vor. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

5	4	0	8	2	4	7

b) Geben Sie ein Array der Länge 4 an, sodass der Quicksort Algorithmus aus der Vorlesung diese Eingabe nicht stabil sortiert.

c) Es sei `sortHeap(int E[])` ein vergleichsbasierter Sortieralgorithmus, der als Eingabe einen Max-Heap (repräsentiert als Array `E`) bekommt und daraufhin das Eingabearray aufsteigend sortiert.

Bestimmen Sie eine asymptotische Worst-Case Laufzeit $f(n)$ in Abhängigkeit der Eingabelänge $n = \texttt{E.length}$, die ein solcher Algorithmus haben kann, sodass es

(i) **eine** Implementierung von `sortHeap(int E[])` gibt, dessen Worst-Case Laufzeit in $\mathcal{O}(f(n))$ liegt und

(ii) **keine** Implementierung von `sortHeap(int E[])` gibt, dessen Worst-Case Laufzeit in $o(f(n))$ liegt.

Begründen Sie informell, warum (i) und (ii) für Ihre angegebene Funktion gelten.

$f(n) =$ _____

Begründung zu (i):

Begründung zu (ii):

Lösung:

a) Lösung zu Insertionsort:

5	4	0	8	2	4	7
4	5	0	8	2	4	7
0	4	5	8	2	4	7
0	4	5	8	2	4	7
0	2	4	5	8	4	7
0	2	4	4	5	8	7
0	2	4	4	5	7	8

b) Mögliche Lösungen: $[1, 2, 3, 1]$, ...

c) $f(n) = n \cdot \log n$

(i) `sortHeap` kann einen Sortieralgorithmus für beliebige Arrays mit Worst-Case Laufzeit $n \cdot \log n$ aufrufen, z.B. Mergesort oder Heapsort.

(ii) Laut Vorlesung lässt sich in $\mathcal{O}(n)$ ein beliebiges Array in einen Heap überführen. Falls es eine Implementierung von `sortHeap` gäbe, die eine Worst-Case Laufzeit $g(n) \in o(n \cdot \log n)$ hat, könnte man also beliebige Arrays schneller als $\mathcal{O}(n \cdot \log n)$ (vergleichsbasiert) sortieren. Dies ist laut Vorlesung nicht möglich.

Aufgabe 4 (Rekursionsgleichungen): (4 + 4 + 5 + 5 = 18 Punkte)

Gegeben sei die Rekursionsgleichung

$$T(1) = 1\,,$$
$$T(n) = 2T\left(\frac{n}{3}\right) + n^2\,.$$

a) Zeichnen Sie einen Rekursionsbaum für die Rekursionsgleichung T. Dieser soll mindestens die obersten 3 Ebenen enthalten, sowie eine Ebene für die Blätter.

b) Stellen Sie anhand des Rekursionsbaumes eine Summenformel für die sich ergebenden Gesamtkosten auf. Bestimmen Sie dazu die asymptotische Tiefe des Baumes, die asymptotische Breite der Ebene der Blätter, die akkumulierten Kosten einer beliebigen Ebene i, die nicht der Blätterebene entspricht, und letztendlich die sich ergebenden Gesamtkosten.

c) Bestimmen Sie aus der obigen Summenformel eine geschlossene Form für die asymptotischen Gesamtkosten. Zeigen Sie mit Hilfe einer beliebigen, Ihnen aus Vorlesung oder Übung bekannten, Methode, dass Ihre gefundene Schranke tatsächlich eine obere Schranke für eine Lösung der Rekursionsgleichung ist.

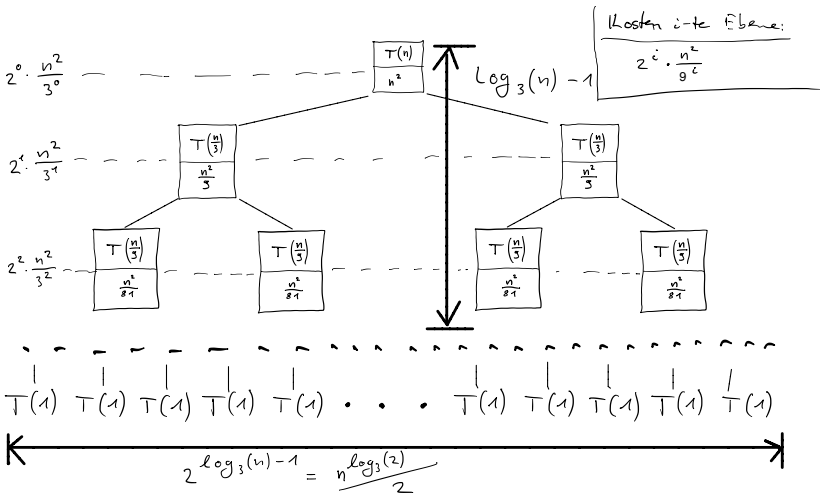
d) Lösen Sie die folgende Rekursionsgleichung

$$T(1) = 1\,,$$
$$T(n) = 3T\left(\frac{n}{2}\right) + n^2\,.$$

mit Hilfe des Mastertheorems.

Lösung:

a)



Gesamtkosten:

$$\sum_{i=0}^{\log_3(n)-1}\left(2^i \cdot \frac{n^2}{9^i}\right) + n^{\log_3(2)} = n^2 \cdot \sum_{i=0}^{\log_3(n)-1}\left(\frac{2}{9}\right)^i + n^{\log_3(2)}$$

$\left(\frac{2}{9}\right)^i < 1$

$$\longrightarrow 7/9\,n^2 + n^{\log_3(2)} \in O(n^2)$$

b) Siehe a)

c) Wir benutzen das Mastertheorem. Es sind $b = 2$, $c = 3$ und $f(n) = n^2$. E wird nun wie folgt bestimmt:

$$E = \frac{\log(2)}{\log(3)} = \log_3(2) < 1$$

Damit gilt $n^E = n^{\log_3(2)}$. Wähle nun $\varepsilon = 2 - E > 0$. Wir bestimmen nun den folgenden Grenzwert:

$$\begin{aligned} & \lim_{n\to\infty} \frac{f(n)}{n^{E+\varepsilon}} \\ = & \lim_{n\to\infty} \frac{n^2}{n^{E+2-E}} \\ = & \lim_{n\to\infty} \frac{n^2}{n^2} \\ = & \lim_{n\to\infty} 1 \\ = & \ 1 > 0 \end{aligned}$$

Damit gilt $f(n) \in \Omega\left(n^{E+\varepsilon}\right)$. Es würde somit der dritte Fall des Mastertheorems Anwendung finden. Dazu müssen wir noch

$$bf\left(\frac{n}{c}\right) \leq d \cdot f(n)$$

für ein $d < 1$ und ab einem hinreichend großen n überprüfen:

$$2 \cdot f\left(\frac{n}{3}\right) = 2\left(\frac{n}{3}\right)^2 = \frac{2}{9}n^2 \leq df(n) \ ,$$

für $d = \frac{2}{9}$. Damit gelten alle Vorbedingungen für den dritten Fall des Mastertheorems und es ergibt sich die Komplexitätsklasse $\Theta(f(n))$ für $T(n)$, also

$$T(n) \in \Theta\left(n^2\right) \ .$$

d) Es sind $b = 3$, $c = 2$ und $f(n) = n^2$. E wird nun wie folgt bestimmt:

$$E = \frac{\log(3)}{\log(2)} = \log_2(3) < 2$$

Damit gilt $n^E = n^{\log_2(3)}$. Wähle nun $\varepsilon = 2 - E > 0$. Wir bestimmen nun den folgenden Grenzwert:

$$\begin{aligned} & \lim_{n\to\infty} \frac{f(n)}{n^{E+\varepsilon}} \\ = & \lim_{n\to\infty} \frac{n^2}{n^{E+2-E}} \\ = & \lim_{n\to\infty} \frac{n^2}{n^2} \\ = & \lim_{n\to\infty} 1 \\ = & \ 1 > 0 \end{aligned}$$

Damit gilt $f(n) \in \Omega\left(n^{E+\varepsilon}\right)$. Es würde somit der dritte Fall des Mastertheorems Anwendung finden. Dazu müssen wir noch

$$bf\left(\frac{n}{c}\right) \leq d \cdot f(n)$$

für ein $d < 1$ und ab einem hinreichend großen n überprüfen:

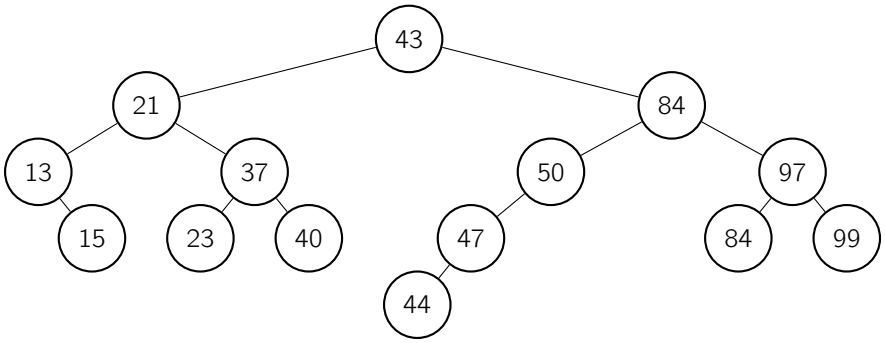
$$3 \cdot f\left(\frac{n}{2}\right) = 3\left(\frac{n}{2}\right)^2 = \frac{3}{4}n^2 \leq df(n) \ ,$$

für $d = \frac{3}{4}$. Damit gelten alle Vorbedingungen für den dritten Fall des Mastertheorems und es ergibt sich die Komplexitätsklasse $\Theta(f(n))$ für $T(n)$, also

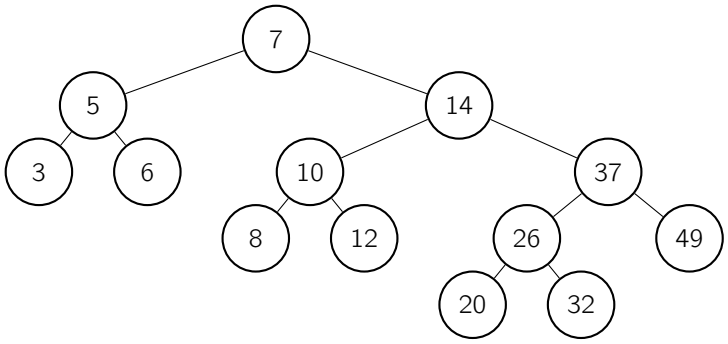
$$T(n) \in \Theta\left(n^2\right) \ .$$

Aufgabe 5 (Suchbäume): (2 + 3 + 4 + 3 = 12 Punkte)

a) Fügen Sie den Knoten mit Schlüssel 20 in den folgenden Binären Suchbaum (BST) ein. Nutzen Sie dazu das entsprechende Verfahren aus der Vorlesung.



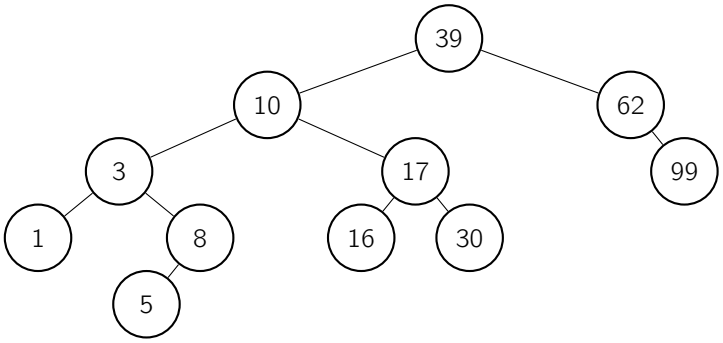
b) Löschen Sie den Knoten mit Schlüssel 14 aus dem folgenden Binären Suchbaum (BST). Nutzen Sie dazu das entsprechende Verfahren aus der Vorlesung.



c) Erstellen Sie einen AVL-Baum, der die folgenden Schlüssel enthält.

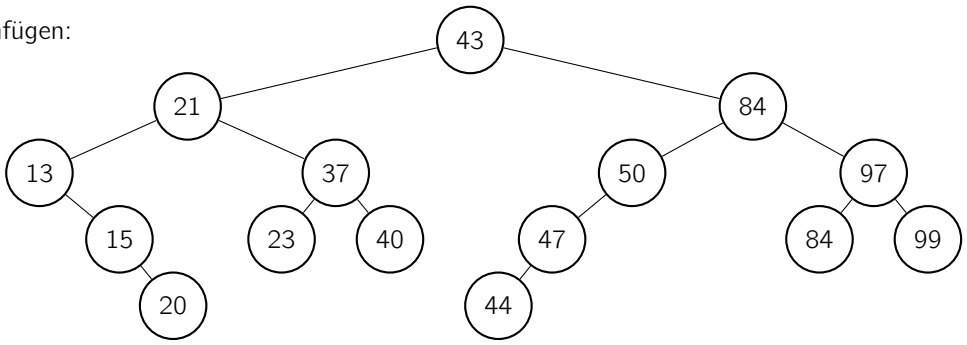
37	72	90	61	6	18	64	5

d) Führen Sie genau eine Rotation aus, um den folgenden BST in einen (balancierten) AVL-Baum zu überführen. Geben Sie den resultierenden AVL-Baum an sowie den Knoten auf dem die Rotation ausgeführt wurde und ob es sich um eine Links- oder Rechtsrotation handelt.

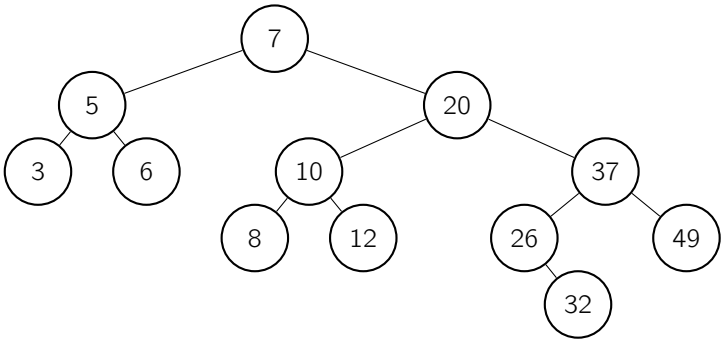


Lösung:

a) Knoten 20 einfügen:



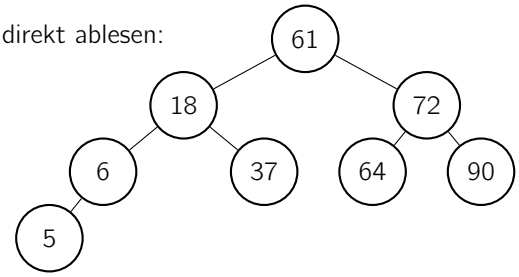
b) Knoten 14 löschen:



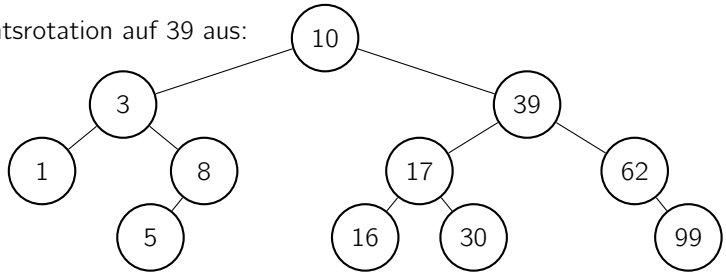
c) Zur Einfachheit sortieren wir zunächst die Schlüssel:

5	6	18	37	61	64	72	90

Ein AVL-Baum lässt sich nun direkt ablesen:



d) Wir führen eine Rechtsrotation auf 39 aus:



Aufgabe 6 (Laufzeit-Analyse): (3 + 3 + 3 + 3 + 3 = 15 Punkte)

Bestimmen Sie von folgenden Funktionen die Worst-Case Laufzeit. Geben Sie dazu eine geschlossene form für die asymptotische Anzahl von Print-Aufrufen an.

a)

```
A(n): Integer -> void

for i in 1..n:
    for j in 1..n:
        for k in 1..n:
            if i = k & j > i:
                print(..)
```

b)

```
B(n): Integer -> void

m = n
c = 0
while m > 1:
    m = m/2
    c = c + 1
    print(..)
while c > 0:
    c = c - 1
    for i in 1..n:
        print(..)
```

c)

```
C(n): Integer -> void

for i in 1..n:
    for j in 1..10:
        h = j*j
        for k in 1..h^j:
            print(..)
```

d)

```
D(n): Integer -> void

for i in 1..n:
    print(..)
    for j in 1..n:
        if i < 25:
            if j < i:
                print(..)
```

e) Hinweis: In dieser Teilaufgabe brauchen Sie keine geschlossene Form anzugeben. Eine Summe reicht!

```
PrimTest(n): Integer -> void

for i in 1..n:
    if i*i < n:
        t = n
        while t > 0:
            t = t - i
            print(Vielleicht)
        if t = 0:
            print(Ja)
        else:
            print(Nein)
```

Lösung:

a) $\Theta(n^2)$

b) $\Theta(n \cdot \log n)$

c) $\Theta(n)$

d) $\Theta(n)$

e) $\Theta(n \cdot \sum_{i=1}^{\sqrt{n}} \frac{1}{i})$