Tutorium Grundlagen: Algorithmen und Datenstrukturen

Übungsblatt Woche 10

Aufgabe 10.1 – Multiple Choice

Kreuzen Sie in den folgenden Teilaufgaben jeweils die richtigen Antworten an. Es können pro Teilaufgabe keine, einige oder alle Antworten richtig sein.

a) Wir erweitern den Bubblesort-Algorithmus, indem vor **jeder** Vergleichsoperation eine Funktion `isSorted` aufgerufen wird. Diese überprüft, ob das gesamte Feld bereits sortiert ist, indem jedes Paar von benachbarten Elementen überprüft wird. Dazu wird das gesamte zu sortierende Feld einmal komplett durchlaufen. Ist das Feld sortiert, wird das modifizierte Sortierverfahren sofort beendet.

Es bezeichne f die **Worst-Case**-Laufzeit des Original-Bubblesort-Algorithmus und g die des modifizierten Bubblesort-Algorithmus. Was gilt?

- ☐ $f \in \Theta(g)$
- ☒ $f \notin \Theta(g)$, aber $f \in \mathcal{O}(g)$
- ☐ $f \notin \Theta(g)$ und $g \in \mathcal{O}(f)$
- ☐ weder $g \in \mathcal{O}(f)$ noch $\underline{f \in \mathcal{O}(g)}$

$O(n \cdot n^2) = O(n^3)$

$O(n^2)$

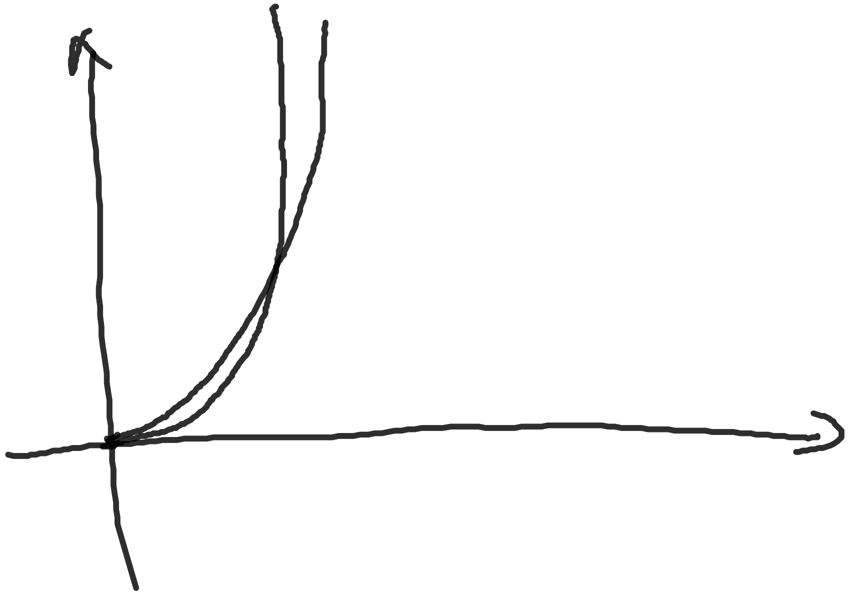
$\lim_{n \to \infty} \frac{\cancel{n^2}}{n^{\cancel{3}}} = \lim_{n \to \infty} \frac{1}{n} = 0$

$f \in o(g)$

$g \in \omega(f)$

Aufgabe 10.1 – Multiple Choice

- ☐ $f \in \Theta(g)$
- ☒ $f \notin \Theta(g)$, aber $f \in \mathcal{O}(g)$
- ☐ $f \notin \Theta(g)$ und $g \in \mathcal{O}(f)$
- ☐ weder $g \in \mathcal{O}(f)$ noch $f \in \mathcal{O}(g)$



$f \in O(g) \quad \exists c>0 \; \exists n_0>0 \; \forall n>n_0 \; c \cdot f(n) < g(n)$

Aufgabe 10.1 – Multiple Choice

b) Kreuzen Sie in den Zeilen (1) bis (3) jeweils das stärkste passende Symbol an. D. h. wenn z.B. $\Delta = o$ (bzw. $\Delta = \Theta$) möglich ist, wählen Sie $\Delta = o$ (bzw. $\Delta = \Theta$) und **nicht** $\Delta = \mathcal{O}$. Falls die Funktionen unvergleichbar sind, kreuzen Sie u. („unvergleichbar") an. Setzen Sie also in jeder Zeile genau ein Kreuz!

Bsp:	$n \in \Delta(n^2)$	☒ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ $u.$
(1)	$7\log_2(n) \in \Delta(4\log_2(n^2))$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☒ Θ	☐ $u.$
(2)	$n^3 \in \Delta(n^8(n \bmod 2))$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☒ $u.$
(3)	$4^n \in \Delta(2^{4n})$	☒ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ $u.$

Hinweis: Der Operator mod berechnet den Divisions-Rest.

(1) $7\log_2(n)$ $\quad 4 \cdot \log_2(n^2) = 2 \cdot 4 \cdot \log_2(n) = 8 \cdot \log_2(n)$

(2) $n^8 \underbrace{(n \bmod 2)}$

(3) $4^n = (2^2)^n \quad 2^{4n} = (2^4)^n = 16^n \quad \lim_{n \to \infty} \frac{4^n}{16^n} = 0$

Aufgabe 10.1 – Multiple Choice

(1)	$7\log_2(n) \in \Delta(4\log_2(n^2))$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☒ Θ	☐ $u.$
(2)	$n^3 \in \Delta(n^8(n \bmod 2))$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☒ $u.$
(3)	$4^n \in \Delta(2^{4n})$	☒ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ $u.$

Aufgabe 10.1 – Multiple Choice

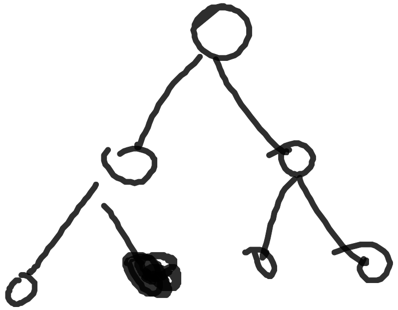
Aufgabe 10.1 – Multiple Choice

c) Welche Aussagen sind wahr?

- [x] Jeder AVL-Baum ist zugleich ein binärer Suchbaum.
- [ ] Jeder binäre Suchbaum ist zugleich ein Binärer Heap.
- [ ] Jeder Binäre Heap ist zugleich ein AVL-Baum.
- [ ] Jeder Binäre Heap ist zugleich ein binärer Suchbaum.

d) Welche Operation muss in einem Binären Min-Heap beim Ausführen von decreaseKey (Verringern eines Schlüssel-Wertes) u.U. aufgerufen werden, um die Heap-Invariante wiederherzustellen?

- [x] `siftUp`
- [ ] `siftDown`
- [ ] `deleteMin`
- [ ] `rotateLeft`



Aufgabe 10.1 – Multiple Choice

c) Welche Aussagen sind wahr?

- ☒ Jeder AVL-Baum ist zugleich ein binärer Suchbaum.
- ☐ Jeder binäre Suchbaum ist zugleich ein Binärer Heap.
- ☐ Jeder Binäre Heap ist zugleich ein AVL-Baum.
- ☐ Jeder Binäre Heap ist zugleich ein binärer Suchbaum.

d) Welche Operation muss in einem Binären Min-Heap beim Ausführen von decreaseKey (Verringern eines Schlüssel-Wertes) u.U. aufgerufen werden, um die Heap-Invariante wiederherzustellen?

☒ `siftUp` ☐ `siftDown` ☐ `deleteMin` ☐ `rotateLeft`

Aufgabe 10.1 – Multiple Choice

e) Die durchschnittliche Laufzeit von Algorithmen (*Average Case*) ist

☐ uninteressant, da man ausschließlich am *Worst-Case* interessiert ist.

☒ oft nur mit großem Aufwand zu berechnen.

☐ stets am besten geeignet, um den passenden Algorithmus zu wählen.

f) Der MergeSort-Algorithmus

☐ ist für alle Eingaben schneller als jede gute Implementierung von InsertionSort.

☒ ist für bestimmte Eingabeklassen signifikant schneller als eine deterministische Implementierung von Quicksort.

☐ ist im Schnitt um einen in der Eingabe linearen Faktor schneller als Quicksort.

☐ sortiert eine Eingabe im *Best Case* in linearer Zeit.

Aufgabe 10.1 – Multiple Choice

e) Die durchschnittliche Laufzeit von Algorithmen (*Average Case*) ist

☐ uninteressant, da man ausschließlich am *Worst-Case* interessiert ist.

☒ oft nur mit großem Aufwand zu berechnen.

☐ stets am besten geeignet, um den passenden Algorithmus zu wählen.

f) Der MergeSort-Algorithmus

☐ ist für alle Eingaben schneller als jede gute Implementierung von InsertionSort.

☒ ist für bestimmte Eingabeklassen signifikant schneller als eine deterministische Implementierung von Quicksort.

☐ ist im Schnitt um einen in der Eingabe linearen Faktor schneller als Quicksort.

☐ sortiert eine Eingabe im *Best Case* in linearer Zeit.

Aufgabe 10.1 – Multiple Choice

g) Beim Hashing mit *linear probing*

☐ werden Elemente, deren Schlüssel auf den gleichen Wert gehasht werden, in einer Liste abgelegt.

☐ ist die Hashfunktion nicht besonders wichtig, da auf ineffiziente Listen verzichtet wird.

☒ ist das Löschen von Elementen kompliziert, da Löcher in der Hashtabelle das Auffinden von anderen Elementen verhindern können.

h) Der PermutationSort-Algorithmus sortiert ein Feld, indem er die Elemente wiederholt umordnet und jede so entstandene Anordnung auf Sortiertheit prüft. Es werden systematisch alle möglichen Anordnungen durchprobiert. Wir gehen im Folgenden davon aus, dass PermutationSort ein Feld ohne Duplikate sortiert.

☐ Für die *Worst-Case*-Laufzeit f von PermutationSort gilt $f \in \mathcal{O}(n^4)$.

☒ Für die *Average-Case*-Laufzeit f von PermutationSort gilt $f \in \Theta(n * n!)$.

☒ Für die *Worst-Case*-Laufzeit f von PermutationSort gilt $f \in \mathcal{O}(n^n)$.

☐ Mit einer sehr geringen, positiven Wahrscheinlichkeit terminiert PermutationSort für bestimmte Eingaben niemals.

☒ PermutationSort hat eine lineare *Best-Case*-Laufzeit.

$O(\frac{n \cdot n!}{2})$

$n!$

$\cancel{n} \cdot (n \cdot (n-1) \cdot (n-2) \ldots)$

$\cancel{n} \cdot (n-1) \cdot \ldots 2 \cdot \cancel{1}$

$\cancel{n} \cdot n \cdot \ldots n)$

$\cancel{n} \cdot \underbrace{n \cdot n \cdot n \cdot n}_{n-1} = n^{n-1}$

Aufgabe 10.1 – Multiple Choice

g) Beim Hashing mit *linear probing*

☐ werden Elemente, deren Schlüssel auf den gleichen Wert gehasht werden, in einer Liste abgelegt.

☐ ist die Hashfunktion nicht besonders wichtig, da auf ineffiziente Listen verzichtet wird.

☒ ist das Löschen von Elementen kompliziert, da Löcher in der Hashtabelle das Auffinden von anderen Elementen verhindern können.

h) Der PermutationSort-Algorithmus sortiert ein Feld, indem er die Elemente wiederholt umordnet und jede so entstandene Anordnung auf Sortiertheit prüft. Es werden systematisch alle möglichen Anordnungen durchprobiert. Wir gehen im Folgenden davon aus, dass PermutationSort ein Feld ohne Duplikate sortiert.

☐ Für die *Worst-Case*-Laufzeit f von PermutationSort gilt $f \in \mathcal{O}(n^4)$.

☒ Für die *Average-Case*-Laufzeit f von PermutationSort gilt $f \in \Theta(n * n!)$.

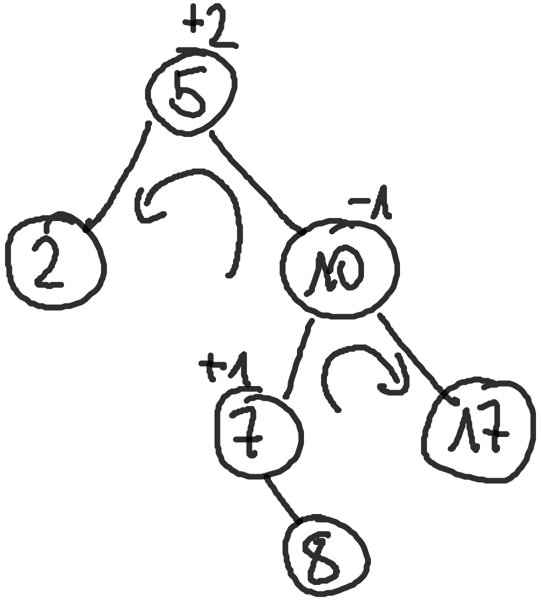
☒ Für die *Worst-Case*-Laufzeit f von PermutationSort gilt $f \in \mathcal{O}(n^n)$.

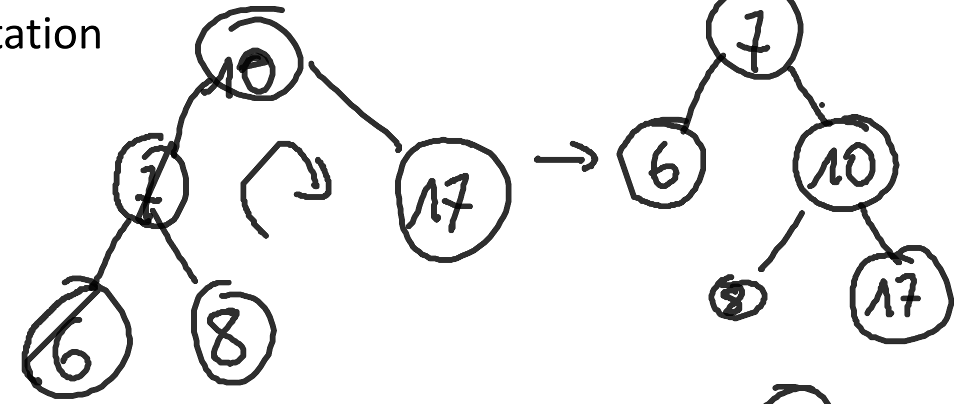
☐ Mit einer sehr geringen, positiven Wahrscheinlichkeit terminiert PermutationSort für bestimmte Eingaben niemals.

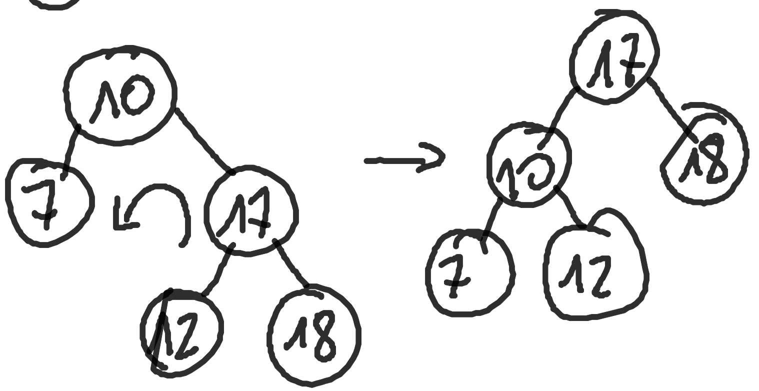
☒ PermutationSort hat eine lineare *Best-Case*-Laufzeit.

Wiederholung: AVL-Baum

- Binärer Suchbaum
- Invariante: -1 ≤ H(rechts) – H(links) ≤ 1
- Wird bei insert() und remove() balanciert (Rotationen)
- Wenn Elternknoten +2/-2 und Kindknoten +1/-1 = Einzelrotation
- Wenn Elternknoten +2/-2 und Kindknoten -1/+1 = Doppelrotation

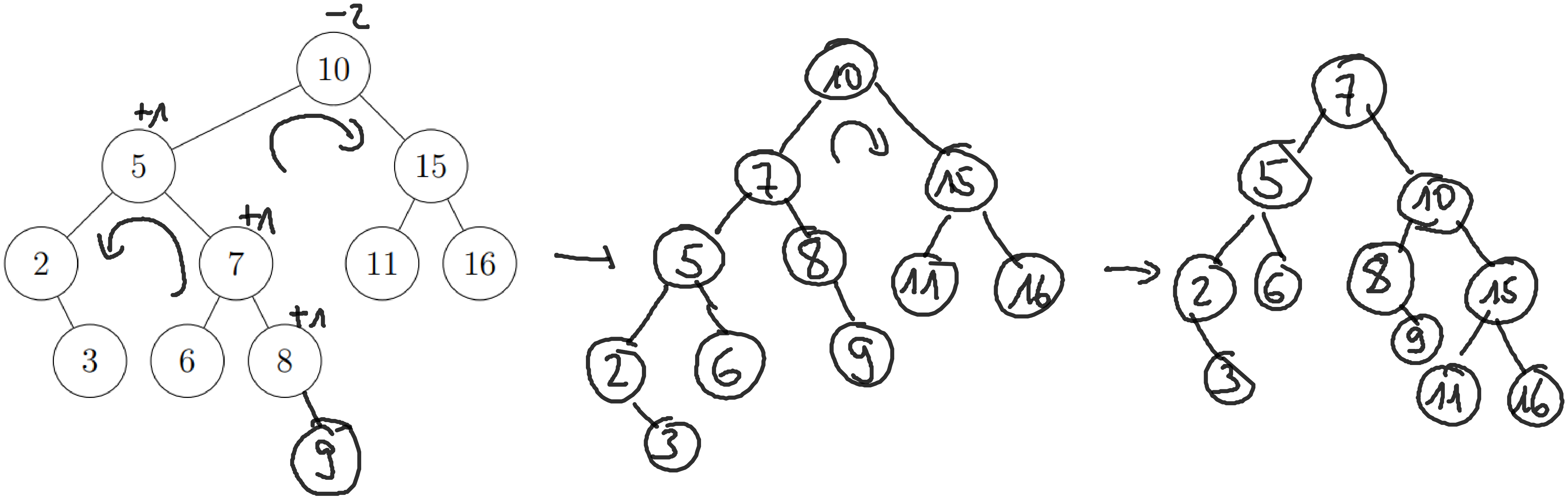






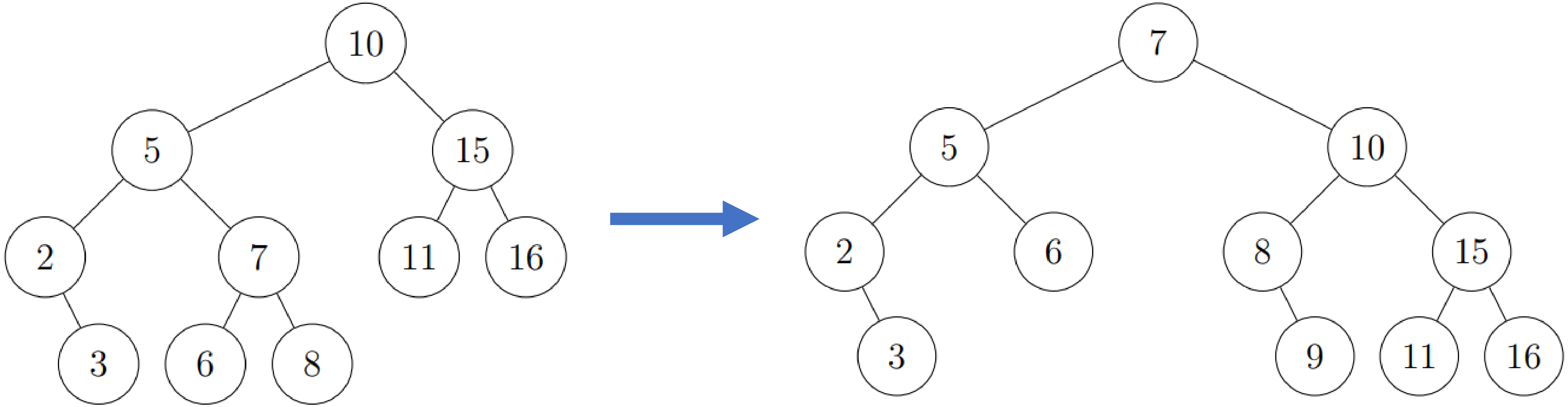
Aufgabe 10.2 – AVL

a) Führen Sie auf dem folgenden AVL-Baum eine `insert`-Operation des Schlüssels 9 durch. Zeichnen Sie den durch die Operation entstehenden AVL-Baum, und schreiben Sie dazu, ob keine Rotation, ob eine Rotation oder ob eine Doppelrotation durchgeführt wurde.



Aufgabe 10.2 – AVL

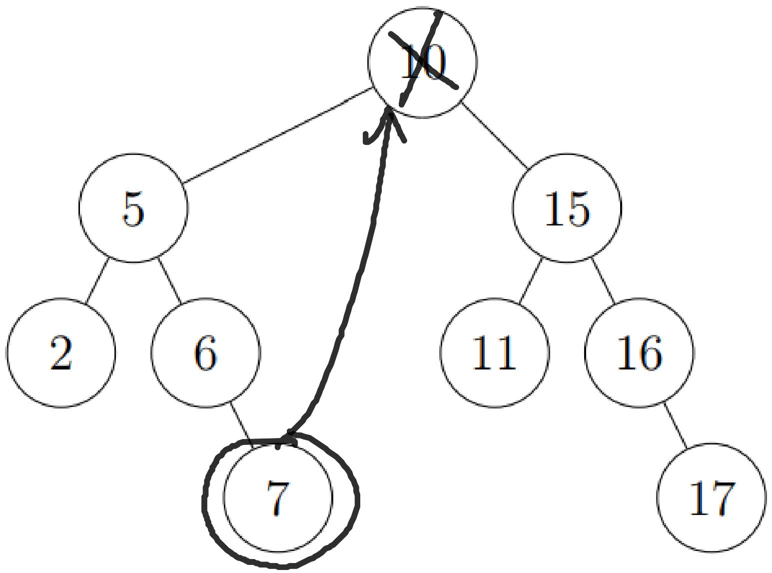
a) Führen Sie auf dem folgenden AVL-Baum eine `insert`-Operation des Schlüssels 9 durch. Zeichnen Sie den durch die Operation entstehenden AVL-Baum, und schreiben Sie dazu, ob keine Rotation, ob eine Rotation oder ob eine Doppelrotation durchgeführt wurde.

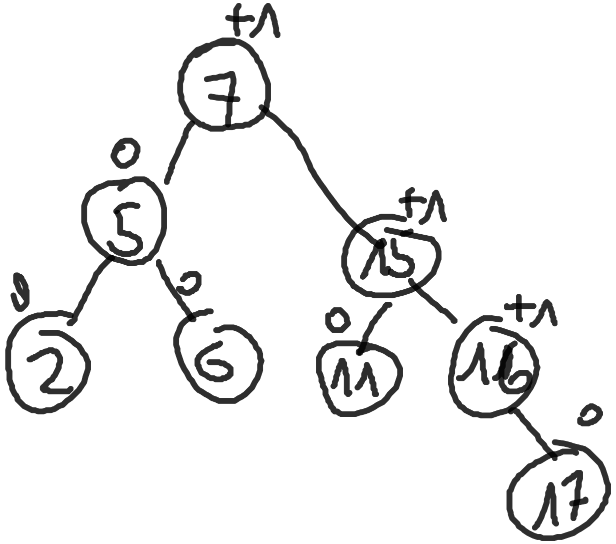


Einfügen am Knoten 8, dann Doppelrotation

Aufgabe 10.2 – AVL

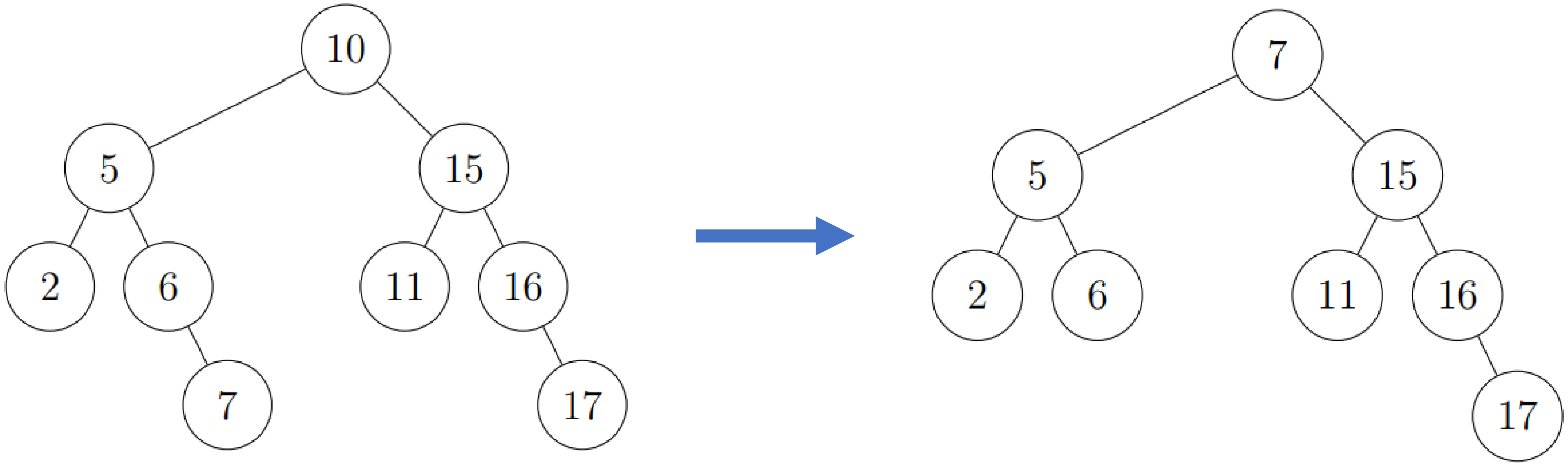
b) Führen Sie auf dem folgenden AVL-Baum eine `remove`-Operation des Schlüssels 10 durch. Zeichnen Sie den durch die Operation entstehenden AVL-Baum, und schreiben Sie dazu, ob keine Rotation, ob eine Rotation oder ob eine Doppelrotation durchgeführt wurde.





Aufgabe 10.2 – AVL

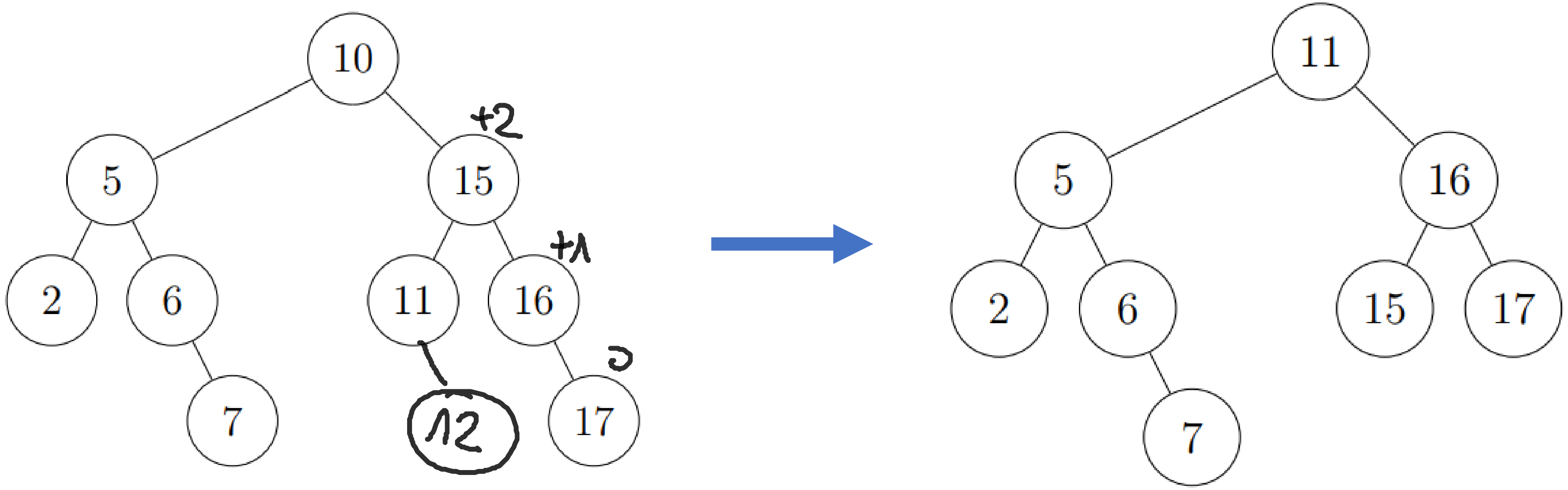
b) Führen Sie auf dem folgenden AVL-Baum eine `remove`-Operation des Schlüssels 10 durch. Zeichnen Sie den durch die Operation entstehenden AVL-Baum, und schreiben Sie dazu, ob keine Rotation, ob eine Rotation oder ob eine Doppelrotation durchgeführt wurde.



1) Vorgänger rückt nach, 7 in die Wurzel, keine Rotation

Aufgabe 10.2 – AVL

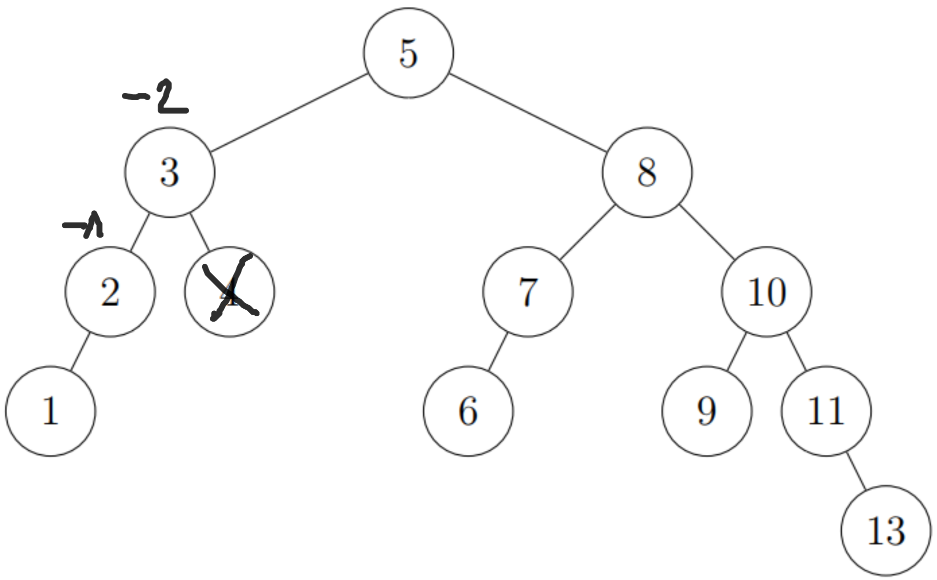
b) Führen Sie auf dem folgenden AVL-Baum eine `remove`-Operation des Schlüssels 10 durch. Zeichnen Sie den durch die Operation entstehenden AVL-Baum, und schreiben Sie dazu, ob keine Rotation, ob eine Rotation oder ob eine Doppelrotation durchgeführt wurde.

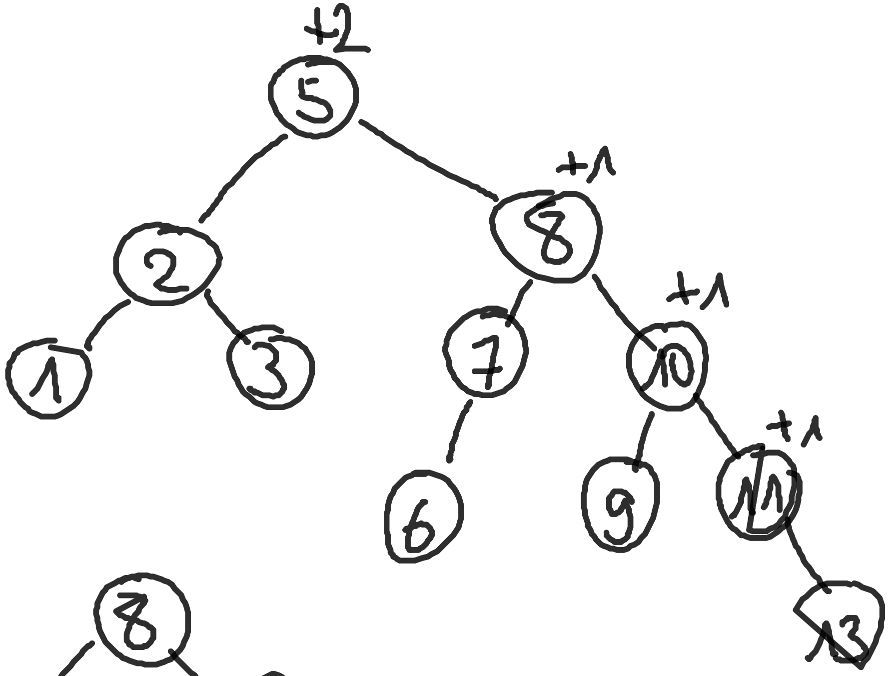


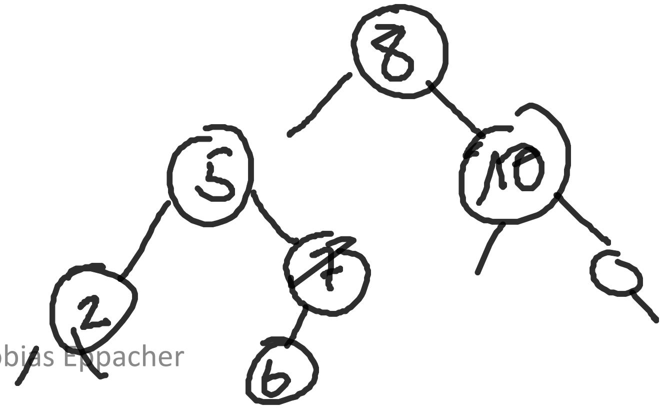
2) Nachfolger rückt nach, 11 in die Wurzel,
Linksrotation im rechten Teilbaum

Aufgabe 10.2 – AVL

c) Führen Sie auf dem folgenden AVL-Baum eine `remove`-Operation des Schlüssels 4 durch. Zeichnen Sie **für jede dabei durchgeführte Rotation** den durch die Rotation entstehenden Baum.

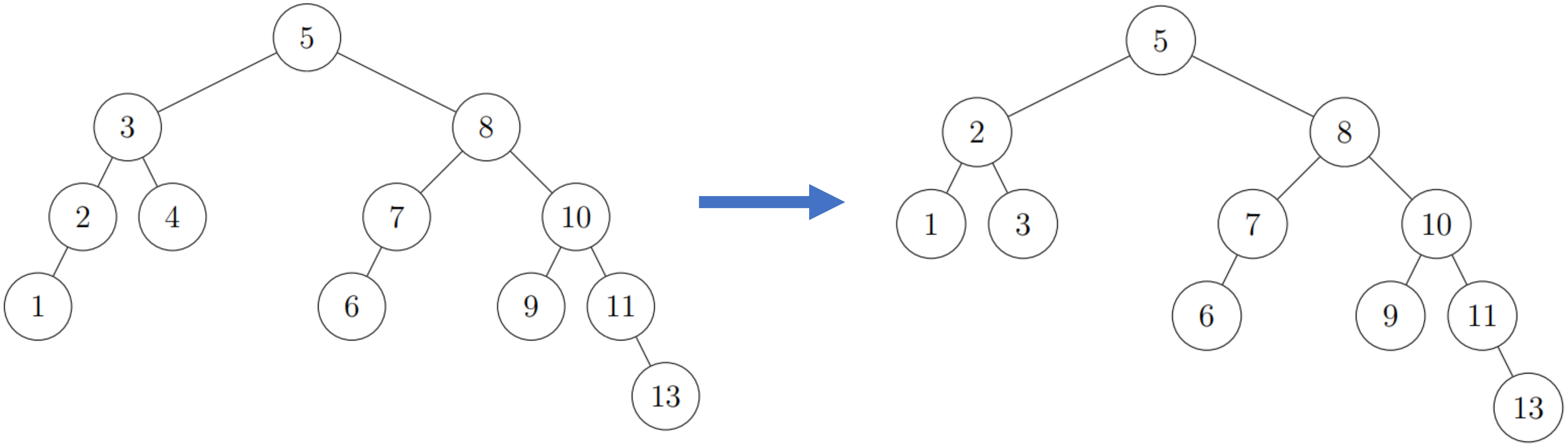






Aufgabe 10.2 – AVL

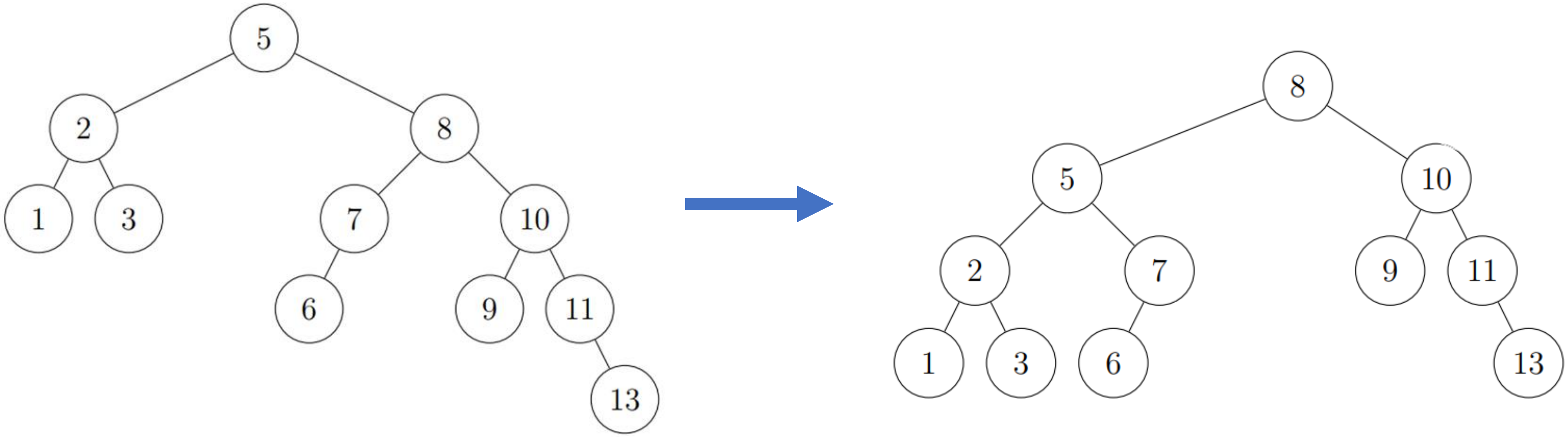
c) Führen Sie auf dem folgenden AVL-Baum eine `remove`-Operation des Schlüssels 4 durch. Zeichnen Sie **für jede dabei durchgeführte Rotation** den durch die Rotation entstehenden Baum.



Schritt 1: 4 wird gelöscht, Rechtsrotation im linken Teilbaum

Aufgabe 10.2 – AVL

c) Führen Sie auf dem folgenden AVL-Baum eine `remove`-Operation des Schlüssels 4 durch. Zeichnen Sie **für jede dabei durchgeführte Rotation** den durch die Rotation entstehenden Baum.

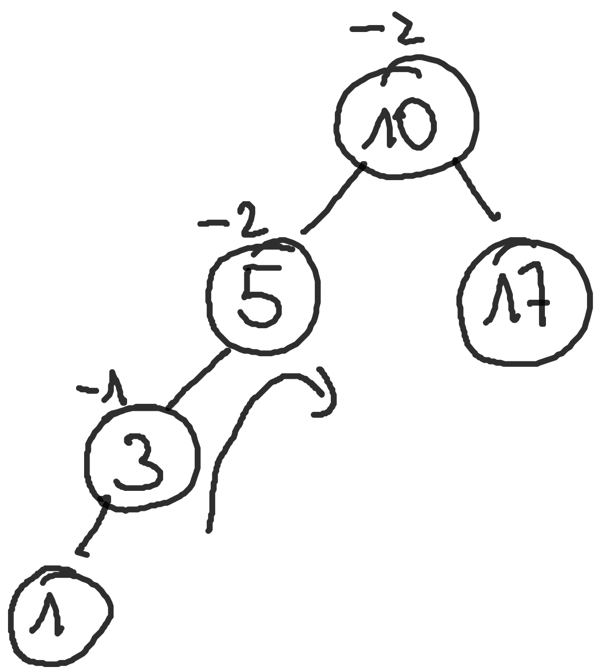


Schritt 2: Linksrotation zur Balancierung

Aufgabe 10.3 – AVL Bebaumung

Gegeben sei ein AVL-Baum der nur aus einem Knoten mit Schlüssel 10 besteht. Fügen Sie nacheinander die Schlüssel 5, 17, 3, 1, 4 ein. Löschen Sie dann den Schlüssel 4, und fügen Sie dann die Schlüssel 8, 2, 7, 6, 9 ein. Löschen Sie dann die Knoten mit den Schlüsseln 2, 1, 8. Zeichnen Sie den AVL-Baum für jede Einfüge- bzw. Löschoperation und geben Sie an, ob Sie keine, eine Einfach- oder Doppelrotation durchgeführt haben.

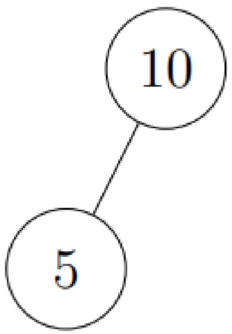
Insert 5, 17, 3, 1, 4
Remove 4
Insert 8, 2, 7, 6, 9
Remove 2, 1, 8



Aufgabe 10.3 – AVL Bebaumung

Insert	5, 17, 3, 1, 4
Remove	4
Insert	8, 2, 7, 6, 9
Remove	2, 1, 8

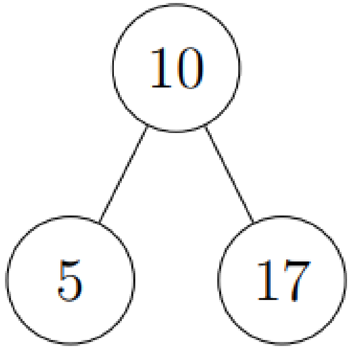
insert von 5 (ohne Rotation):



Aufgabe 10.3 – AVL Bebaumung

Insert 5, 17, 3, 1, 4
Remove 4
Insert 8, 2, 7, 6, 9
Remove 2, 1, 8

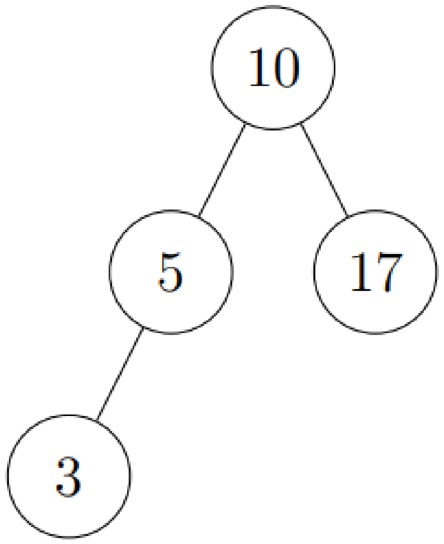
insert von 17 (ohne Rotation):



Aufgabe 10.3 – AVL Bebaumung

Insert	5, 17, 3, 1, 4
Remove	4
Insert	8, 2, 7, 6, 9
Remove	2, 1, 8

insert von 3 (ohne Rotation):

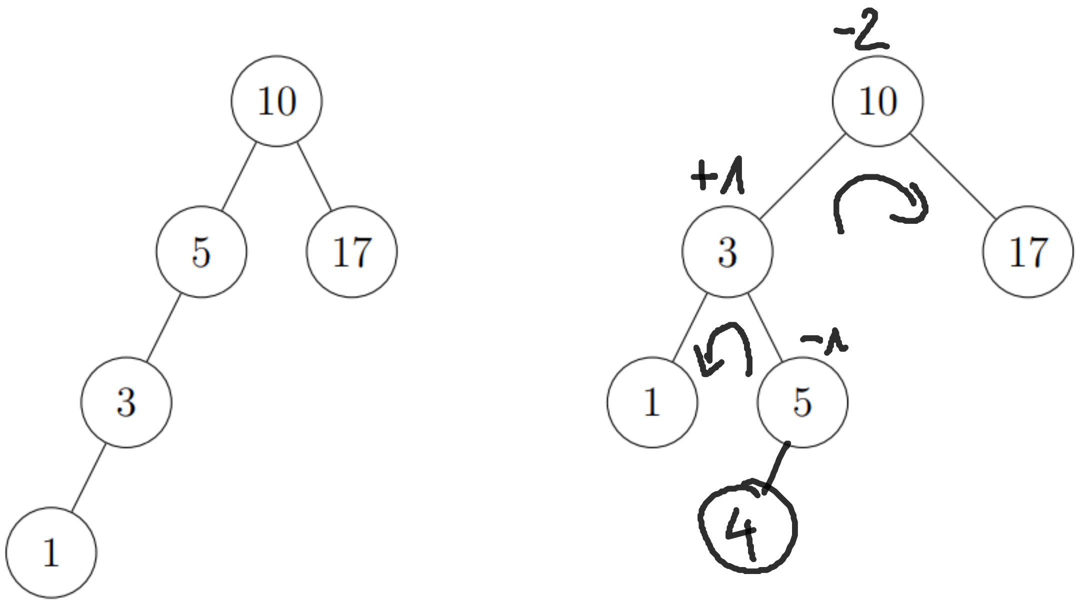


Aufgabe 10.3 – AVL Bebaumung

Insert	5, 17, 3, 1, 4
Remove	4
Insert	8, 2, 7, 6, 9
Remove	2, 1, 8

insert von 1 mit Rotation (nach rechts) am Knoten 5:

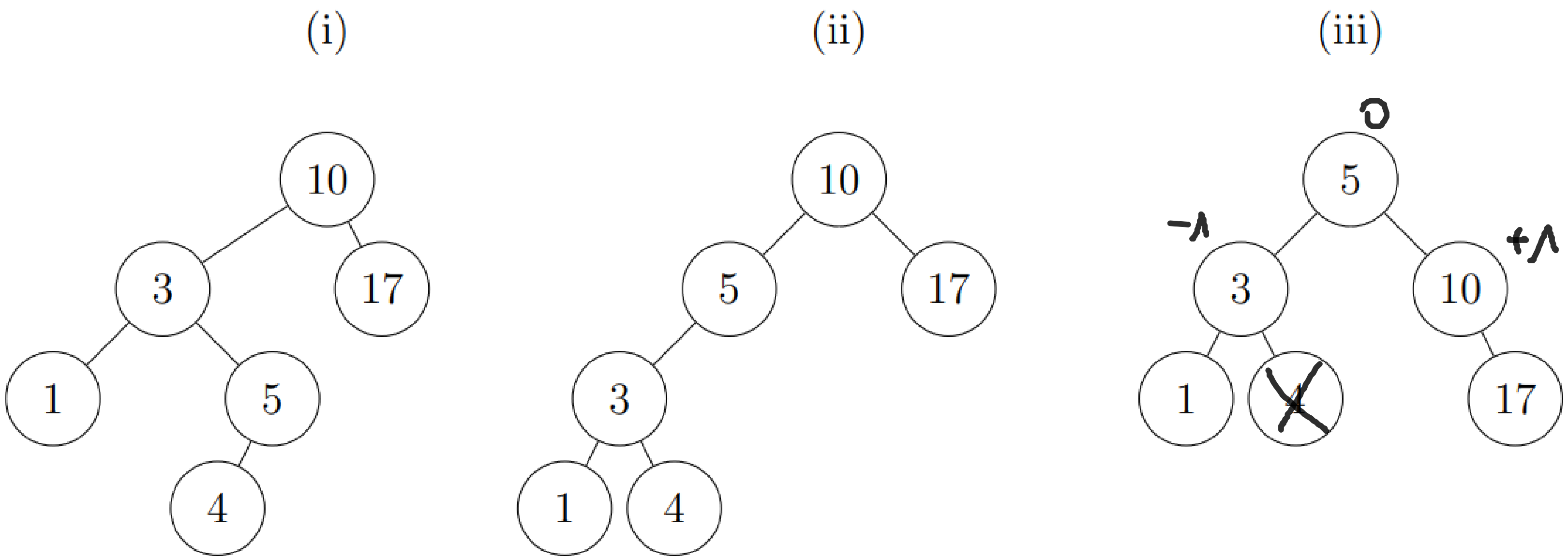
(i) (ii)



Aufgabe 10.3 – AVL Bebaumung

Insert 5, 17, 3, 1, 4
Remove 4
Insert 8, 2, 7, 6, 9
Remove 2, 1, 8

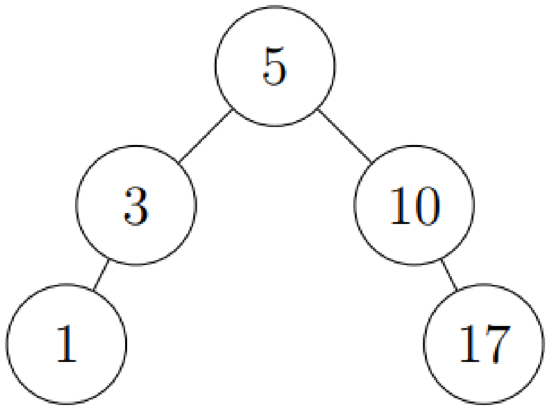
insert von 4 mit Doppelrotation am Knoten 10 (zuerst eine Linksrotation am Knoten 3, dann eine Rechtsrotation am Knoten 10):



Aufgabe 10.3 – AVL Bebaumung

Insert	5, 17, 3, 1, 4
Remove	4
Insert	8, 2, 7, 6, 9
Remove	2, 1, 8

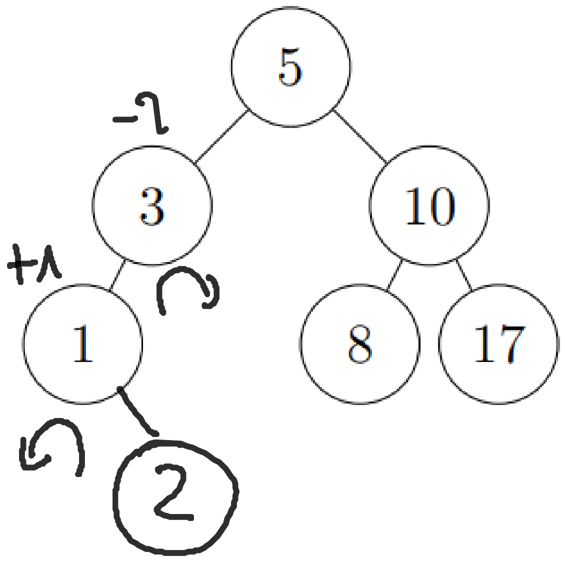
Löschen von 4 (ohne Rotation):



Aufgabe 10.3 – AVL Bebaumung

Insert 5, 17, 3, 1, 4
Remove 4
Insert 8, 2, 7, 6, 9
Remove 2, 1, 8

insert von 8 (ohne Rotation):

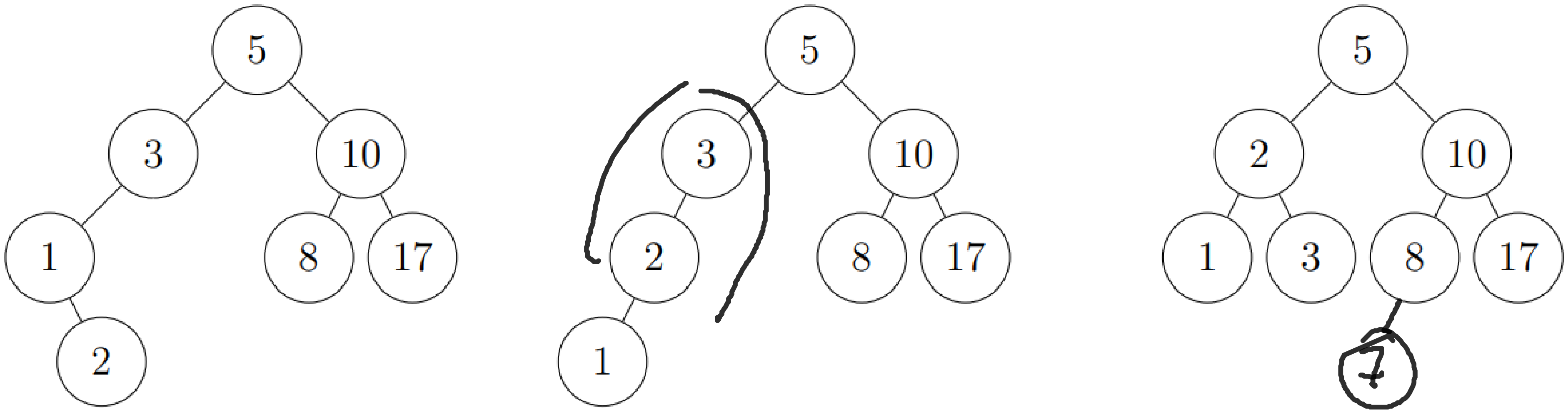


Aufgabe 10.3 – AVL Bebaumung

Insert	5, 17, 3, 1, 4
Remove	4
Insert	8, 2, 7, 6, 9
Remove	2, 1, 8

insert von 2 mit Doppelrotation am Knoten 3 (zuerst eine Linksrotation am Knoten 1, dann eine Rechtsrotation am Knoten 3):

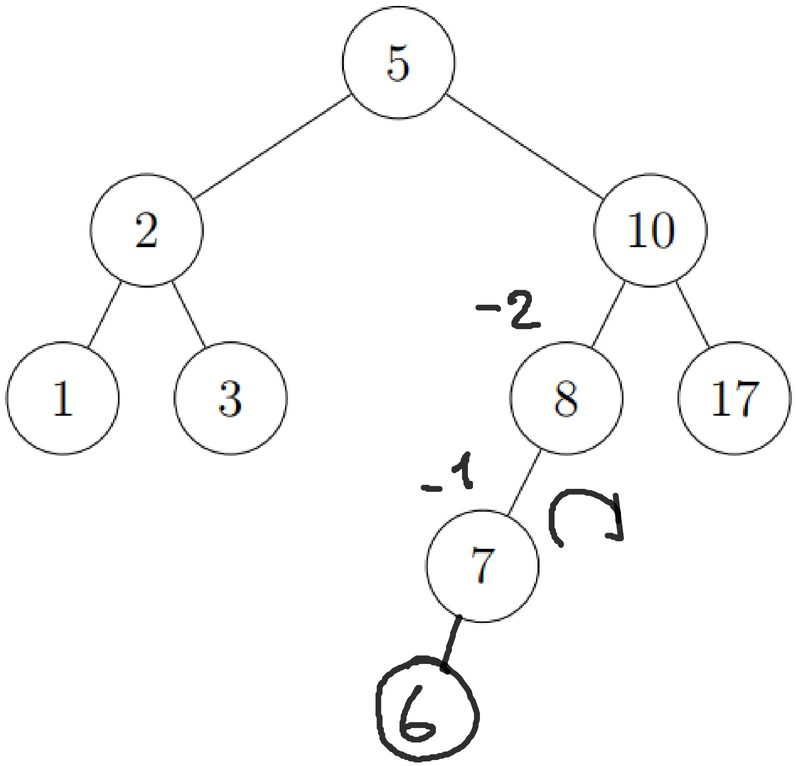
(i) (ii) (iii)



Aufgabe 10.3 – AVL Bebaumung

Insert 5, 17, 3, 1, 4
Remove 4
Insert 8, 2, 7, 6, 9
Remove 2, 1, 8

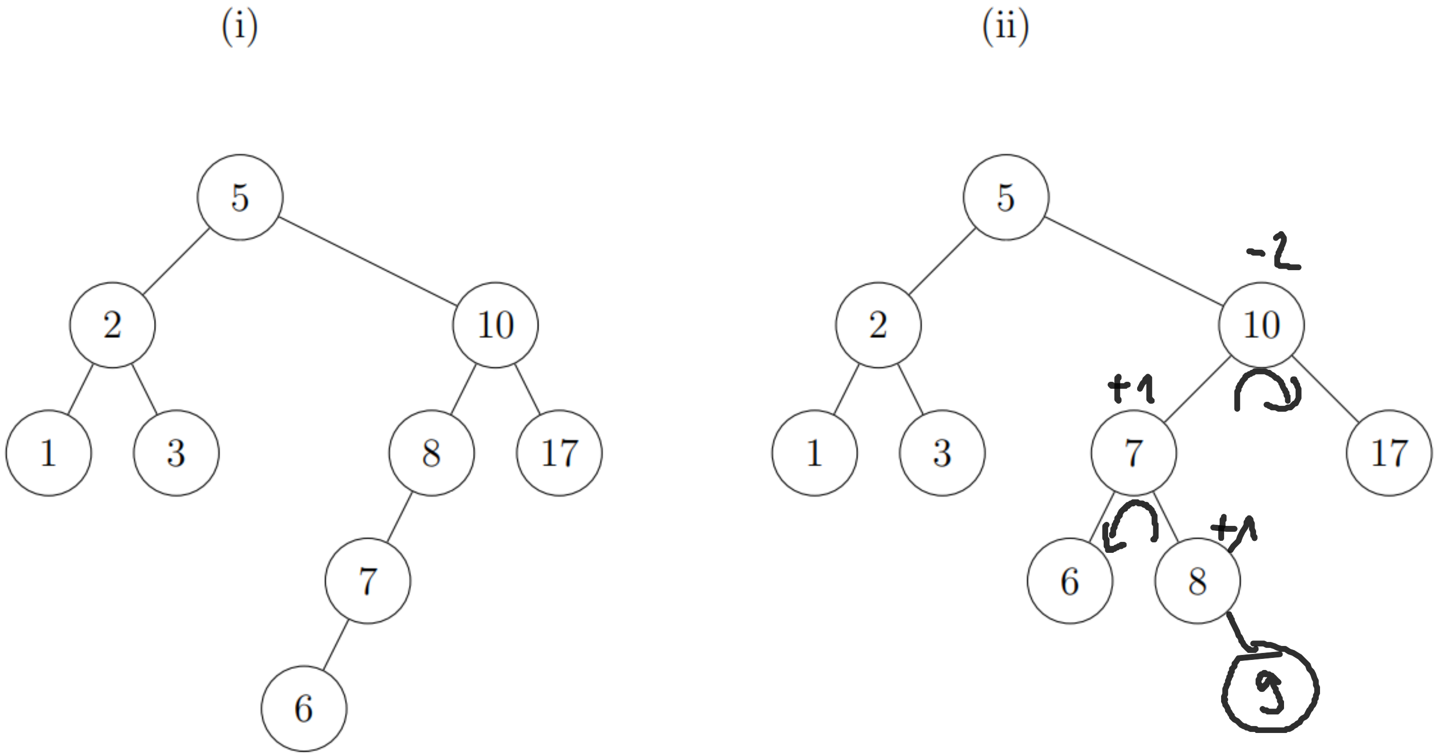
insert von 7 (ohne Rotation):



Aufgabe 10.3 – AVL Bebaumung

Insert 5, 17, 3, 1, 4
Remove 4
Insert 8, 2, 7, 6, 9
Remove 2, 1, 8

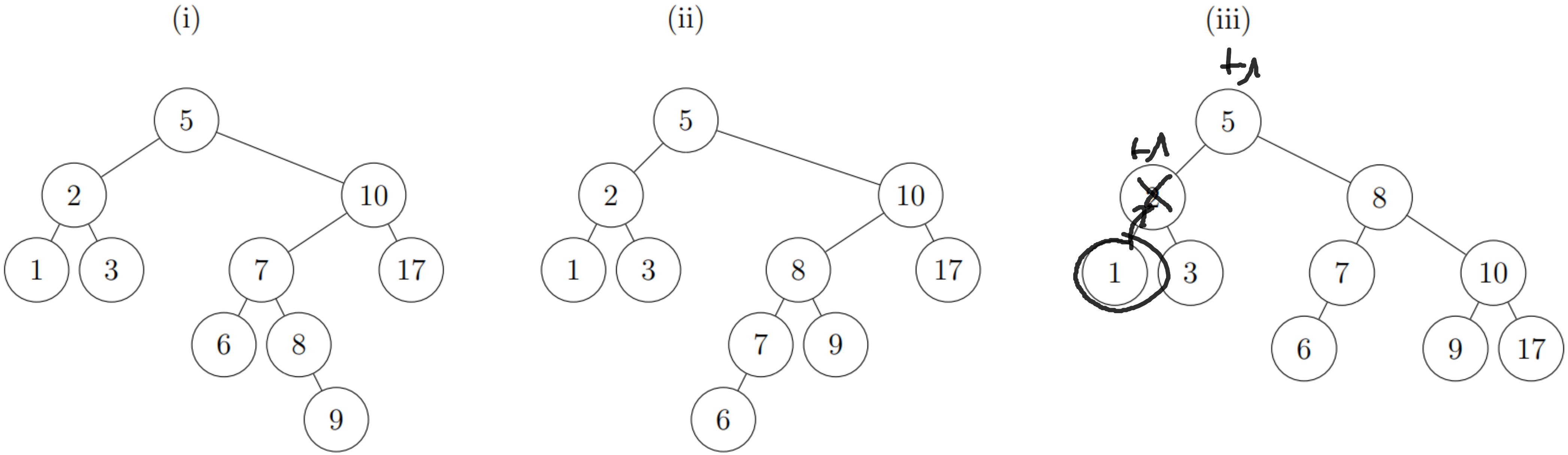
insert von 6 mit Rotation (nach rechts) am Knoten 8:



Aufgabe 10.3 – AVL Bebaumung

Insert 5, 17, 3, 1, 4
Remove 4
Insert 8, 2, 7, 6, 9
Remove 2, 1, 8

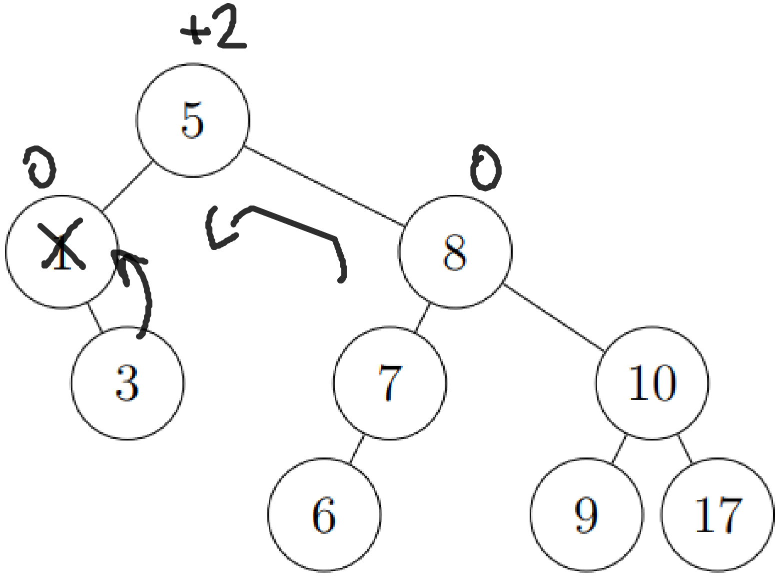
insert von 9 mit Doppelrotation am Knoten 10 (zuerst eine Linksrotation am Knoten 7, dann eine Rechtsrotation am Knoten 10):



Aufgabe 10.3 – AVL Bebaumung

Insert 5, 17, 3, 1, 4
Remove 4
Insert 8, 2, 7, 6, 9
Remove 2, 1, 8

Löschen von 2 (ohne Rotation):

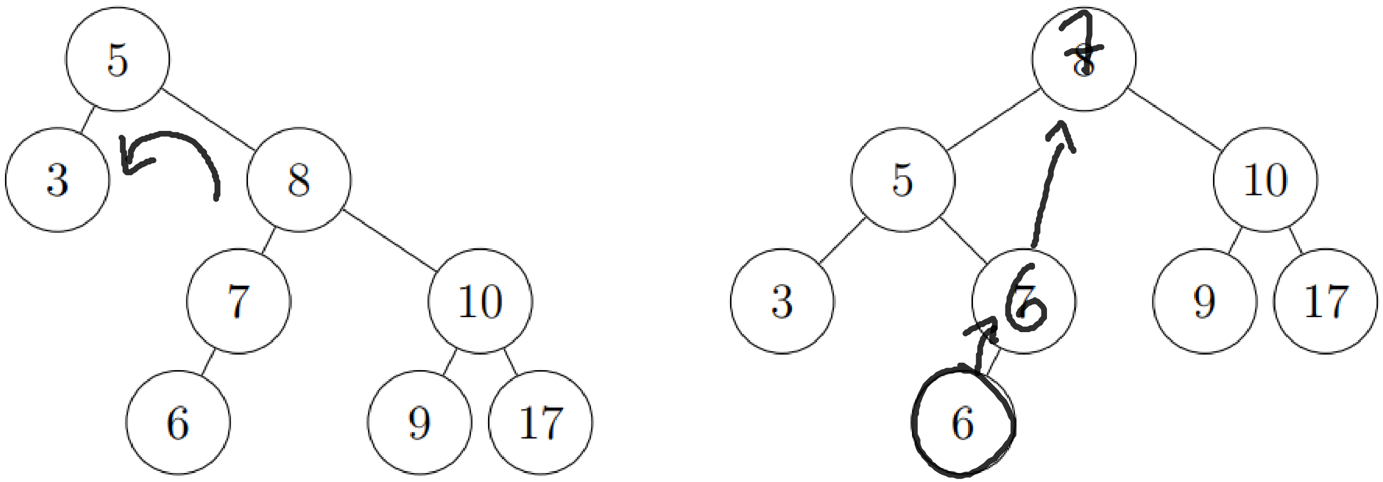


Aufgabe 10.3 – AVL Bebaumung

Insert	5, 17, 3, 1, 4
Remove	4
Insert	8, 2, 7, 6, 9
Remove	2, 1, 8

Löschen von 1 mit Rotation (nach links) am Knoten 5:

(i) (ii)



Aufgabe 10.3 – AVL Bebaumung

Insert 5, 17, 3, 1, 4
Remove 4
Insert 8, 2, 7, 6, 9
Remove 2, 1, 8

Löschen von 8 (ohne Rotation):