Lehrstuhl für Informatik I
Informationsstrukturen und
wissensbasierte Systeme
Universität Würzburg

Würzburg, den 26. Januar 2001

Prof. Dr. D. Seipel
Dipl.-Inform. I. Demgensky

Klausur
zur Vorlesung und Übung
»Praktische Informatik I« WS 2000/01

Name	
Vorname	

Aufgabe 1 – Verständnisfragen

Die folgenden Fragen sind durch Ankreuzen der richtigen Antworten zu lösen. Es können pro Teilaufgabe *mehrere* Antworten richtig sein! Für jede vollständig richtig gelöste Teilaufgabe gibt es einen Punkt. Für falsch gelöste Teilaufgaben gibt es einen Minuspunkt. Teilaufgaben, die nicht beantwortet werden, gehen nicht in die Wertung ein. Minuspunkte wirken sich nur innerhalb der Aufgabe 1 aus.

a) Seien $f, g\colon \mathbb{N} \to \mathbb{R}$ zwei Funktionen für die gilt: $\mathcal{O}(f) \nsubseteq \mathcal{O}(g)$. Was kann man daraus sofort schließen?

- ☐ $\mathcal{O}(g) \subseteq \mathcal{O}(f)$
- ☐ $f \notin \Omega(g)$
- ☐ $g \in \mathcal{O}(f)$
- ☐ keine der drei Aussagen.

b) Es sei nun $f \in \Omega(g)$. Welche der folgenden Aussagen kann dann noch wahr sein?

- ☐ $g \in \mathcal{O}(f)$
- ☐ $f(n) < g(n)$ für alle $n \in \mathbb{N}$
- ☐ $\mathcal{O}(f) = \mathcal{O}(g)$
- ☐ keine der drei Aussagen.

c) Welches Sortierverfahren beruht auf einem Scanline-Prinzip?

- ☐ Mergesort
- ☐ Quicksort
- ☐ Heapsort
- ☐ keines der bisher genannten

d) Welches Sortierverfahren realisiert eine Divide-and-Conquer-Strategie?

- ☐ Mergesort
- ☐ Bubblesort
- ☐ Selectionsort
- ☐ Quicksort

e) Wieviele Kanten hat ein vollbesetzter binärer Wurzelbaum der Höhe h?

- ☐ $2^{h+1} - 2$
- ☐ $2^h - 1$
- ☐ $2^{h-1} + 1$
- ☐ $2^{h+1} - 1$

f) Mit welcher Durchlaufordnung kann man aus einem binären Suchbaum eine absteigend sortierte Folge generieren?

- ☐ LWR
- ☐ RWL
- ☐ WRL
- ☐ RLW

g) Welche Durchlaufordnung gehört nicht zu den Präordnungen?

- ☐ RLW
- ☐ LWR
- ☐ LRW
- ☐ RWL

7 Punkte

Aufgabe 2 – $\mathcal{O}$-Notation

a) Ordnen Sie die folgenden Mengen bezüglich Inklusion.

$$\mathcal{O}(\log_2(n) + n^3), \mathcal{O}(128n^3 + 22n^2 + 15), \mathcal{O}(\sqrt[3]{n} + n), \mathcal{O}(\sqrt{n}),$$
$$\mathcal{O}(\log_{10} n), \mathcal{O}(\log_2 124^{1000})$$

2 Punkte

b) Es seien $f, g\colon \mathbb{N} \to \mathbb{R}^+$ zwei beliebige, positive Funktionen.
Wir definieren ferner $(f + g)(n) := f(n) + g(n)$, sowie $h(n) := \max\{f(n), g(n)\}$, für $n \in \mathbb{N}$.
Zeigen oder widerlegen Sie dann folgende Behauptung:

$$(f + g) \in \mathcal{O}(h)$$

3 Punkte

c) Für welche der folgenden Paare von Funktionen $f, g\colon \mathbb{N} \to \mathbb{R}$ gilt:
$f \in \mathcal{O}(g)$ bzw. $f \in \Omega(g)$? Begründen Sie ihre Angaben.

i)	$f(n) := \sqrt{n} + 25$	$g(n) := 5n$
ii)	$f(n) := \log_4 n$	$g(n) := \log_8 n + \log_4 n$
iii)	$f(n) := n^2$	$g(n) := n(\log_2 n) + n$
iv)	$f(n) := 73n^4 + 15n + 27n^2$	$g(n) := n^4$
v)	$f(n) := n^6 + \log_4 n$	$g(n) := n^6 + \sqrt[4]{n}$
vi)	$f(n) := \sqrt{n} + 5n$	$g(n) := 2450 \cdot \sqrt{n}$

6 Punkte

Aufgabe 3 – Heaps, Heapsort

a) Erzeugen sie aus folgendem Array durch Versickern (`heapaufbau1()` mit `downheap()`) die Arraydarstellung eines binären Min-Heaps. Geben Sie hierbei nach jedem Schlüsseltausch das veränderte Array als Zwischenschritt an.

```
int[] A = {98, 81, 17, 70, 19, 15, 16, 56, 65, 28, 18}
```

4 Punkte

b) Transformieren Sie die Arraydarstellung des resultierenden Heaps aus Teilaufgabe a) in die Baumdarstellung. Erzeugen Sie nun aus der Baumdarstellung eine aufsteigend sortierte Folge, indem Sie wiederholt den kleinsten Schlüsselwert löschen. Geben Sie nach jedem Löschen eines Schlüsselwertes jeweils den resultierenden Baum als Zwischenschritt an. **4 Punkte**

Aufgabe 4 – Quicksort

a) Wenden Sie den Quicksort-Algorithmus an, um die Zahlenfolge

$$S = (9, 7, 54, 12, 71, 6, 11, 50, 68, 20)$$

aufsteigend zu sortieren. Benutzen Sie als Pivotelement jeweils das *zweite* Elemente der Folge und geben Sie den Aufrufbaum an. **2 Punkte**

b) Ändern Sie die Reihenfolge der Elemente von S so ab, dass für die Pivotstrategie aus Teilaufgabe a) eine worst-case Folge entsteht. **2 Punkte**

c) Da die Laufzeit des Quicksort-Algorithmus bekanntermaßen stark von der Reihenfolge der zu sortierenden Schlüssel abhängt, wollen wir nun eine Pivot-Strategie betrachten, die unabhängig von der Reihenfolge ist. Wir definieren hierzu zunächst die „Mitte“ einer Schlüsselmenge S als

$$\mathrm{Mitte}(S) := \left\lfloor \frac{\max_{s \in S}(s) + \min_{s \in S}(s)}{2} \right\rfloor$$

Wir wählen dann als Pivotelement $\mathrm{Pivot}(S)$ das Element der Schlüsselmenge, das den geringsten Abstand zur Mitte hat. Da es möglicherweise zwei Schlüssel mit dem gleichen Abstand zur Mitte gibt, wählen wir davon den Schlüssel mit dem kleineren Wert. (Folglich ist $\mathrm{Pivot}(S) \leq \mathrm{Mitte}(S)$).

Wenden Sie den Quicksort-Algorithmus mit dieser neuen Pivotstrategie auf die Folge aus Teilaufgabe a) an. Geben Sie hierbei wieder den Aufrufbaum an. **4 Punkte**

d) Geben Sie jeweils eine Folge der Länge 10 mit geeigneten Schlüsselwerten an , sodass das Verfahren aus Teilaufgabe c) den worst-case bzw. den best-case annimmt. **4 Punkte**

Aufgabe 5 – Elementare Sortierverfahren

a) Sortieren Sie folgendes Array mittels Bubblesort. Geben Sie als Zwischenschritte nach jedem Schlüsseltausch das resultierende Array an.

```
int[] B = {35, 20, 7, 4, 16, 7, 3}
```

4 Punkte

b) Sortieren Sie folgendes Array mittels Selectionsort. Geben Sie als Zwischenschritte wieder nach jedem Schlüsseltausch das resultierende Array an.

```
int[] B = {32, 6, 20, 19, 14, 5, 20, 16, 14, 47}
```

4 Punkte

Aufgabe 6 – Stacks

Gegeben seien 3 Stacks, auf denen bereits beliebige natürliche Zahlen gespeichert sind. Geben Sie ein Verfahren an (verbale Beschreibung), mit dem sich die Zahlen aufsteigend sortieren lassen und zwar ohne weiteren externen Speicherplatz zu benutzen. D. h. die Schlüsselwerte dürfen nur zwischen den Stacks bewegt und *nicht* in externen Variablen gespeichert werden. Am Schluss soll einer der drei Stacks die sortierte Folge enthalten, mit dem kleinsten Schlüsselwert als top-Element.

Die Stacks unterstützen die Funktionen `pop()`, `top()`, `push()` und `empty()`. Gehen Sie vereinfachend davon aus, dass die Stacks beliebig viele Schlüsselwerte speichern können und somit nicht „überlaufen“. **8 Punkte**

Aufgabe 7 – Durchlaufordnungen

a) Geben Sie den binären Wurzelbaum an, der durch die beiden folgenden Durchlaufordnungen eindeutig gegeben ist:

$$\text{LWR} = (3, 5, 10, 9, 7, 1, 2, 8, 11, 6, 4)$$
$$\text{WRL} = (2, 8, 4, 11, 6, 3, 7, 1, 10, 9, 5)$$

4 Punkte

b) Geben Sie zwei alternative Durchlaufordnungen an, durch die der Baum aus Teilaufgabe a) eindeutig gegeben ist. **2 Punkte**

c) Geben Sie zwei *verschiedene* binäre Bäume T und T′ mit jeweils 5 Knoten an, sodass T und T′ die gleiche WLR- und LRW-Durchlaufordnung besitzen. **4 Punkte**

Aufgabe 8 – binäre Suchbäume

a) Tragen Sie in das Skelett des binären Baumes aus Abbildung 1 die Schlüsselwertmenge

$$M = \{30, 53, 47, 50, 17, 52, 31, 21, 72, 46, 51\}$$

so ein, dass die binäre Suchbaumeigenschaft erfüllt ist. **3 Punkte**

b) Geben Sie eine Reihenfolge an, in der man die Schlüssel aus Teilaufgabe a) in einen leeren Suchbaum einfügen muss, damit der angegebene Baum entsteht.

2 Punkte

c) Geben Sie einen Algorithmus an (verbale Beschreibung), der das Problem aus Teilaufgabe a) löst. Er soll also eine Liste mit Schlüsselwerten in ein vorgegebenes Baumskelett (binärer Baum) einfügen.

Das Baumskelett ist hierbei wie folgt entstanden: Es wurden alle Elemente der Liste (in einer nicht näher bekannten Reihenfolge) eingefügt und dann die Schlüsselwerte im fertigen Baum durch eine 0 überschrieben. **5 Punkte**

d) Die Frage ist nun, ob das Verfahren aus Teilaufgabe c) allgemein funktionieren kann. D. h., angenommen wir haben ein beliebiges binäres Baumskelett, das $n \in \mathbb{N}$ Schlüssel speichern kann, und eine Menge $M = \{s_1, s_2, \ldots, s_n\}$ von n paarweise verschiedenen Schlüsselwerten, ist es dann immer möglich die Schlüssel aus S so in das Baumskelett einzutragen, dass die binäre Suchbaumeigenschaft erfüllt ist? Begründen Sie Ihre Aussage! **4 Punkte**

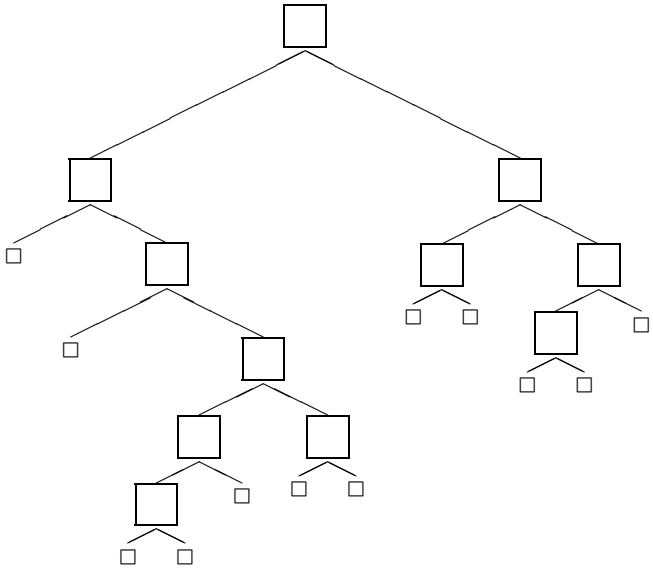


ABBILDUNG 1: Baumskelett

Aufgabe 9 – AVL-Bäume

a) Fügen Sie folgende Schlüssel in der angegebenen Reihenfolge in einen AVL-Baum ein.

$$S = (45, 25, 15, 13, 12, 11, 8, 7, 50, 53, 35, 63, 52, 67, 9, 10)$$

Geben Sie dabei bei jedem Einfügeschritt den entstandenen Baum sowie die angewandten Rotationen an. **10 Punkte**

b) Geben Sie für den AVL-Baum T aus Abbildung 2 an, in welcher Reihenfolge die Schlüssel möglicherweise eingefügt wurden, wenn man davon ausgeht, dass keine Rotationen zur Rebalancierung notwendig waren. **4 Punkte**

c) Löschen Sie aus dem Baum T sukzessiv die Schlüsselwerte 16, 30, 40 und 11. Geben wieder nach jedem Löschvorgang den entstandenen Baum sowie die benutzten Rotationen an. **8 Punkte**

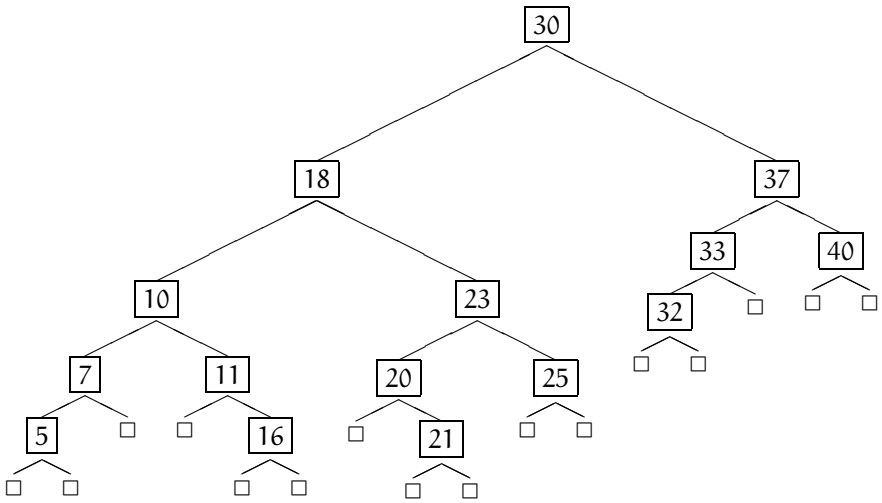


ABBILDUNG 2: Der AVL-Baum T

Lösung zu Aufgabe 1

a) Seien $f, g\colon \mathbb{N} \to \mathbb{R}$ zwei Funktionen für die gilt: $\mathcal{O}(f) \nsubseteq \mathcal{O}(g)$. Was kann man daraus sofort schließen?

- ☐ $\mathcal{O}(g) \subseteq \mathcal{O}(f)$
- ☐ $f \notin \Omega(g)$
- ☐ $g \in \mathcal{O}(f)$
- ☒ keine der drei Aussagen.

b) Es sei nun $f \in \Omega(g)$. Welche der folgenden Aussagen kann dann noch wahr sein?

- ☒ $g \in \mathcal{O}(f)$
- ☒ $f(n) < g(n)$ für alle $n \in \mathbb{N}$
- ☒ $\mathcal{O}(f) = \mathcal{O}(g)$
- ☐ keine der drei Aussagen.

c) Welches Sortierverfahren beruht auf einem Scanline-Prinzip?

- ☐ Mergesort
- ☐ Quicksort
- ☐ Heapsort
- ☒ keines der bisher genannten

d) Welches Sortierverfahren realisiert eine Divide-and-Conquer-Strategie?

- ☒ Mergesort
- ☐ Bubblesort
- ☐ Selectionsort
- ☒ Quicksort

e) Wieviele Kanten hat ein vollbesetzter binärer Wurzelbaum der Höhe h?

- ☒ $2^{h+1} - 2$
- ☐ $2^h - 1$
- ☐ $2^{h-1} + 1$
- ☐ $2^{h+1} - 1$

f) Mit welcher Durchlaufordnung kann man aus einem binären Suchbaum eine absteigend sortierte Folge generieren?

- ☐ LWR
- ☒ RWL
- ☐ WRL
- ☐ RLW

g) Welche Durchlaufordnung gehört nicht zu den Präordnungen?

- ☒ RLW
- ☒ LWR
- ☒ LRW
- ☒ RWL

Lösung zu Aufgabe 2

a) Die richtige Sortierung lautet:

$$\mathcal{O}(\log_2 124^{1000}) \subset \mathcal{O}(\log_{10} n) \subset \mathcal{O}(\sqrt{n}) \subset$$
$$\mathcal{O}(\sqrt[3]{n} + n) \subset \mathcal{O}(\log_2(n) + n^3) = \mathcal{O}(128n^3 + 22n^2 + 15)$$

b) Die Behauptung ist richtig. Es gilt: $(f + g) \in \mathcal{O}(h)$.

Beweis. Mit $h(n) := \max\{f(n), g(n)\}$ ist sofort klar, dass:

$$f(n) \leq h(n) \quad \forall n \in \mathbb{N}$$
$$g(n) \leq h(n) \quad \forall n \in \mathbb{N}$$

Daraus folgt aber auch, dass:

$$(f + g)(n) = f(n) + g(n) \leq h(n) + h(n) = 2 \cdot h(n) \quad \forall n \in \mathbb{N}$$

Wählt man also für die Konstante c den Wert 2, so folgt mit der Definition von $\mathcal{O}$ sofort die Behauptung. □

c) Die Ergebnisse im Überblick:

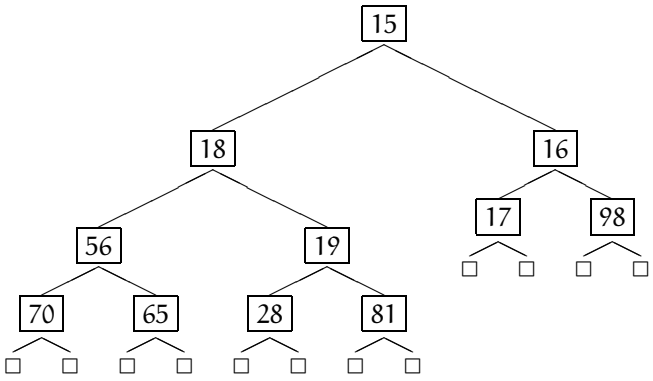
i)	$f \in \mathcal{O}(g)$	$f \notin \Omega(g)$
ii)	$f \in \mathcal{O}(g)$	$f \in \Omega(g)$
iii)	$f \notin \mathcal{O}(g)$	$f \in \Omega(g)$
iv)	$f \in \mathcal{O}(g)$	$f \in \Omega(g)$
v)	$f \in \mathcal{O}(g)$	$f \in \Omega(g)$
vi)	$f \notin \mathcal{O}(g)$	$f \in \Omega(g)$

Lösung zu Aufgabe 3

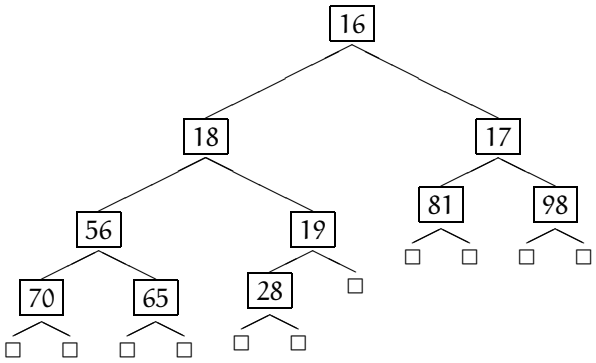
a) Die korrekten Zwischenschritte lauten:

(98, 81, 17, 70, 19, 15, 16, 56, 65, 28, 18)
(98, 81, 17, 70, **18**, 15, 16, 56, 65, 28, **19**)
(98, 81, 17, **56**, 18, 15, 16, **70**, 65, 28, 19)
(98, 81, **15**, 56, 18, **17**, 16, 70, 65, 28, 19)
(98, **18**, 15, 56, **81**, 17, 16, 70, 65, 28, 19)
(98, 18, 15, 56, **19**, 17, 16, 70, 65, 28, **81**)
(**15**, 18, **98**, 56, 19, 17, 16, 70, 65, 28, 81)
(15, 18, **16**, 56, 19, 17, **98**, 70, 65, 28, 81)

b) Heap in Baumdarstellung:

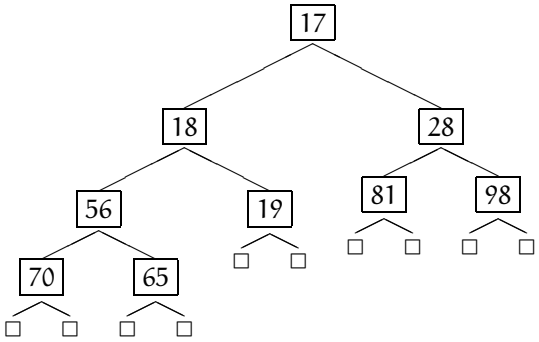


Heap nach Löschen der 15



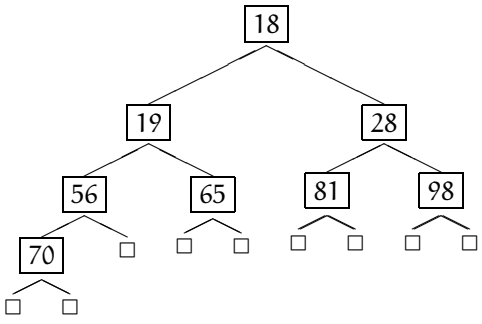
(15)

Heap nach Löschen der 16



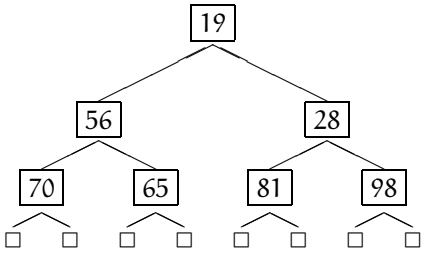
(15, 16)

Heap nach Löschen der 17



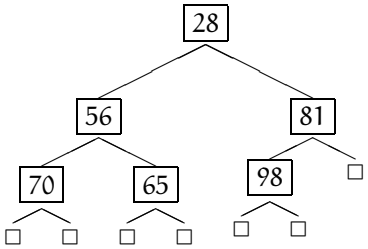
(15, 16, 17)

Heap nach Löschen der 18



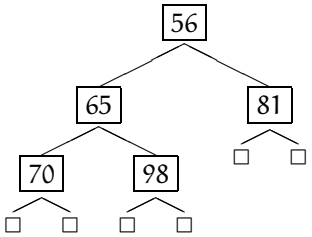
(15, 16, 17, 18)

Heap nach Löschen der 19



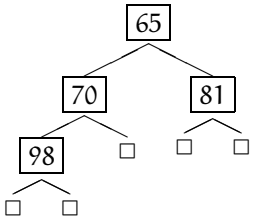
(15, 16, 17, 18, 19)

Heap nach Löschen der 28



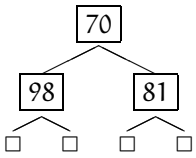
(15, 16, 17, 18, 19, 28)

Heap nach Löschen der 56



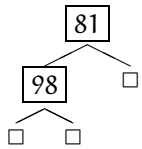
(15, 16, 17, 18, 19, 28, 56)

Heap nach Löschen der 65



(15, 16, 17, 18, 19, 28, 56, 65)

Heap nach Löschen der 70



(15, 16, 17, 18, 19, 28, 56, 65, 70)

Heap nach Löschen der 81

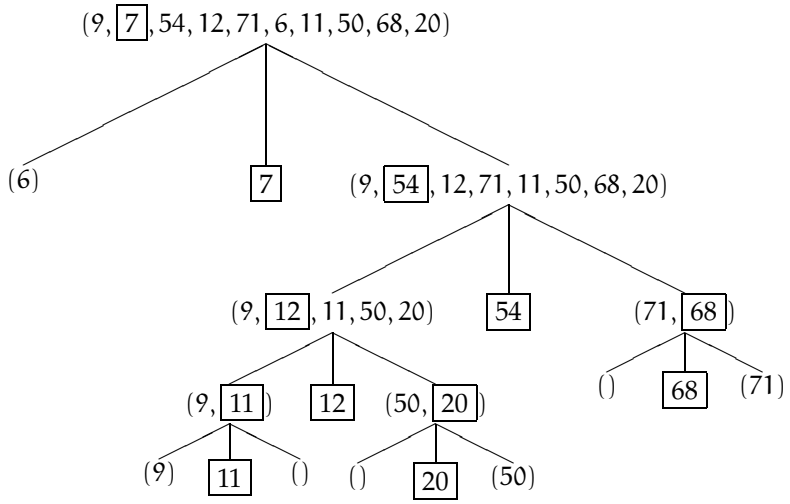
(15, 16, 17, 18, 19, 28, 56, 65, 70, 81)

Heap nach Löschen der 98

□ (15, 16, 17, 18, 19, 28, 56, 65, 70, 81, 98)

Lösung zu Aufgabe 4

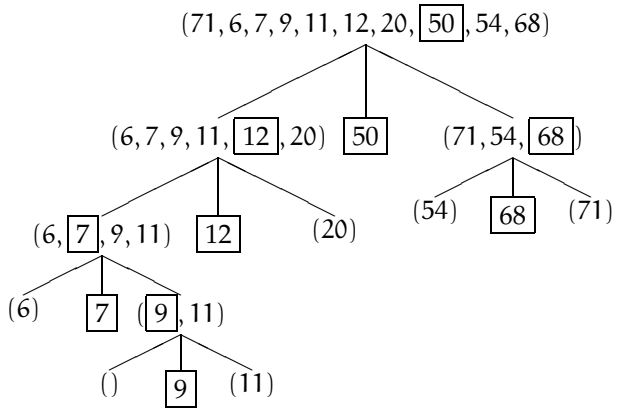
a) Der Aufrufbaum sieht wie folgt aus:



b) Eine worst-case Folge lautet:

$$S' = (71, 6, 7, 9, 11, 12, 20, 50, 54, 68)$$

c) Aufrufbaum mit der neuen Pivotstrategie:



d) Mögliche Folgen lauten:

$$S_{\text{best-case}} = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$$
$$S_{\text{worst-case}} = (1, 2, 4, 8, 16, 32, 64, 128, 256, 512)$$

Lösung zu Aufgabe 5

a) Die Schlüsselvertauschungen im Einzelnen:

(35,	20,	7,	4,	16,	7,	3)
(20,	35,	7,	4,	16,	7,	3)
(20,	7,	35,	4,	16,	7,	3)
(20,	7,	4,	35,	16,	7,	3)
(20,	7,	4,	16,	35,	7,	3)
(20,	7,	4,	16,	7,	35,	3)
(20,	7,	4,	16,	7,	3,	35)
(7,	20,	4,	16,	7,	3,	35)
(7,	4,	20,	16,	7,	3,	35)
(7,	4,	16,	20,	7,	3,	35)
(7,	4,	16,	7,	20,	3,	35)
(7,	4,	16,	7,	3,	20,	35)
(4,	7,	16,	7,	3,	20,	35)
(4,	7,	7,	16,	3,	20,	35)
(4,	7,	7,	3,	16,	20,	35)
(4,	7,	3,	7,	16,	20,	35)
(4,	3,	7,	7,	16,	20,	35)
(3,	4,	7,	7,	16,	20,	35)

b) Hier wieder die Schlüsselvertauschungen:

(32,	6,	20,	19,	14,	5,	20,	16,	14,	47)
(5,	6,	20,	19,	14,	32,	20,	16,	14,	47)
(5,	6,	14,	19,	20,	32,	20,	16,	14,	47)
(5,	6,	14,	14,	20,	32,	20,	16,	19,	47)
(5,	6,	14,	14,	16,	32,	20,	20,	19,	47)
(5,	6,	14,	14,	16,	19,	20,	20,	32,	47)

Lösung zu Aufgabe 6

Zur einfacheren Beschreibung des Verfahrens nummerieren wir die Stacks. Eine einfache Lösung gewinnt man durch Nachahmung von Selection-Sort. Zunächst bringt man alle Elemente der Stacks 1 und 2 auf Stack 3. Wir wollen nun das Maximum der Schlüsselwerte von Stack 3 bestimmen und das entsprechende Element auf Stack 1 legen. Das geht wie folgt:

Man legt das oberste Element von Stack 3 als neues, potentielles Maximum auf Stack 1. Jetzt werden alle Elemente von Stack 3 nach Stack 2 umgefüllt, wobei jedes Element mit dem potentiellen Maximum auf Stack 1 verglichen wird. Ist das umzufüllende Element kleiner, wandert es auf den Stack 2. Ist das umzufüllende Element jedoch größer, dann wird das bisherige potentielle Maximum von Stack 1 auf Stack 2 gelegt und das umzufüllende Element nimmt seinen Platz auf Stack 1 ein als neues potentielles Maximum. Ist der Umfüllvorgang beendet und somit Stack 3 leer, dann befindet sich auf Stack 1 das tatsächliche Maximum. Nun bringt man einfach alle Elemente von Stack 2 wieder auf Stack 3 (ohne jegliche Vergleiche).

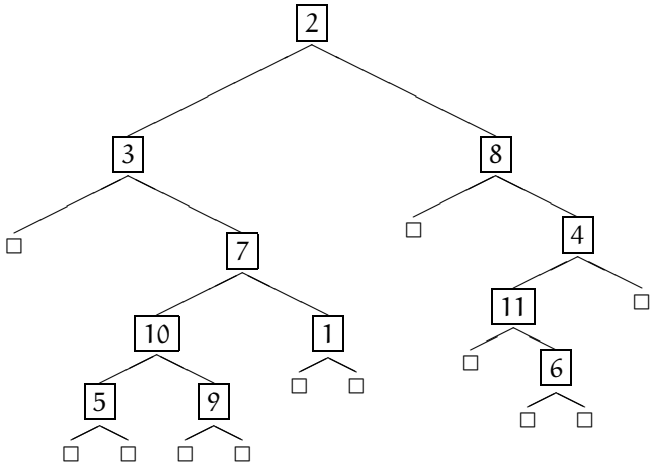
Diese Maximumsbestimmung wird solange iteriert, bis Stack 3 leer ist, dann enthält Stack 1 die sortierte Schlüsselmenge mit dem kleinsten Element als Topelement.

In Java könnte der Algorithmus ungefähr wie folgt aussehen:

```
while (!stack1.empty())
   stack3.push(stack1.pop());
while (!stack2.empty())
   stack3.push(stack2.pop());
while (!stack3.empty())
{
   stack1.push(stack3.pop());
   while (!stack3.empty())
      if (stack1.top() < stack3.top())
      {
         stack2.push(stack1.pop());
         stack1.push(stack3.pop());
      }
      else
         stack2.push(stack3.pop());
   while (!stack2.empty())
      stack3.push(stack2.pop());
}
```

Lösung zu Aufgabe 7

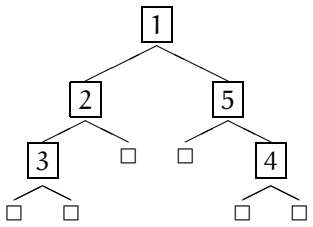
a) Der korrekte Baum lautet:

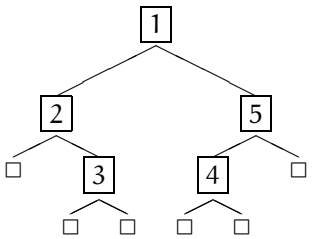


b) Alternative Durchlaufordnungen sind:

$$RWL = (4, 6, 11, 8, 2, 1, 7, 9, 10, 5, 3)$$
$$WLR = (2, 3, 7, 10, 5, 9, 1, 8, 4, 11, 6)$$

c) Die folgenden Bäume besitzen die gleiche WLR- und LRW-Durchlaufordnung:



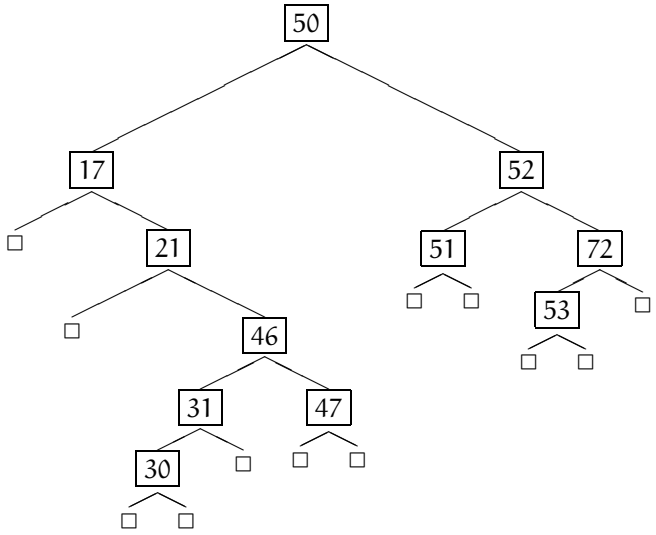


Man sieht:

$$LRW = (3, 2, 4, 5, 1)$$
$$WLR = (1, 2, 3, 5, 4)$$

Lösung zu Aufgabe 8

a) Nach korrektem Einfügen ergibt sich folgender Suchbaum:



b) Es ist jede Reihenfolge möglich, die gewährleistet, dass vor der Ausgabe eines Knotens sämtliche Vorgänger ausgegeben werden. Oder graphentheoretisch ausgedrückt: Betrachtet man den Baum als gerichtetetn Graphen mit Pfeilen von den Knoten zu ihren Söhnen, dann ist jede topologische Sortierung eine zulässige Folge. Also ist unter anderem eine WLR-Durchlaufordnung möglich:

$$(50, 17, 21, 46, 31, 30, 47, 52, 51, 72, 53)$$

c) Man sortiert zunächst die Schlüsselwerte aufsteigend mit einem der bekannten Sortierverfahren. Dann durchläuft man den Baum gemäß der LWR-Ordnung, fügt jeweils das erste Element der Liste ein und löscht dieses aus der Liste.

In Java könnte das so aussehen:

```
void fill_in(BinTree Baum, Queue Schluesselliste)
{
   Mergesort(Schluesselliste);
   fill_in(Baum.root, Queue Schluesselliste);
}


void fill_in(BinTreeNode Knoten, Queue Schluesselliste)
{
   if (Knoten == null)
      return;
   fill_in(Knoten.leftson, Schluesselliste);
   Knoten.key = Schluesselliste.pophead();
   fill_in(Knoten.rightson, Schluesselliste);
   return;
}
```

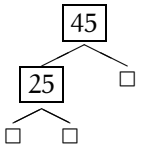
d) Das Verfahren aus Teilaufgabe c) funktioniert allgemein. Jeder beliebige binäre Baum kann gemäß der LWR-Ordnung durchlaufen werden. Es gibt genausoviele Schlüsselwerte wie Knoten, somit wird jedem Knoten genau ein Schlüsselwert zugewiesen. Die geforderte Suchbaumeigenschaft resultiert aus dem LWR-Durchlauf und der Tatsache, dass die

Schlüssel aufsteigend sortiert werden. Denn wenn ein Schlüsselwert s in einen Knoten v geschrieben wird, dann sind bereits sämtliche Knoten im Teilbaum des linken Sohnes von v mit Schlüsselwerten belegt, die allesamt kleiner sind als s, da sie früher aus der aufsteigend sortierten Liste entnommen wurden. Erst nachdem der Schlüsselwert s geschrieben ist, werden Schlüsselwerte in den Teilbaum des rechten Sohnes von v eingefügt und diese sind alle größer als s.

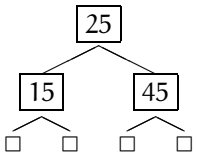
Lösung zu Aufgabe 9

a) Einfügen der 45 :

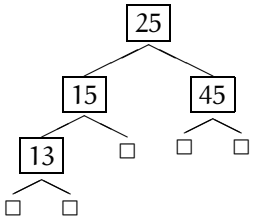
Einfügen der 25 :



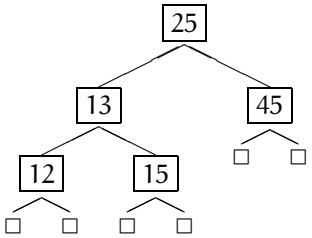
Einfügen der 15 :
Mit Rotation (rechts) im Teilbaum der 45:



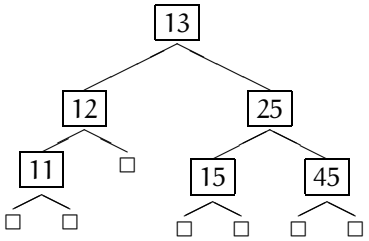
Einfügen der 13 :



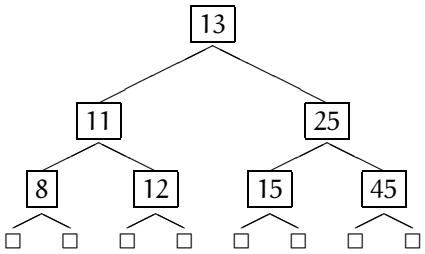
Einfügen der 12 :
Mit Rotation (rechts) im Teilbaum der 15:



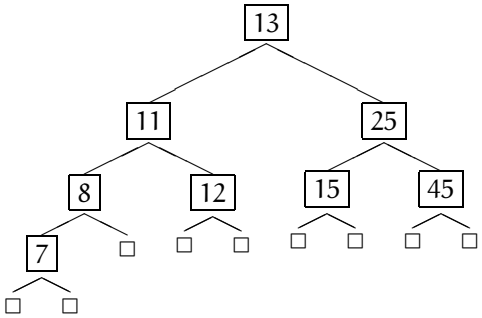
Einfügen der 11 :
Mit Rotation (rechts) im Teilbaum der 25:



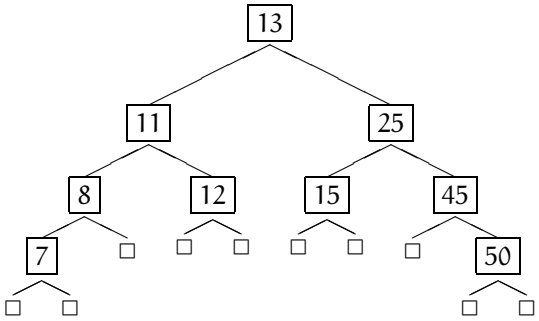
Einfügen der 8 :
Mit Rotation (rechts) im Teilbaum der 12:



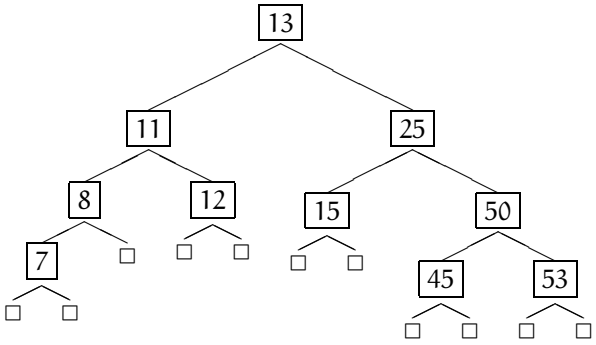
Einfügen der 7 :



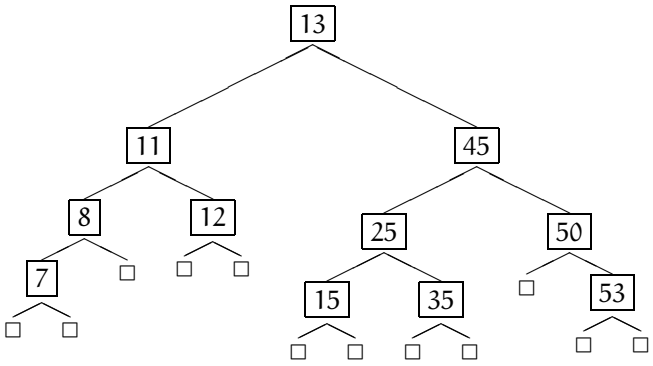
Einfügen der 50 :



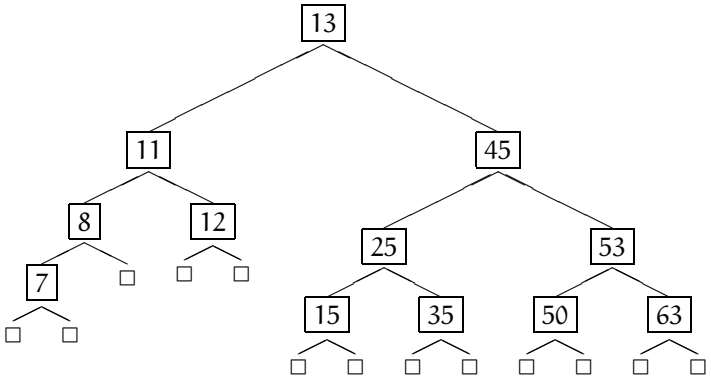
Einfügen der 53 :
Mit Rotation (links) im Teilbaum der 45:



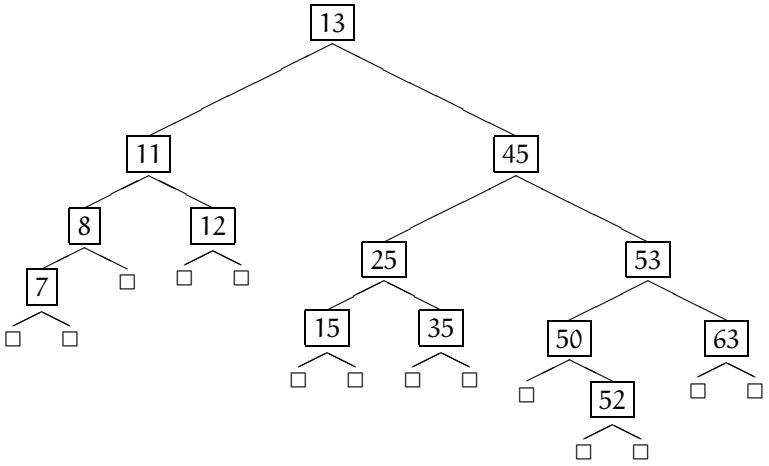
Einfügen der 35 :
Mit Doppelrotation (rechts-links) im Teilbaum der 25:



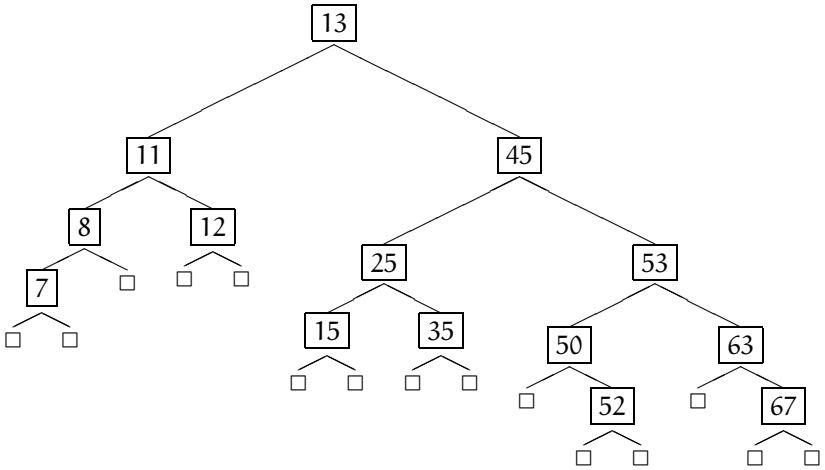
Einfügen der 63 :
Mit Rotation (links) im Teilbaum der 50:



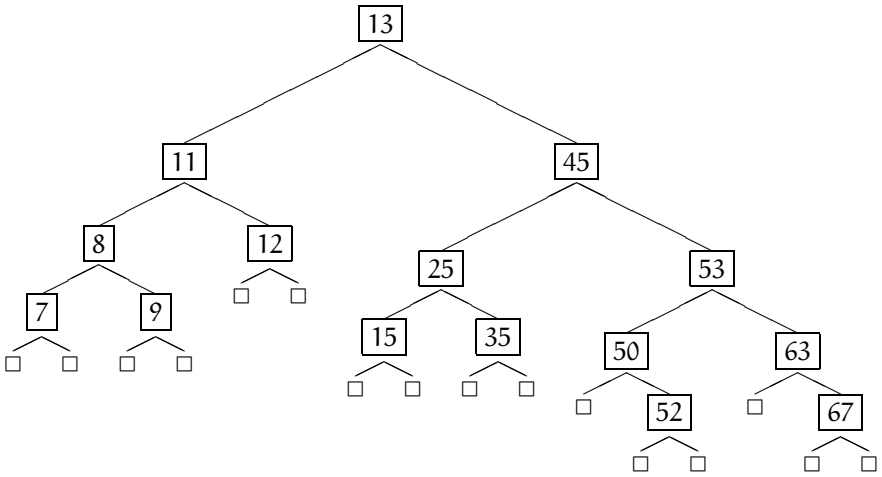
Einfügen der 52 :



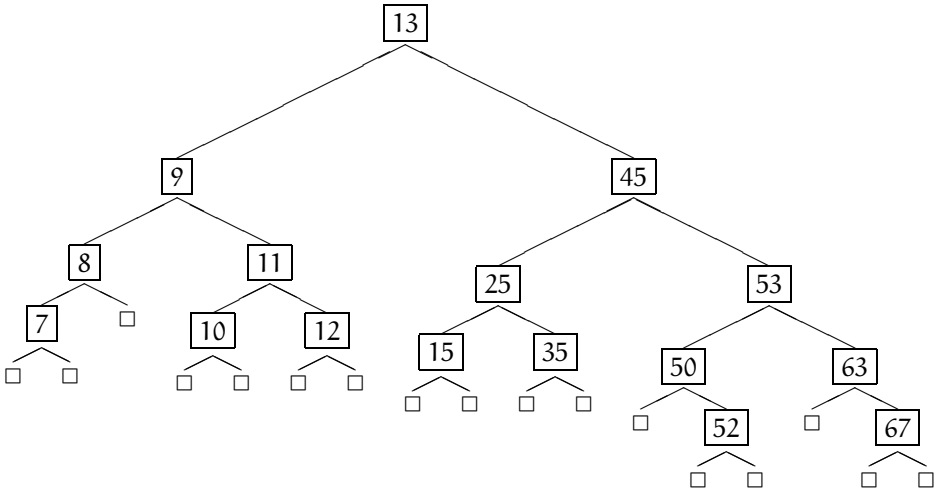
Einfügen der 67 :



Einfügen der 9 :



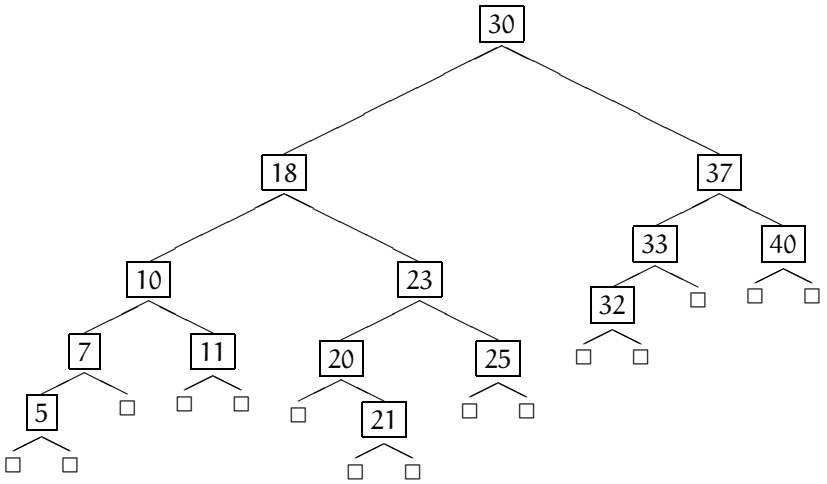
Einfügen der 10 :
Mit Doppelrotation (links-rechts) im Teilbaum der 11:



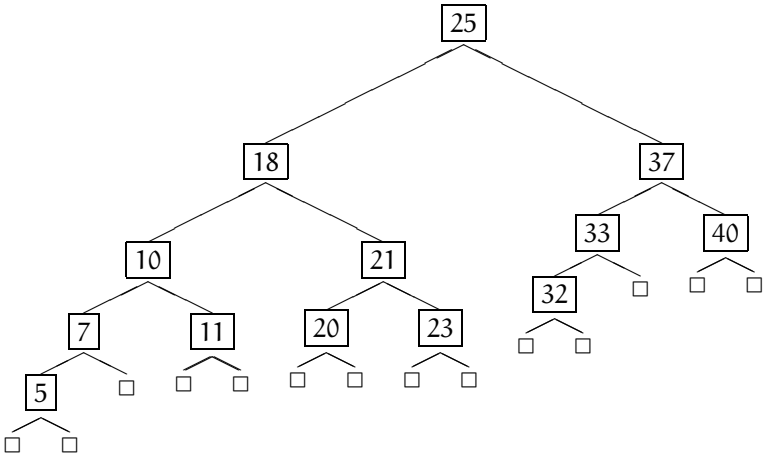
b) Eine der möglichen Folgen entsteht durch Level-Order-Durchlauf des AVL-Baumes:

$$(30, 18, 37, 10, 23, 33, 40, 7, 11, 20, 25, 32, 5, 16, 21)$$

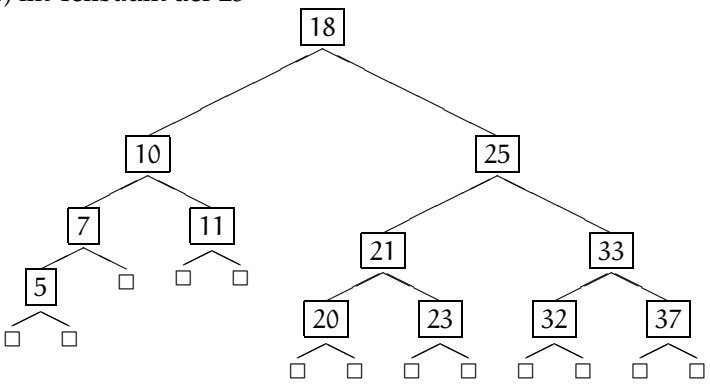
c) Löschen der 16:



Löschen der 30:
Vertauschen mit dem symmetrischen Vorgänger 25
Doppelrotation (links-rechts) im Teilbaum der 23



Löschen der 40:
Rotation (rechts) im Teilbaum der 37
Rotation (rechts) im Teilbaum der 25



Löschen der 11:
Rotation (rechts) im Teilbaum der 10