Übungsblatt 4, Algorithmen und Datenstrukturen

Aufgabe 26

	0	1	2	3	4	5	6	7	8	9	10	11	12
Lineares Sondieren			93			109	97	111	123	124	136		
Quadratisches Sondieren			93	136		109	97	111		123	124		
Double Hashing	124		93			109	97	111			136	123	

Aufgabe 27

	0	1	2	3	4	5	6	7	8	9	10
Tabelle ist leer											
27 wird einfach eingefügt						27					
81 wird einfach eingefügt					81	27					
19 wird einfach eingefügt					81	27			19		
$j = 4$											
$j_1 = (j + (70 \bmod 5) + 1) \bmod 11 = 5$											
$j_2 = (j + (81 \bmod 5) + 1) \bmod 11 = 6$					70	27	81		19		
35 wird einfach eingefügt			35		70	27	81		19		
$j = 4$											
$j_1 = (j + (4 \bmod 5) + 1) \bmod 11 = 9$											
$j_2 = (j + (70 \bmod 5) + 1) \bmod 11 = 5$											
$j = (4 + 4 + 1) \bmod 11 = 9$✓			35		70	27	81		19	4	
$j = 9$											
$j_1 = (j + (75 \bmod 5) + 1) \bmod 11 = 10$✓			35		70	27	81		19	4	75
$j = 6$											
$j_1 = (j + (127 \bmod 5) + 1) \bmod 11 = 9$											
$j_2 = (j + (81 \bmod 5) + 1) \bmod 11 = 8$											
$j = 9$											
$j_1 = (j + (127 \bmod 5) + 1) \bmod 11 = 1$✓		127	35		70	27	81		19	4	75

Aufgabe 28

- $26209, 3517, 4127, 3499$ sind Primzahlen und daher prinzipiell eine gute Wahl, 3499 kann aber gar nicht verwendet werden, weil es kleiner als 3500 ist.
- 4096 ist eine schlechte Wahl, da es sich um eine Potenz von 2 handelt: $2^{12} = 4096$ D.h. nur die letzten 12 Binärziffern spielen dann eine Rolle.

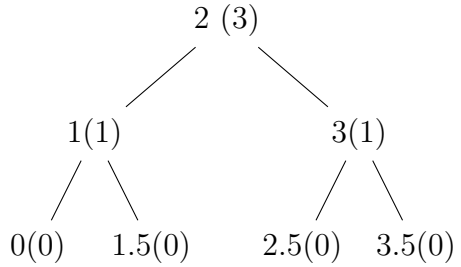
Für 4127:

$h_1(k) = k \bmod 4127$
$h_2(k) = k \bmod 1709 + 1$

Die Folge $(h_1 + i * h_2) \bmod 4127$ muss abhängig von i alle Schlüssen erreichen können, d.h. h_2 nicht 0 sein oder 4127 teilen.

Aufgabe 29

Jeder Knoten speichert sich die Anzahl an Knoten in der linken Hälfte:



In Klammern steht die zusätzliche Information, im Pseudocode heißt sie *leftweight*.

```
FIND-NTH(i, root):
  if root.leftweight == i:
    return root.value
  elif root.leftweight < i:
    return FIND-NTH(i - root.leftweight - 1, root.right)
  elif root.leftweight > i:
    return FIND-NTH(i, root.left)
```

Laufzeit

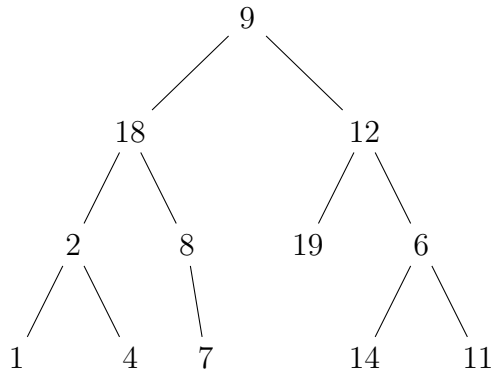
Der Algorithmus liegt in $O(\log n)$, da es nur eine Rekursion nach unten gibt und diese die Problemgröße bei einem balancierten Baum halbiert.

Korrektheit

Der Wert *leftweight* eines Knotens i entspricht genau der Position in der Sortierreihenfolge des Baumes mit der Wurzel i. Wenn wir also an der Wurzel beginnen und genau unseren erwarteten Wert finden, sind wir fertig.

Andernfalls ist unser gesuchter Wert im linken oder rechten Teilbaum. Wenn man bedenkt dass die Sortierreihenfolge des linken (rechten) Teilbaums die erste (zweite) Hälfte der gesamten Sortierreihenfolge darstellt, sollte der Rekursionsschritt klar sein. Er enspricht dem einer binären Suche auf dieser Sortierreihenfolge.

Aufgabe 30



Algorithmus:

```
PRE–IN(preorder, inorder):
  if preorder is empty or inorder is empty:
    return null
  root = (remove first element of pre)
  root_i = (position of root in inorder)

  rv = new node
  rv.value = root
  rv.left = PRE–IN(preorder, inorder[0..root_i])
  rv.right = PRE–IN(preorder, inorder[root_i+1..end])
  return rv
```

Laufzeit

Der Algorithmus läuft in $O(n \log n)$, Argumentation siehe Mergesort (zwei rekursive Aufrufe mit je halber Problemgröße, lineare Suche als Zusatzaufwand pro Rekursion).

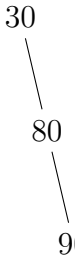
Korrektheit

Das erste Element in der Pre-Order ist offensichtlich der Wurzelknoten, daher wird dieser entfernt und in der In-Order gesucht. Alle Elemente vor dem Wurzelknoten in der In-Order müssen zur linken Hälfte dazugehören, alle Elemente danach zur rechten Hälfte.

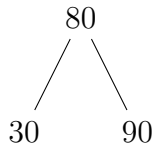
Aufgabe 31

Aufgabe 31a

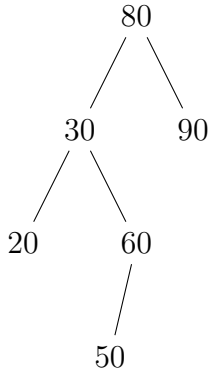
(a) Einfügen von 30 und 80



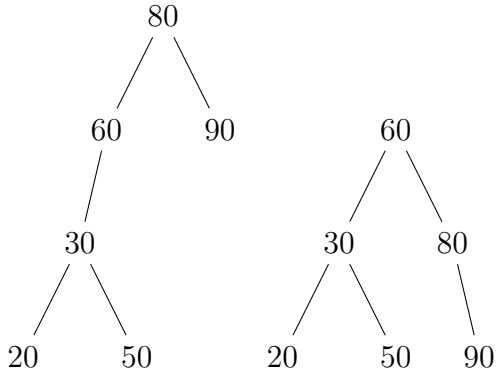
(b) Einfügen von 90



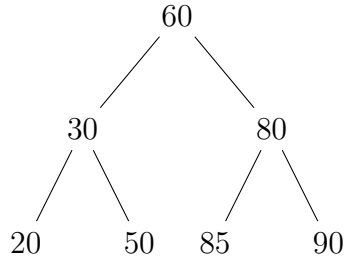
(c) Rotation



(d) Einfügen von 20, 60, 50

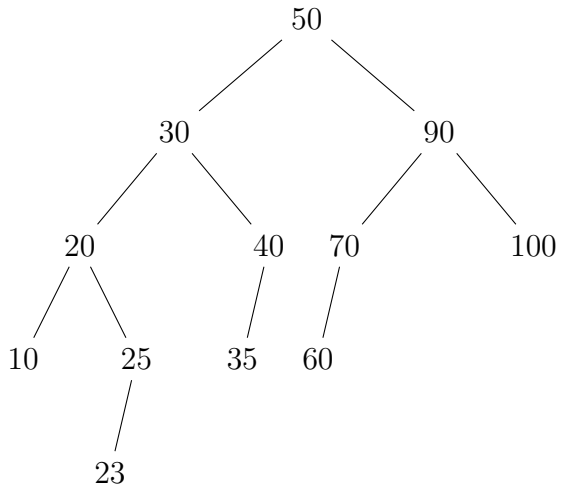


(e) Doppelrotation (Fall 1.2)

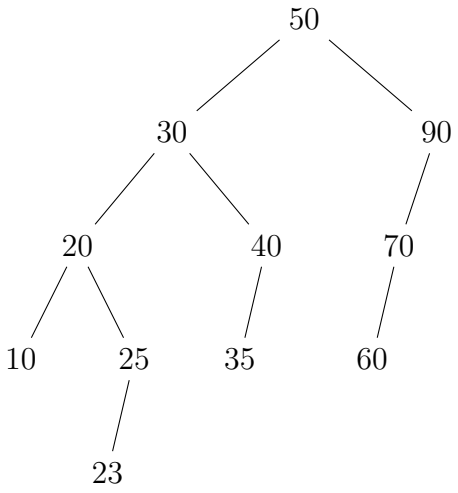


(f) Einfügen von 85

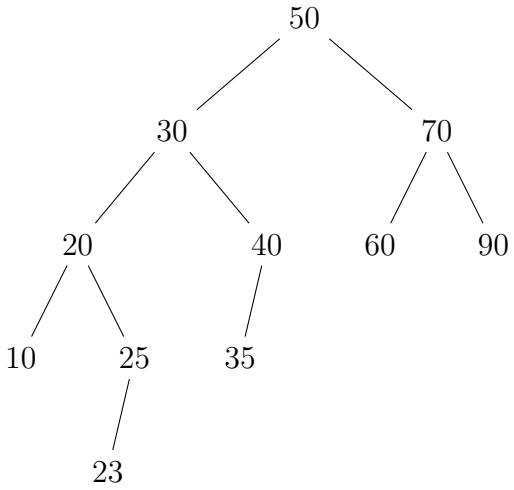
Aufgabe 31b



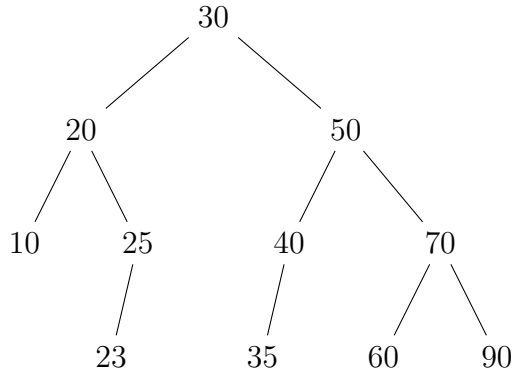
(a) Der gegebene Baum



(b) 100 gelöscht



(c) Rotation des Baumes mit Wurzel 90

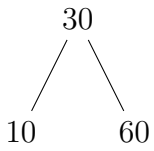


(d) Rotation des gesamten Baumes (Fall 1.1)

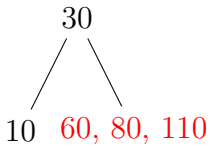
Aufgabe 32

Knoten in rot verletzen Invarianten (zu groß, zu wenige Nachfolger, ...).

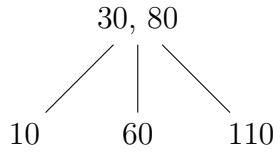
(a) Einfügen von $10, 30, 60$



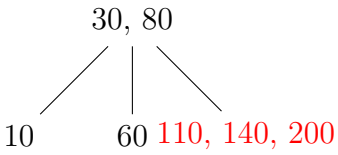
(b) Der Knoten wird gesplittet.



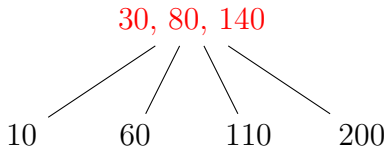
(c) Einfügen von $80, 110$.



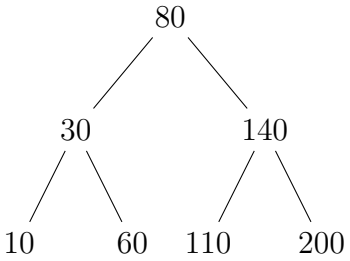
(d) Der Knoten wird gesplittet.



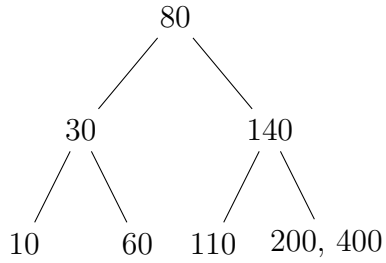
(e) Einfügen von $140, 200$.



(f) Der Knoten wird gesplittet.

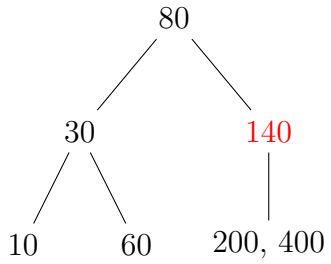


(g) Der Knoten wird gesplittet.

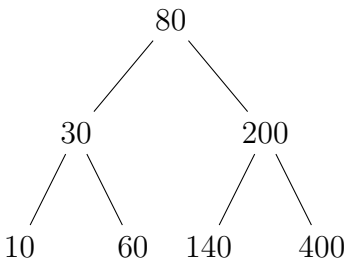


(h) Einfügen von 400

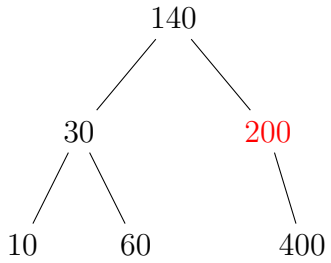
Da diese Sequenz streng monoton ist, ist der entsprechende binäre Suchbaum ein einfacher Pfad, die Suche darin ist $O(n)$.



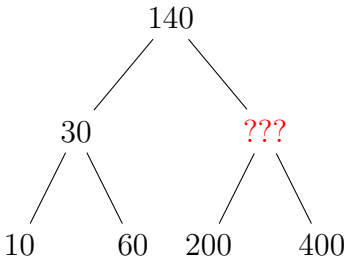
(a) 100 wird gelöscht.



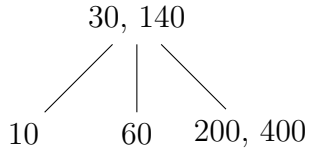
(b) Baum wird rotiert.



(c) 80 wird gelöscht



(d) Baum wird rotiert.



(e) Eine Ebene wird entfernt.