Grundzüge DS & Alg (WS14/15)

Lösungsvorschlag zu Aufgabenblatt 7

Aufgabe 1

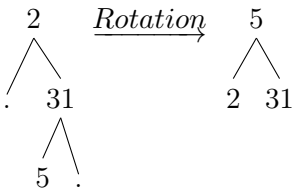
(a) Anmerkung: Der Punkt . in den Bäumen hat keinerlei Bedeutung und ist nur da, um darstellen zu können, was linkes und rechtes Kind eines Elternteils sein soll.

Einfügen von 2, 31, 5, 29, 9, 25, 13, 20, 17

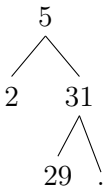
2: 2

31:

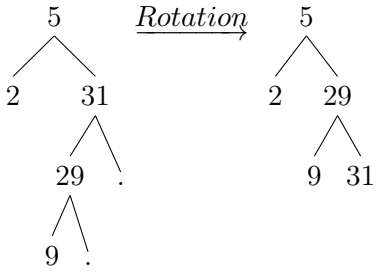
5:



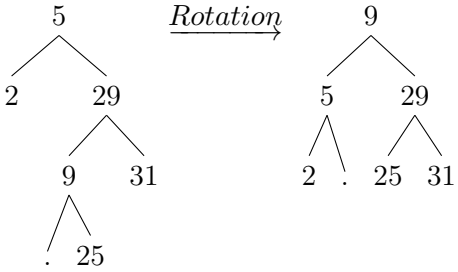
29:



9:



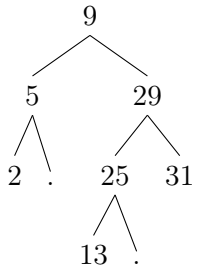
25:



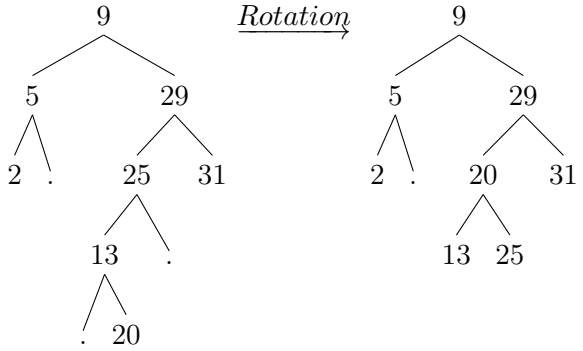
Grundzüge DS & Alg (WS14/15)

Lösungsvorschlag zu Aufgabenblatt 7

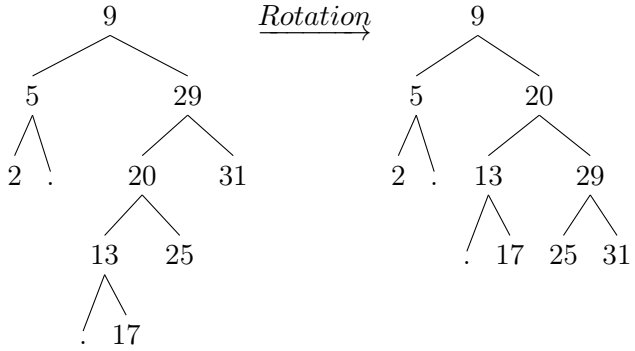
13:



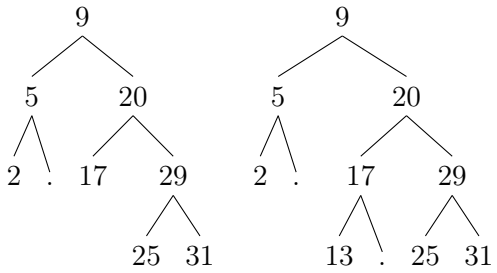
20:



17:



(b) Löschen und Hinzufügen von 13:



Grundzüge DS & Alg (WS14/15)

Lösungsvorschlag zu Aufgabenblatt 7

Aufgabe 2

Bei einem Beinahe-AVL-Baum sind die AVL-Kriterien bei allen Kindern außer bei der Wurzel erfüllt. Dort sind die Höhenunterschiede zwischen linkem und rechtem Teilbaum genau 2.

Algorithmus: BALANCE(KNOTEN x)

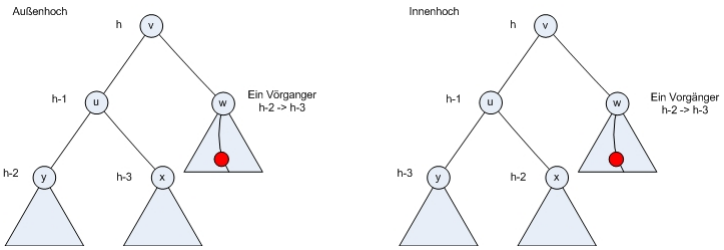
```
1  if(|h(left(X)))-h(right(X))|<2)
2          return; //Baum ist bereits balanciert.
3  KNOTEN S = parent(X);
4  if(left(S)==X)
5          isleft = 1;
6  else
7          isleft = 0;
8  if(h(left(X))==h(right(X)))+2) //Verwende Rechtsrotation.
9          KNOTEN Y = left(X);
10         KNOTEN A = left(Y);
11         KNOTEN D = right(X);
12         if(h(A)==h(X)-2)          //einfache Rotation
13                 KNOTEN B = right(Y);
14                 if (isleft==1)
15                         S.left = Y;
16                 else
17                         S.right = Y;
18                 Y.right = X;
19                 X.left = B;
20                 X.h = max(h(B),h(D))+1;
21                 Y.h = max(h(A),h(X))+1;
22                 S.h = h(Y)+1;
23         else //doppelte Rotation
24                 KNOTEN Z = right(Y);
25                 KNOTEN B = left(Z);
26                 KNOTEN C = right(Z);
27                 if (isleft==1)
28                         S.left = Z;
29                 else
30                         S.right = Z;
31                 Z.left = Y;
32                 Y.right = B;
33                 Z.right = X;
34                 X.left = C;
35                 X.h = max(h(C),h(D))+1;
36                 Y.h = max(h(A),h(B))+1;
37                 Z.h = max(h(X),h(Y))+1;
```

```
38                  S.h = h(Z)+1;
39 else //Verwende Linksrotation.
40          KNOTEN Y = right(X);
41          KNOTEN D = right(Y);
42          KNOTEN A = left(X);
43          if(h(D)==h(X)-2) //einfache Rotation
44                  KNOTEN B = left(Y);
45                  if (isleft==1)
46                          S.left = Y;
47                  else
48                          S.right = Y;
49                  Y.left = X;
50                  X.right = B:
51                  X.h = max(h(A),h(B))+1;
52                  Y.h = max(h(X),h(D))+1;
53                  S.h = h(Y)+1;
54          else //doppelte Rotation
55                  KNOTEN Z = left(Y);
56                  KNOTEN B = left(Z);
57                  KNOTEN C = right(Z);
58                  if (isleft==1)
59                          S.left = Z;
60                  else
61                          S.right = Z;
62                  Z.left = X;
63                  X.right = B;
64                  Z.right = Y;
65                  Y.left = C;
66                  X.h = max(h(A),h(B))+1;
67                  Y.h = max(h(C),h(D))+1;
68                  Z.h = max(h(X),h(Y))+1;
69                  S.h = h(Z)+1;
70 return;
```

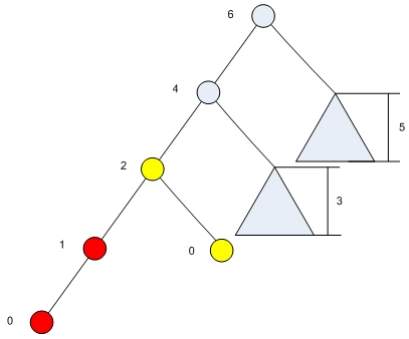
Aufgabe 3

Eine Löschoperation könnte zu einer Verletzung der Höheneigenschaft vom AVL-Baum führen. Sie müssen in einigen Knoten, deren Teilbäume Höhenunterschied 2 haben, auf dem Pfad vom Vater des gelöschten Knoten zur Wurzel auftreten. Daher wird eine (Doppel)Rotation in diesen Knoten ausgeführt, um die AVL-Höheneigenschaften aufrechtzuerhalten.
Wenn jeder Vorgänger des gelöschten Knoten balanciert (repariert) werden muss, kann es zu logarithmisch vielen Rotationen kommen. Denn bei einer Löschoperation kann die Höhe des

Vaters vom gelöschten Knoten (auch seiner Vorgänger nach der Balancierungen) maximal um 1 reduziert werden. Dann passiert die Rotation nur wenn jeder Vorgänger eine kleinere Höhe um 1 als sein Bruder vor dem Löschen hatte und die Teilbäume des Bruders Höhenunterschied 1 hatten.



Wie ein unten Beispiel (die Zahlen sind die entsprechenden Höhen der Teilbäume) gezeigt, wenn einer der beiden roten Knoten gelöscht wird, so kommen 2 Doppelrotationen hinzu, wenn einer der beiden gelben Knoten gelöscht wird so kommt zusätzlich zu den 2 Doppelrotationen noch eine Rechtsrotation hinzu.



Aufgabe 4

Quelle: `www.cs.toronto.edu/~avner/teaching/263/A/2sol.pdf`

Nach Aufgabenstellung sind zwei AVL-Bäume T_1 und T_2 gegeben, wobei alle Werte aus T_1 kleiner als alle Werte aus T_2 sind. Somit ist insbesondere das Maximum aus T_1 kleiner als das Minimum aus T_2.

Sei h_1 die Höhe von T_1 und h_2 die Höhe von T_2. Zudem nehmen wir an, dass $h_1 \geq h_2$ ist. Der umgekehrte Fall funktioniert symmetrisch.

Als nächstes entfernen wir das kleinste Element von T_2 in Zeit $O(h_2)$ und nennen es x. Die Höhe des verbliebenen Baumes T_2' nennen wir h, wobei offensichtlich $h \leq h_2$ gilt.

Als nächstes suchen wir den äußersten rechten Knoten v aus T_1 der Höhe h oder $h+1$ hat. Dies machen wir wie folgt:

$v \leftarrow root(T_1)$
$h' \leftarrow h_1$
while h' > h + 1 **do**
  $v \leftarrow rightchild(v)$
  $h' \leftarrow v.height$
end while

Dies benötigt $O(h_1)$ Zeit. Zudem sei u das Elter von v und T_v der Teilbaum, der v als Wurzel hat.

Nun bilden wir einen neuen Baum T_x mit Wurzel x (der aus T_2 entfernte Knoten). T_2' wird der rechte Teilbaum und T_v der linke Teilbaum. Dies ist ein gültiger AVL-Baum, da $h \leq h' \leq h+1$ gilt und somit die Bedingung des maximalen Höhenunterschieds nicht verletzt werden kann. Zudem sind alle Werte des rechten Teilbaums größer als x, da x das kleinste Element aus T_2 war und alle Werte aus dem linken Teilbaum kleiner als x, da diese aus T_1 stammen und das Maximum aus T_1 kleiner war als das Minimum aus T_2. Die Höhe dieses neu erstellten Baumes ist $h'+1$ (denn $h' \geq h$ und $+1$ wegen der Wurzel x). T_x wird nun das rechte Kind von u. Diese Konstruktionen laufen alle in konstanter Zeit ab.

Nun wandern wir den Baum T_1 beginnend bei u nach oben. Dabei aktualisieren wir die Höhen und führen gegebenenfalls Rotationen durch. Dies funktioniert genau wie in der INSERT-Operation und benötigt $O(h_1)$ Zeit.

Da sowohl h_1 als auch $h_2 \in O(\log n)$ ist, ist auch die Gesamtlaufzeit $O(\log n)$, wobei n = $|A_1 \cup A_2|$ ist.

Sonderpunkteaufgabe

- In einem AVL-Baum gilt immer: für jeden Knoten v mit Vaterknoten w gilt: höhe(w) - höhe(v) $\in$ {1,2}. Mit $rk(knoten) = hoehe(knoten) - 1$ ergibt sich die s-Baum Bedingung.
  Jedoch ist nicht jeder s-Baum ein AVL-Baum, da im s-Baum beide Kinder einen Unterschied von 2 haben können, im AVL Baum aber mindestens ein Kind nur einen Unterschied von 1 haben kann.

- Wir betrachten die minimale Anzahl Knoten eines Baumes mit Rang h, f_h:

$f_0 = 1 \wedge f_1 = 2$

$$\begin{aligned} f_h &= 1 + 2f_{h-2} \\ &= 1 + 2(1 + 2f_{h-2\cdot 2}) \\ &= 1 + 2(1 + 2(1 + 2f_{h-2\cdot 3})) \\ &= \left(\sum_{i=0}^{k-1} 2^i\right) + 2^k f_{h-k\cdot 2} \end{aligned}$$

- Fall $h \equiv 0 \bmod 2$ Mit $k = \frac{h}{2}$ ergibt sich:

$$\begin{aligned} f_h &= \left(\sum_{i=0}^{\frac{h}{2}-1} 2^i\right) + 2^{\frac{h}{2}} \cdot f_0 \\ &= \frac{2^{\frac{h}{2}} - 1}{2-1} + 2^{\frac{h}{2}} \cdot 1 \\ &= 2^{\frac{h}{2}} - 1 + 2^{\frac{h}{2}} \\ &= 2^{\frac{h}{2}+1} - 1 \end{aligned}$$

- Fall $h \equiv 1$ Mit $k = \lfloor \frac{h}{2} \rfloor$ ergibt sich:

$$\begin{aligned} f_h &= \left(\sum_{i=0}^{\frac{h-1}{2}-1} 2^i\right) + 2^{\frac{h-1}{2}} \cdot f_1 \\ &= \frac{2^{\frac{h-1}{2}} - 1}{2-1} + 2^{\frac{h-1}{2}} \cdot 2 \\ &= 2^{\frac{h-1}{2}} - 1 + 2^{\frac{h-1}{2}} \cdot 2 \\ &= 3 \cdot 2^{\frac{h-1}{2}} - 1 \end{aligned}$$

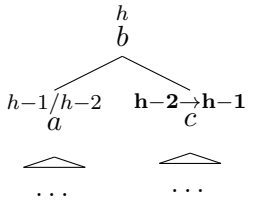
Da die minimale Anzahl Knoten exponentiell im Rang ist, ist der maximale Rang eines solchen Baumes logarithmisch in der Anzahl seiner Knoten.
Da der Rang höchstens um Faktor 2 von der Höhe abweicht, ist somit auch die Höhe logarithmisch in der Anzahl der Knoten.

- Die Idee ist analog zu AVL-Bäumen.
  insert() fügt ein Element wie in einem Binärbaum ein, weißt dem Element den Rang 0 zu und rebalanciert danach. $rebalanceAfterInsert(nodeb)$ unterscheidet folgende 5 Fälle, in denen sich die Höhe des rechten Unterbaumes erhöht hat und deren 5 symmetrische Fälle. Für die Fallerkennung werden nichtexistierenden Knoten der Rang (rk-Wert) -1 zugewiesen.

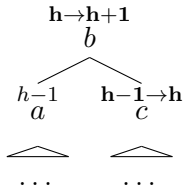
Grundzüge DS & Alg (WS14/15)

Lösungsvorschlag zu Aufgabenblatt 7

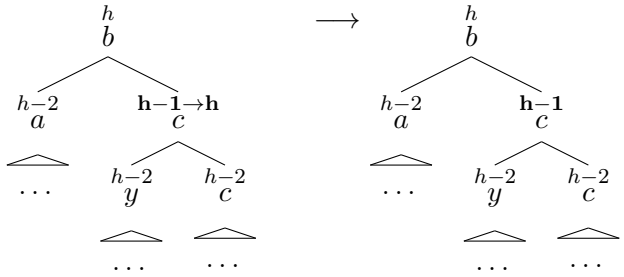
1.



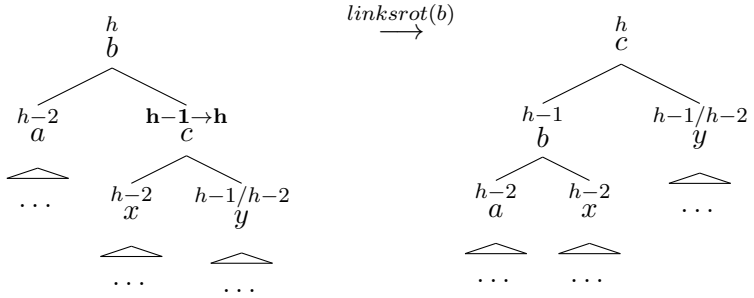
2.



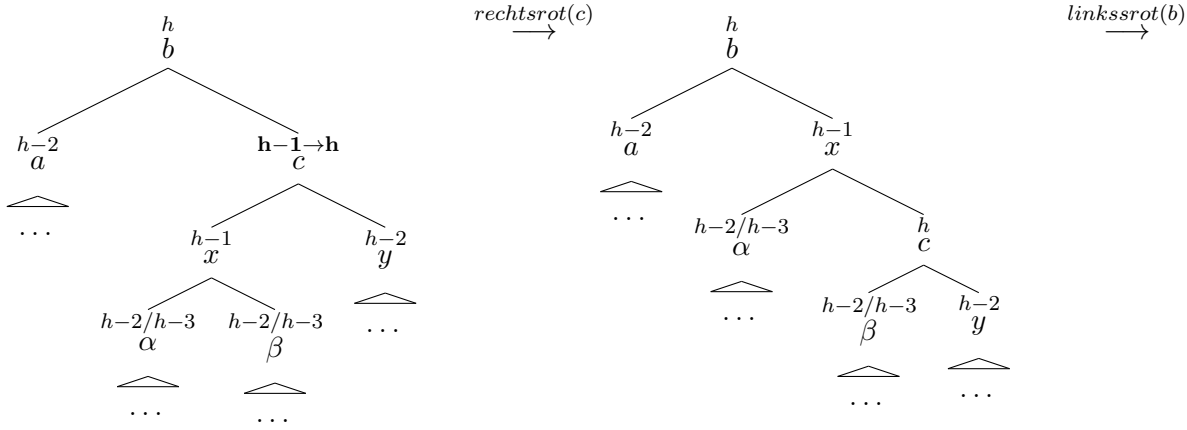
3.

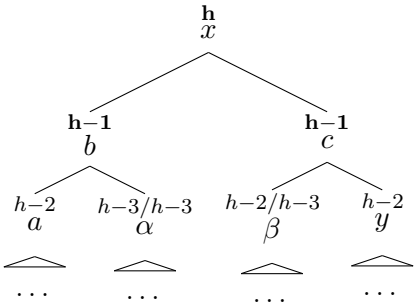


4.



5.

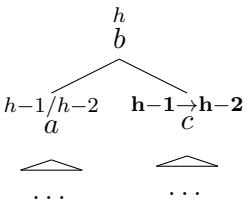




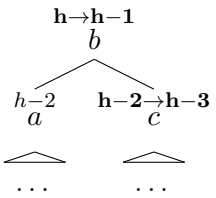
In Fall 5 ist der Zwischenzustand auch ein ungültiger Baum (zwischen x/c). Für Insert kann der Fall mit $rk(a) = h-2,\ rk(b) = h,\ rk(c) = h-1 \to h,\ rk(x) = h-1,\ rk(y) = h-1$ nicht auftreten, da ein solcher die Invarianten verletzender Baum weder durch Fall 2 (der einzige Fall mit Rekursion) noch den Rekursionbeginn in *insert* erzeugt werden kann.

$Delete(x)$ sucht ebenso wie in AVL-Bäumen das nächstkleinere (oder nächstgrößere) Element im Unterbaum, dessen Wurzel x ist und rebalanciert danach. $rebalanceAfterDelete(nodeb)$ unterscheidet dabei die folgenden 5 Fälle:

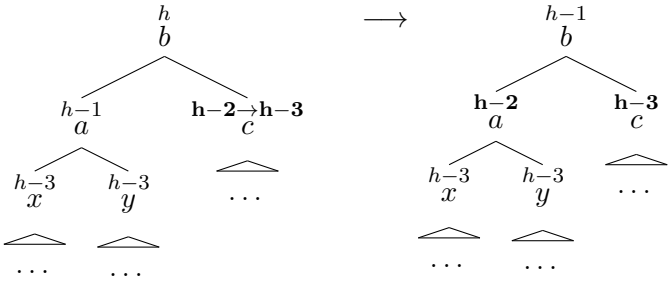
1.



2.



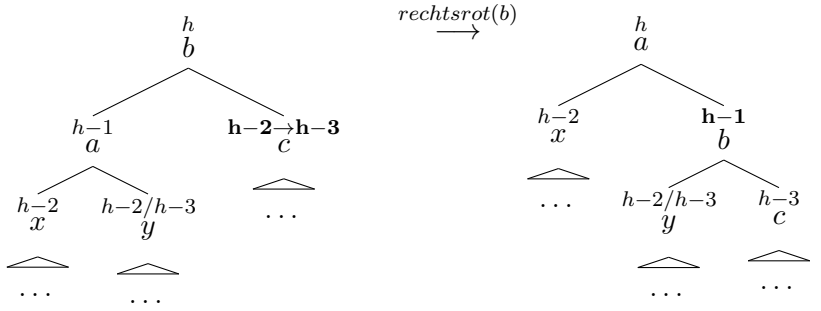
3.



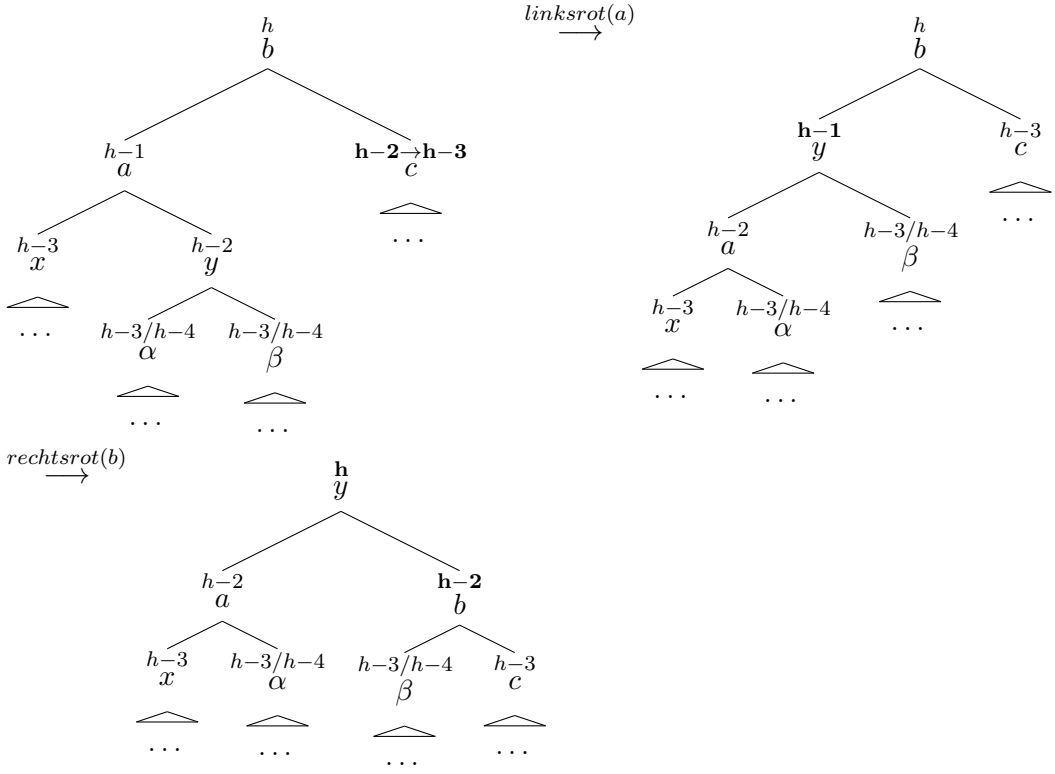
Grundzüge DS & Alg (WS14/15)

Lösungsvorschlag zu Aufgabenblatt 7

4.



5.



In Fall 5 ist der Zwischenzustand auch ein ungültiger Baum (zwischen x/c/y bei insert und β und y beim delete).

In Fall 2 rekurrieren $rebalanceAfterInsert$ und $rebalanceAfterDelete$ jeweils (sofern b noch nicht die Wurzel ist), da sich in diesem (und nur in diesem) Fall der rk-Wert des betrachteten Unterbaumes ändert. In allen anderen Fällen wird die Invariante wiederhergestellt, ohne dass sich dadurch die Höhe des Baumes ändert, und damit kann *rebalance* dann terminieren.

Da alle Fälle, in denen eine Rotation oder Doppelrotation auftritt (Fälle 3,4,5) terminieren, wird höchstenes eine (Doppel-)Rotation pro Aufruf von $rebalanceAfter\{insert|delete\}$ und damit pro Aufruf von *insert* bzw. *delete* ausgeführt.

Da die Erkennung des korrekten Falls in konstanter Zeit funktioniert, ergibt sich die Laufzeit zu je $O(log\ n)$ offensichtlich.