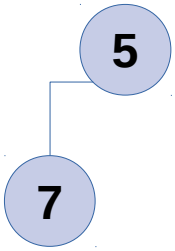
**1.1 Heap**

a) Zeichnen sie den (min)-heap, der durch Einfügen von 5, 7, 1, 6 entsteht.

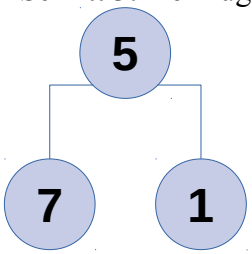
Min Heap = Eigenschaft, dass die Schlüssel der Kinder eines Knotens stets größer als der Schlüssel ihres Vaters sind (Heap-Bedingung).

Schritt 1: 5 einfügen

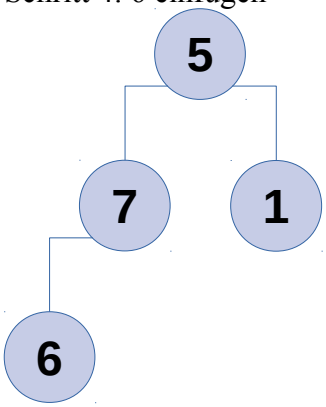

Schritt 2: 7 einfügen



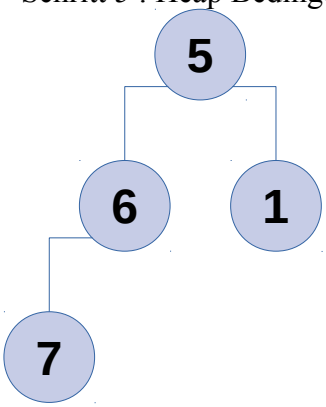

Schritt 3: 1 einfügen



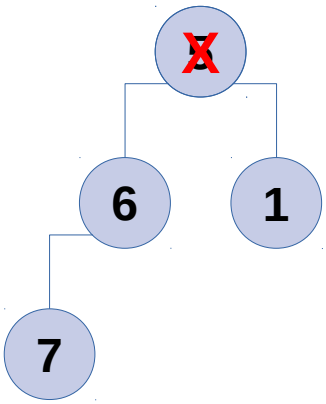

Schritt 4: 6 einfügen

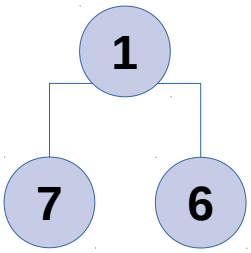



Schritt 5 : Heap Bedingung verletzt! Umschichten




b) Löschen sie ein Element aus dem Heap (pop()) und zeichnen Sie den neuen Heap.



**1.2 Wie funktioniert heap sort?** Implementieren sie heap sort mit Hilfe der bekannten Datenstrukturen. Argumentieren Sie warum der Algorithmus worst case Komplexität O(n log n) besitzt.

**1.3 Wie kann man die Ausführungszeit eines Programms oder einer Funktion messen? Welche Alternativen gibt es? Was sind die Vor- und Nachteile von time(), clock_gettime(). Clock(), rdtsc? Was ist processor time? Was ist wall clock time?**

Ausführungszeit eines Programms messen:

| | Vorteile | Nachteile |
|---|---|---|
| time() | - gibt die Zeit seit 1.1.1970 wider<br>- Zeit in Sekunden | - keine Zeiten kleiner als eine Sekunde |
| clock_gettime() | - Gibt die Zeit wieder, man kann sich über die clock_id eine Uhr aussuchen | |
| clock() | - gibt die Prozessorzeit an die vom Programm verbraucht wird.<br>- Gibt clock_ticks wider | - Systemspezifisch |
| rdtsc() | - gibt die Anzahl der clock-cycles wider die seit dem letzten Reset vergangen sind | |

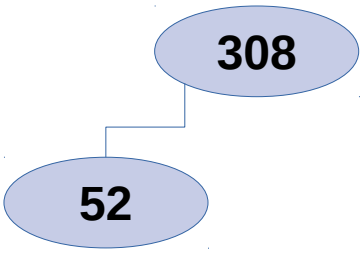
- Processor time ist die Zeitspanne die der Prozessor tatsächlich etwas zu tun hat und nicht wartet oä.
  - Bevorzugt eingesetzt bei Algortihmen ohne Festplattennutzung.
- Mit wall clock time misst man die Gesamtzeit wie mit einer herkömlichen Uhr.
  - Bevorzugt eingesetzt bei Algorithmen mit Festplattennutzung

**1.4 Schreiben sie ein kleines Programm, mit welchem n Zufallszahlen in einen binären Suchbaum eingefügt werden können. Bestimmen Sie die Höhe des resultierenden Baumes (Grafik!). Wiederholen sie diesen Test für n = 10000, 20000, 30000,... und vergleichen Sie die Höhe und die Laufzeit mit einem Ausbalancierten (AVL-)Baum (Grafik!).**

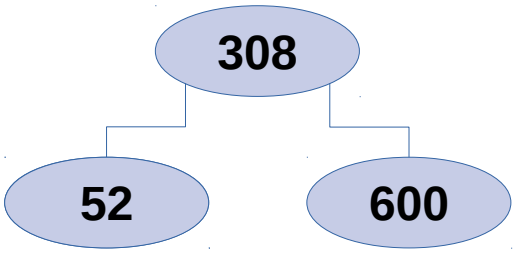
**1.5 Fügen Sie die Schlüssel 308, 52, 600, 987, 51, 4, 222, 412, 999, 577, 578 in einen leeren AVL-Baum ein und zeichnen Sie den Baum nach jeder Einfügeoperation.**

Schritt 1:

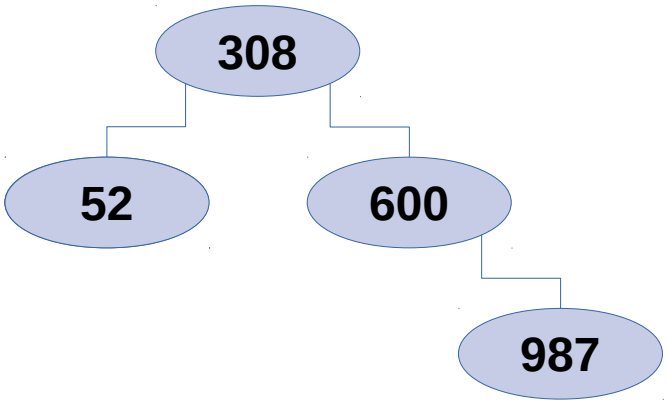

Schritt 2:



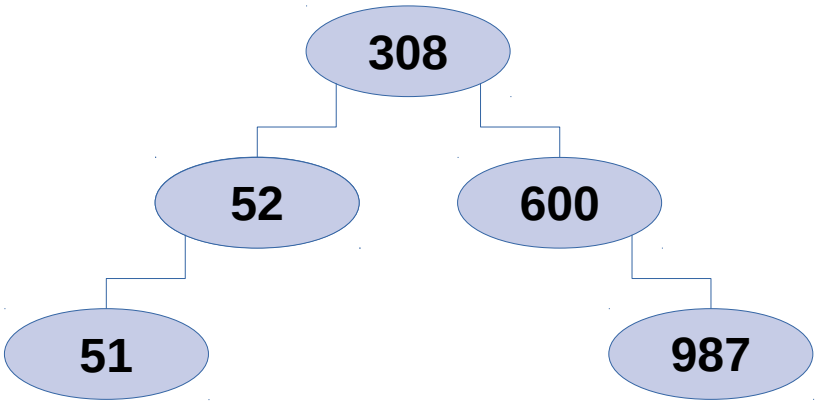

Schritt 3:



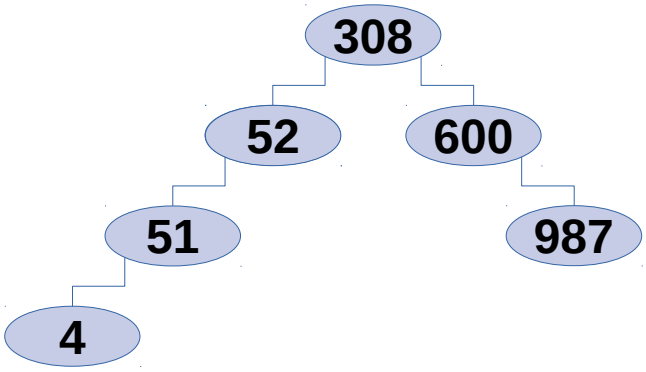

Schritt 4:



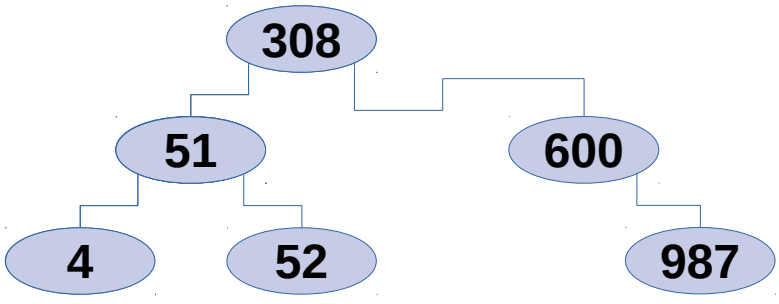

Schritt 5:



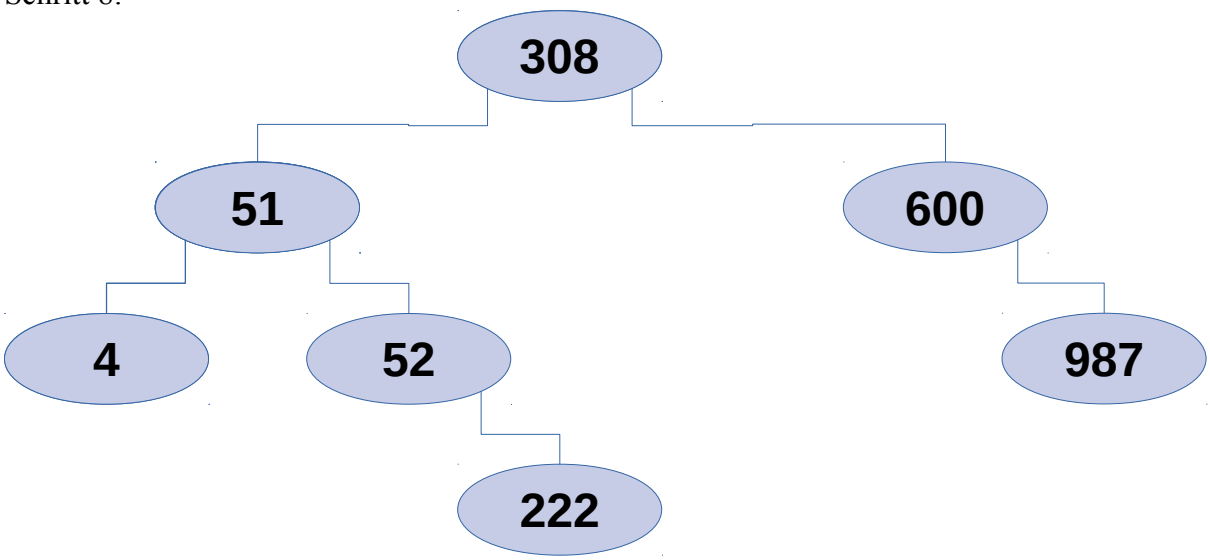

Schritt 6:



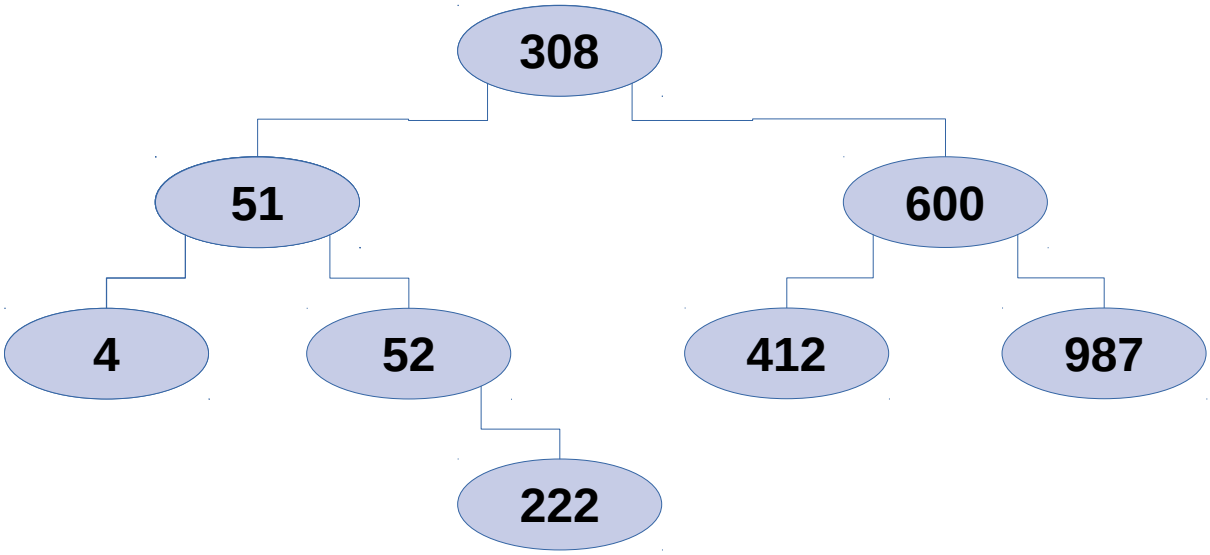

Schritt 7:



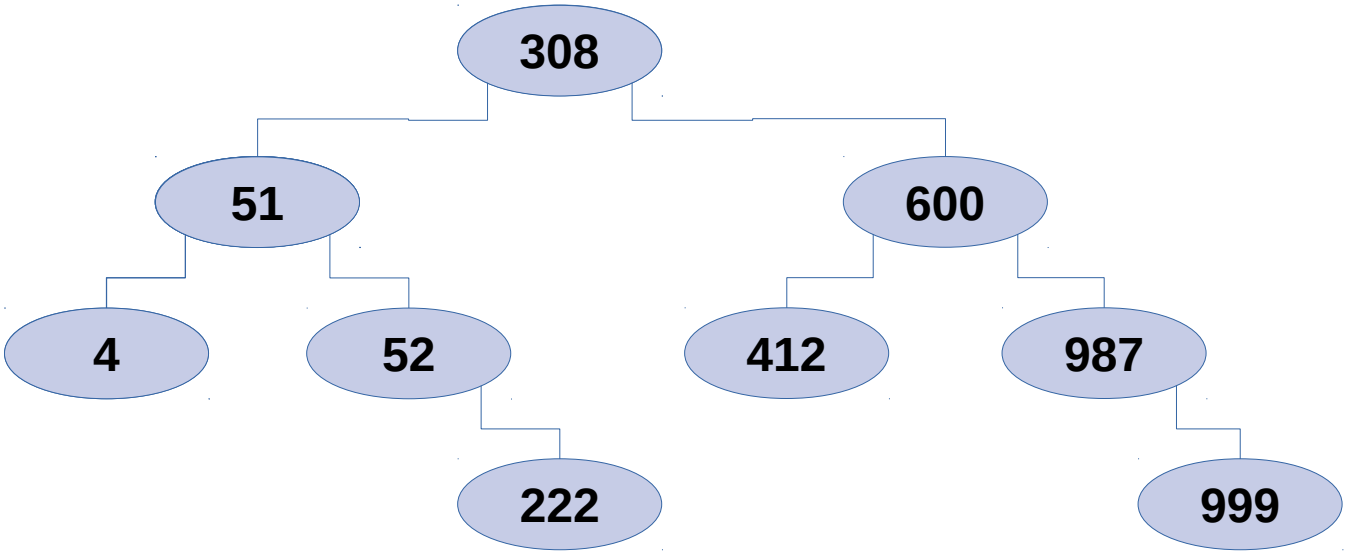
Schritt 8:



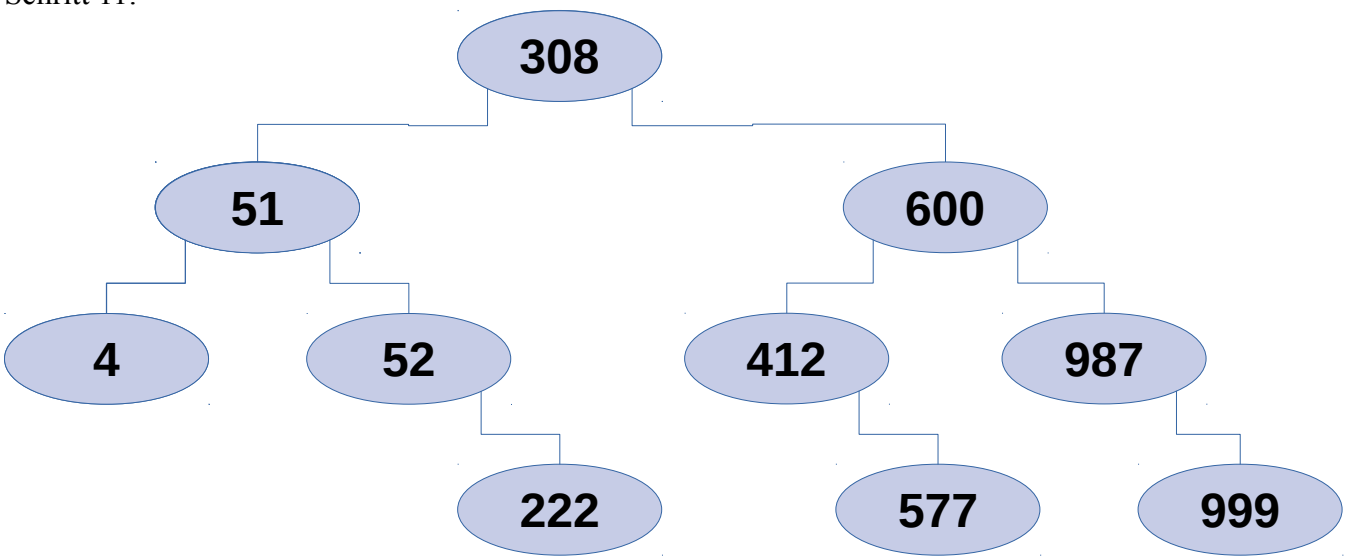

Schritt 9:



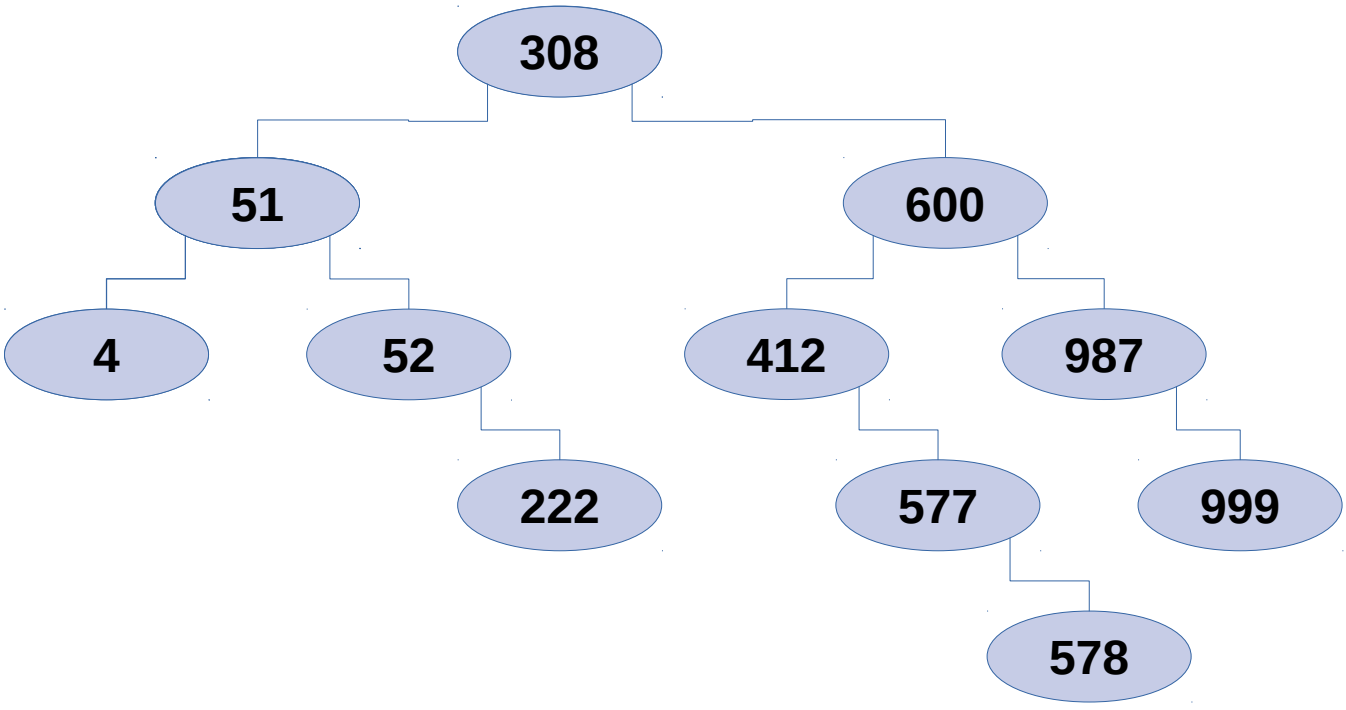

Schritt 10:



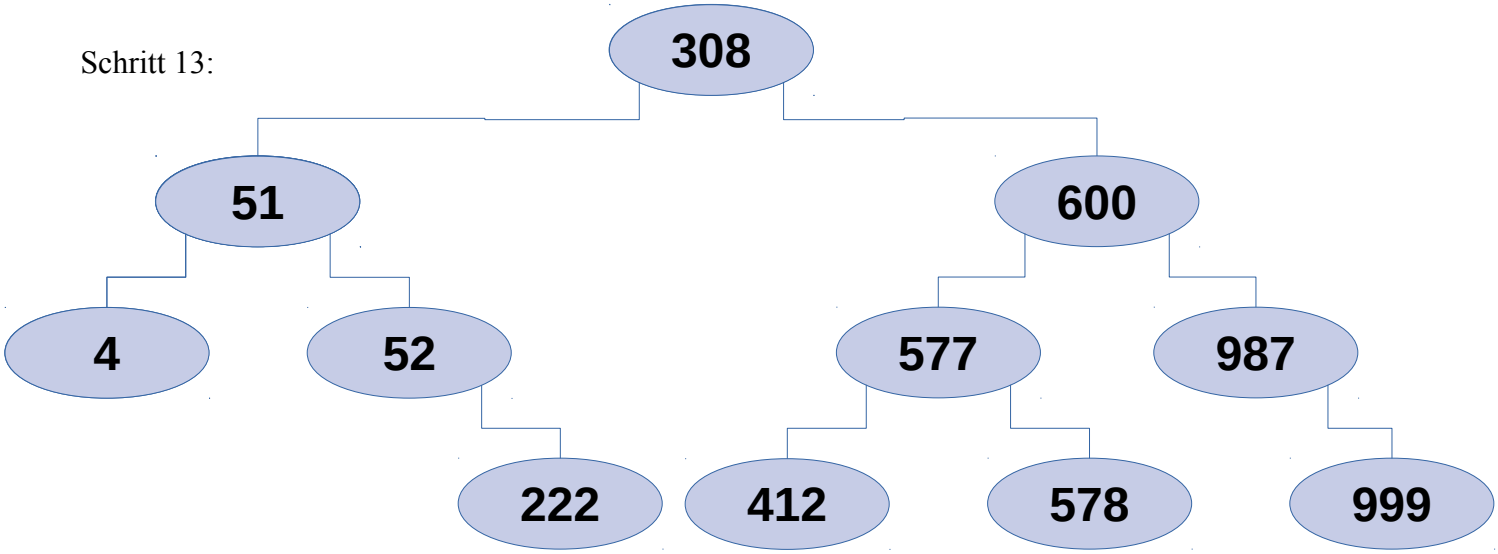
Schritt 11:




Schritt 12:

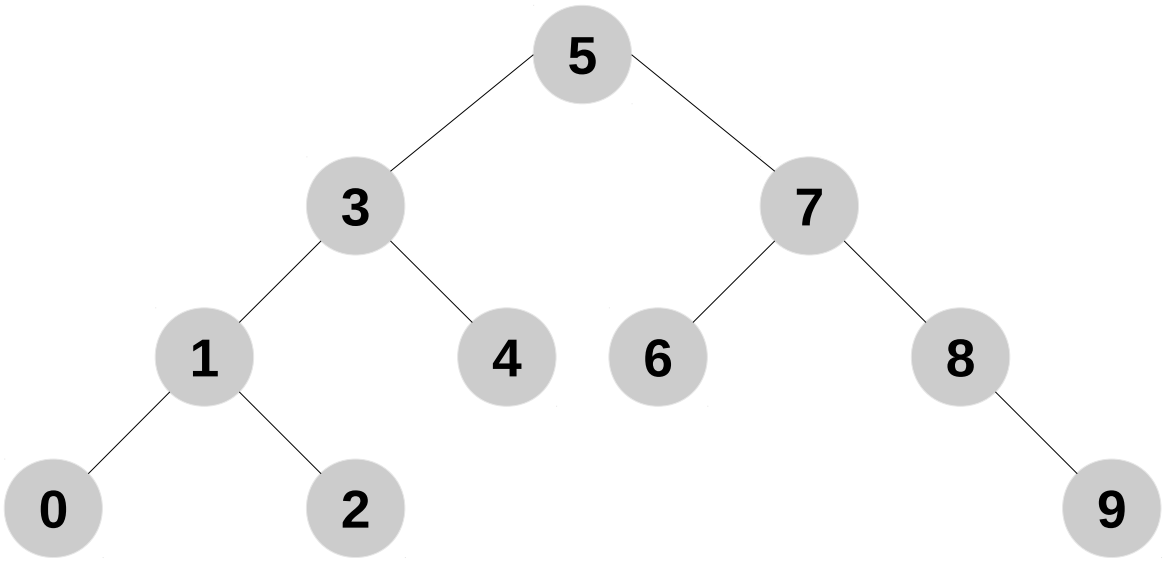



Schritt 13:



**1.6 Geben Sie eine Reihenfolge für das Einfügen der 10 Schlüssel 0, 1, 2,..., 8, 9 in einen (zunächst leeren) AVL-Baum an, sodass keine Rotationen notwendig sind. (Es gibt mehrere Lösungen.)**

5,3,7,4,6,8,1,0,9,2



**1.7 Gegeben sei der folgende AVL-Baum.**
**a) Fügen Sie den Knoten 0 ein (ausgehend vom Baum oben) und balancieren sie den Baum.**
**b) Fügen sie den Knoten 11 ein (ausgehend vom Baum oben) und balancieren sie den Baum.**

**1.8 Der AVLTree aus der Vorlesung hat ein Problem. Was ist die Komplexität? Wie beheben Sie das Problem (Idee)?**

**1.9 AVL-Baum**

a)Welche wesentliche Eigenschaft besitzt ein AVL-Baum?

b) Zeichnen sie den AVL-Baum, der sich durch Einfügen von 9, 13, 5, 7, 1, 8 ergibt. Geben Sie die Gewichte aller Knoten vor und nach eventueller Korrekturoperationen an.

("--> -linker teilbaum + rechter teilbaum")

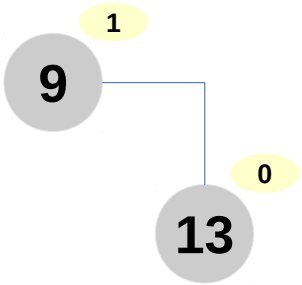
a)

Die wesentliche Eigenschaft eines AVL-Baumes ist, dass sich die "Tiefe" des linken Teilbaumes maximal um 1 von der Tiefe des rechten Teilbaumes unterscheiden darf.

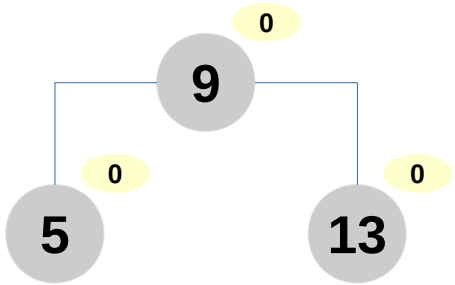
b)

Schritt1:

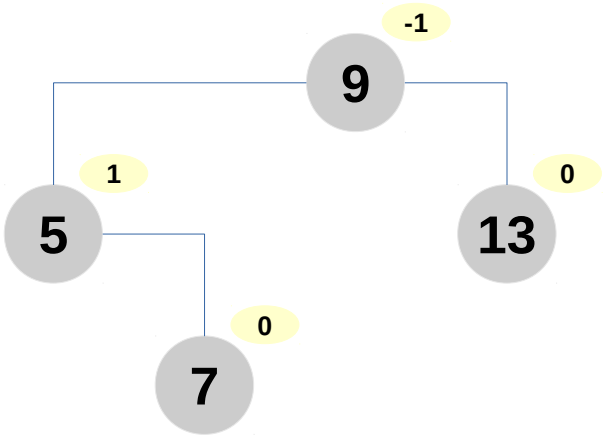

Schritt2:



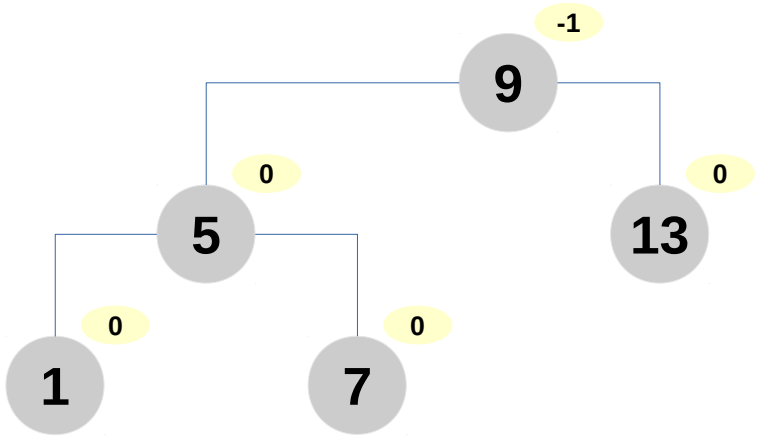

Schritt3:



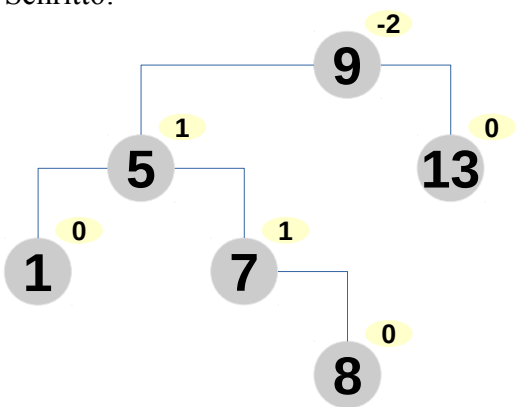

Schritt4:



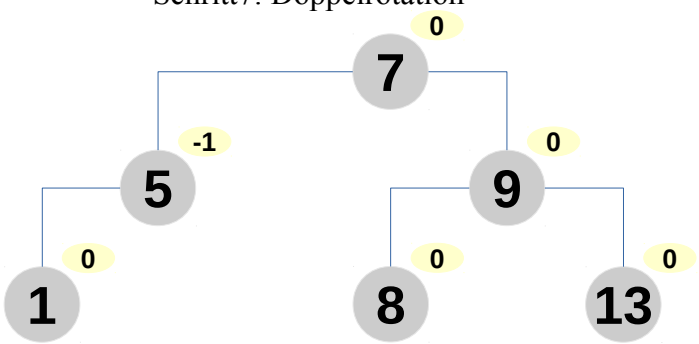

Schritt5:




Schritt6:




Schritt7: Doppelrotation



**10a) Studieren Sie das Paper zu Patricia und beantworten Sie folgende Fragen:**

- **Wie kann die Suche in einem Patricia-Baum implementiert werden? Erklären Sie Ihren (Pseudo-)Code! Welche Daten werden für jeden Knoten gespeichert?**
- **Welche Vor- und Nachteile hat der Patricia-Baum gegenüber einem AVL-Baum bzw. Gegenüber radix-Suche?**

1. Nummeriere Bits der Keys von rechts nach links
2. Speichere im Knoten die Nummer desjenigen Bits, das an diesem Knoten getestet werden muss
3. Konsequenz: auf jedem Pfad durch den Baum von oben nach unten nehmen diese Nummern ab.
4. Suche:
5. Laufe im Baum abwärts, wobei man den Bitindex in jedem Knoten benutzt, um festzustellen, welches Bit im Key zu testen ist.
6. Die Keys in den Knoten werden auf dem Weg im Baum abwärts überhaupt nicht beachtet!
7. Schließlich wird ein aufwärts zeigender Pointer auf einen inneren Knoten vorgefunden -> führe vollständigen Vergleich zwischen gesuchtem Key in jenem inneren Knoten durch.
8. Es ist leicht zu testen, ob ein Zeiger nach oben zeigt, da die Bitindizes in den Knoten (per Defintion) kleiner werden, wenn man sich im Baum abwärts bewegt.

Welche Daten werden für jeden Knoten gespeichert?
Bitindex und Key

Vor- und Nachteile von Patricia gegenüber Avl-Baum/Radix-Search

| **Patricia** | **Radix-Search** | **AVL-Baum** |
|---|---|---|
| - identifies the bits which distinguish the search keys and build them into a data structure<br>- with no surplus nodes<br>- quite well balanced<br>- insensitive to the order in which keys are inserted | - Reasonable worst-case performance without complication of balanced trees<br>- provide an easy way to handle variable length keys<br>- some allow savings in space by storing part of the key within the searh structure<br>- very fast access to the data, competitive with search trees and hashing<br>disadvantages:<br>- biased data can lead to degenerate trees with bad performance (data comprised of characters is biased)<br>- some of the methods can make very inneficient use of space.<br>- difficult to implement in some high-level languages<br>- the number of internal keys can be somewhat larger than the number of keys<br>- tow type of nodes complicate implementation | - can degenerate to a list<br>- insertion or deletion may require complete rearranging the tree to maintain balance |