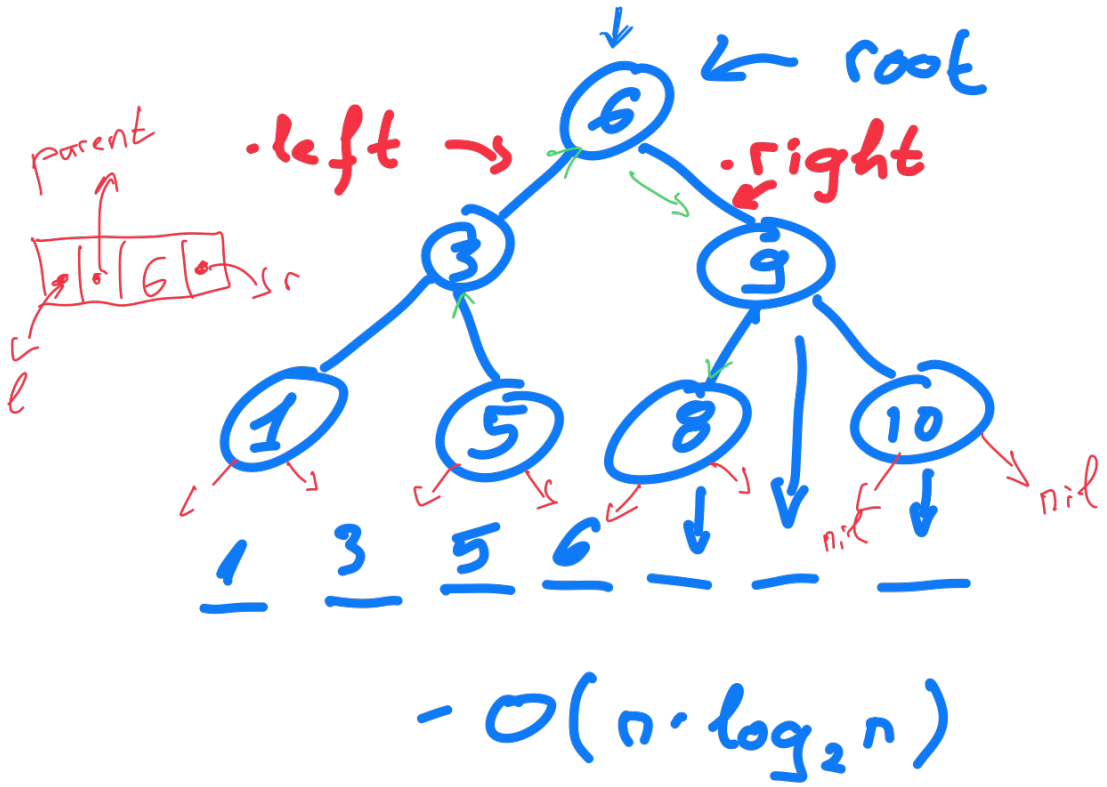


$- O(n \cdot \log_2 n)$

Operationen:

- Search(x)

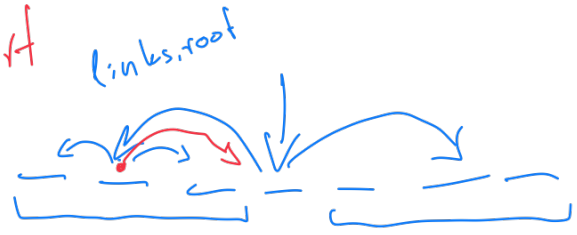
Arr → Baum

balanciert

1. Sortieren.
2. $m = \left\lfloor \frac{arr.length}{2} \right\rfloor$

   root = arr[m]

   root.parent = nil



3. rekursiv links & rechts

   root.left = links.root

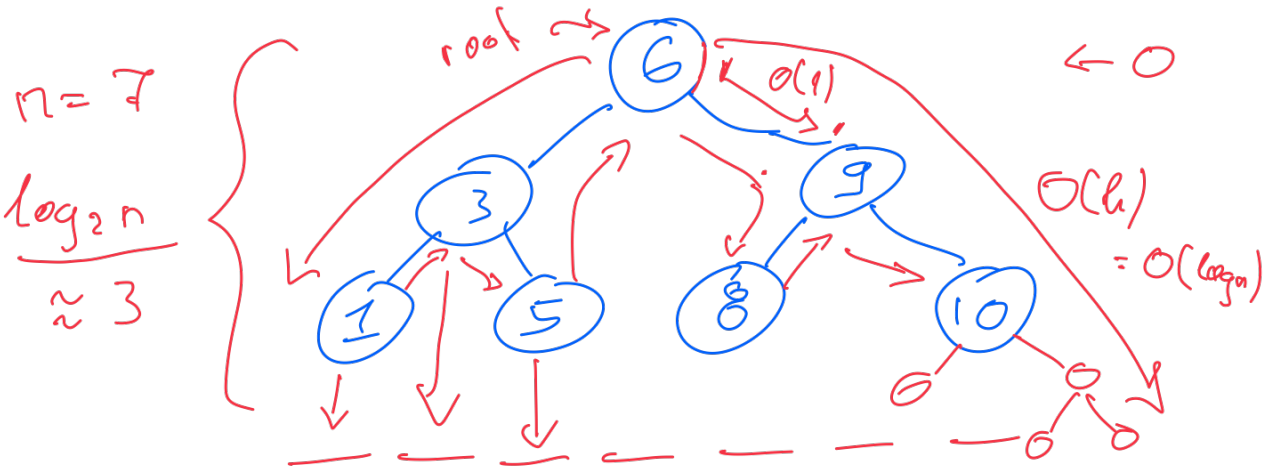
   root.left.parent = root

Alternativ: [] ; Insert(x) wiederholen

Baum → Array

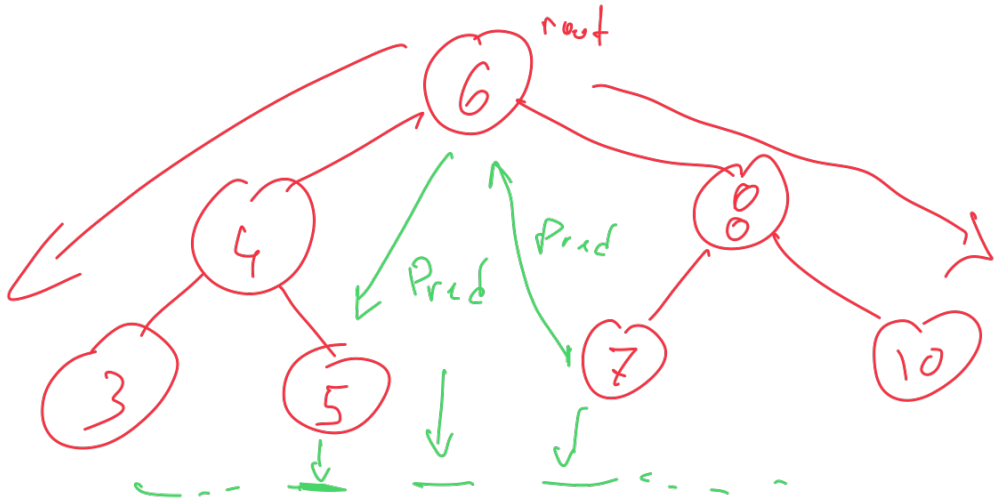
- rekursiv Größe finden (= Anzahl Knoten) n
- Array der Größe n anlegen
- InOrderTreeWalk mit arr[i] = x.key

  (statt "x.key ausgeben")

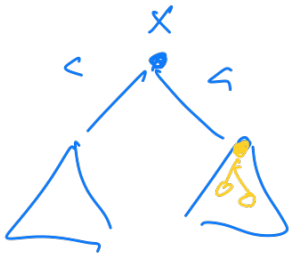


- jedes El. hat 2 Kinder
- => auf Ebene i: 2^i Elemente ($0 \dots h$)

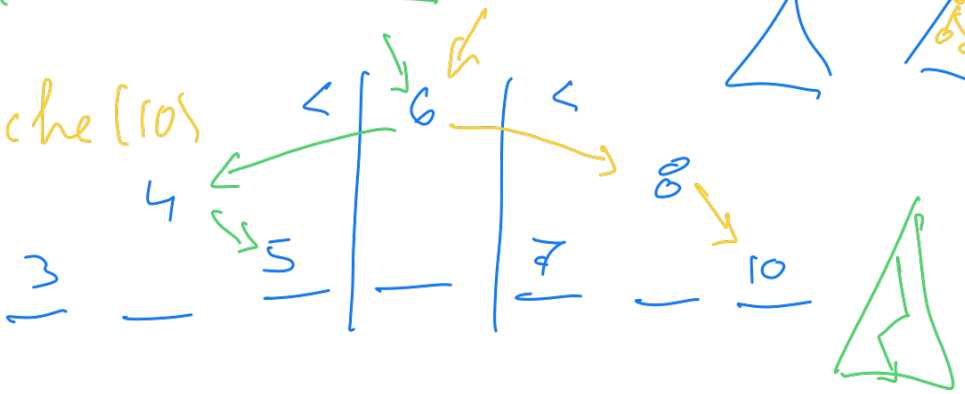
- Ins/Del: $O(h) = O(\log n)$
- Search: $O(h)$ ...
- Pred/Succ: $O(h)$
- Min/Max: $O(h)$
- Baum durch Insert bauen: n Elemente $\cdot\ O(\log n)$ $= O(n \log n)$



- Predecessor (x)

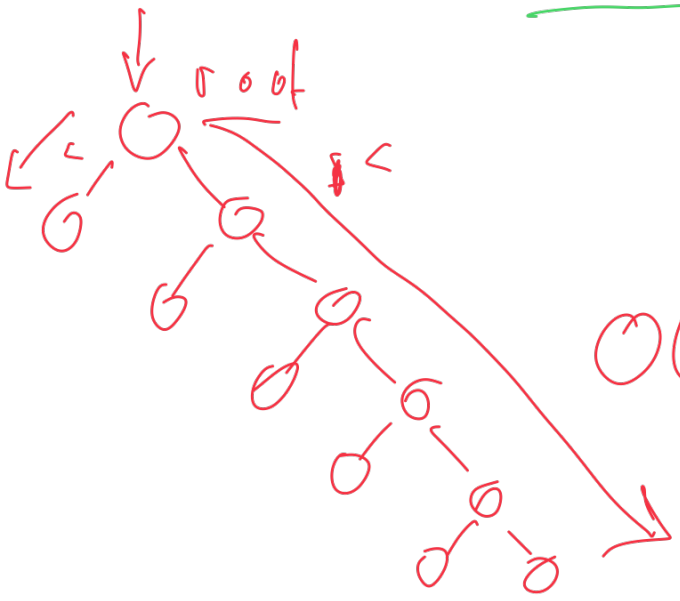


- Suche (10)



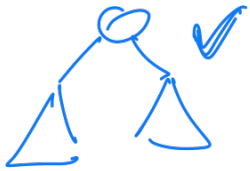
- Suche (5)

Suche: $O(\log n)$

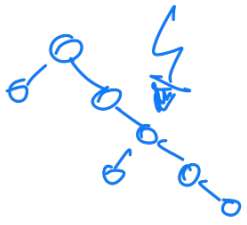


$O(n)$

- balanciert: $\mathcal{O}(\log n) = \mathcal{O}(h)$
- sonst: $\mathcal{O}(h) = \mathcal{O}(n)$



Insert Array halbieren



- alle Operationen: $\mathcal{O}(h)$
- denn: $h :=$ Länge eines längsten Wurzel-Blatt-Pfades