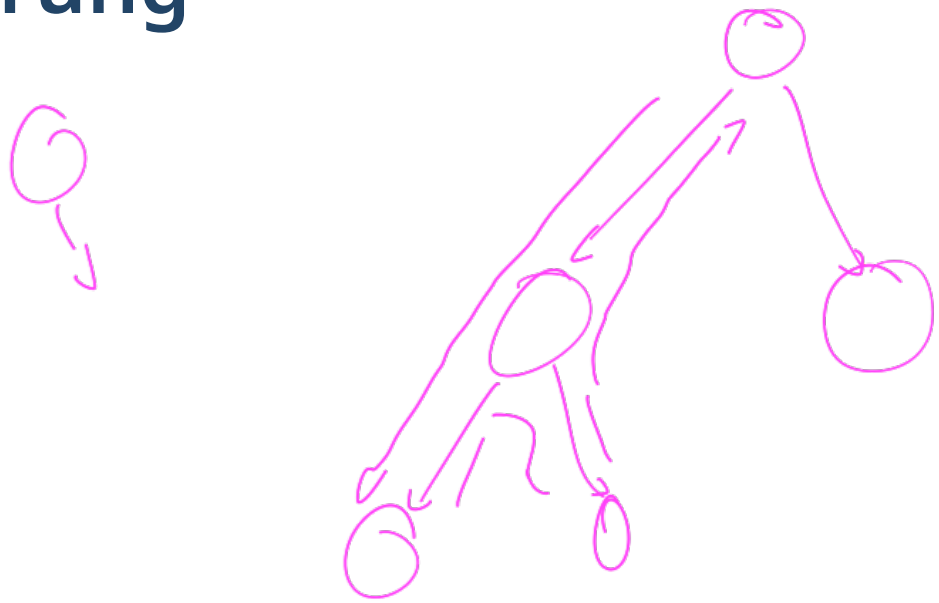
GAD Tutorium Woche 12

Wald und Forstwirtschaft II

Lukas Döllerer - lukas.doellerer@tum.de - lukas-doellerer.de/gad

Aufgabe 11.1 Baumtraversierung



- **PreOrder**:
  Knoten $\longrightarrow$ linkes Kind $\longrightarrow$ rechtes Kind
  = Anordnung nach DFS-Nummer
- **InOrder**:
  linkes Kind $\longrightarrow$ Knoten $\longrightarrow$ rechtes Kind
- **PostOrder**:
  linkes Kind $\longrightarrow$ rechtes Kind $\longrightarrow$ Knoten
  = Anordnung nach DFS-Finish-Nummer

Gesucht:

Algorithmen für `preNext(v)` und `postNext(v)` die für Knoten `v` den folgenden Knoten berechnen + Asymptotische *Worst-Case*-Laufzeit für $k = 1$ und $k = n$.

Lukas Döllerer - lukas.doellerer@tum.de - lukas-doellerer.de/gad

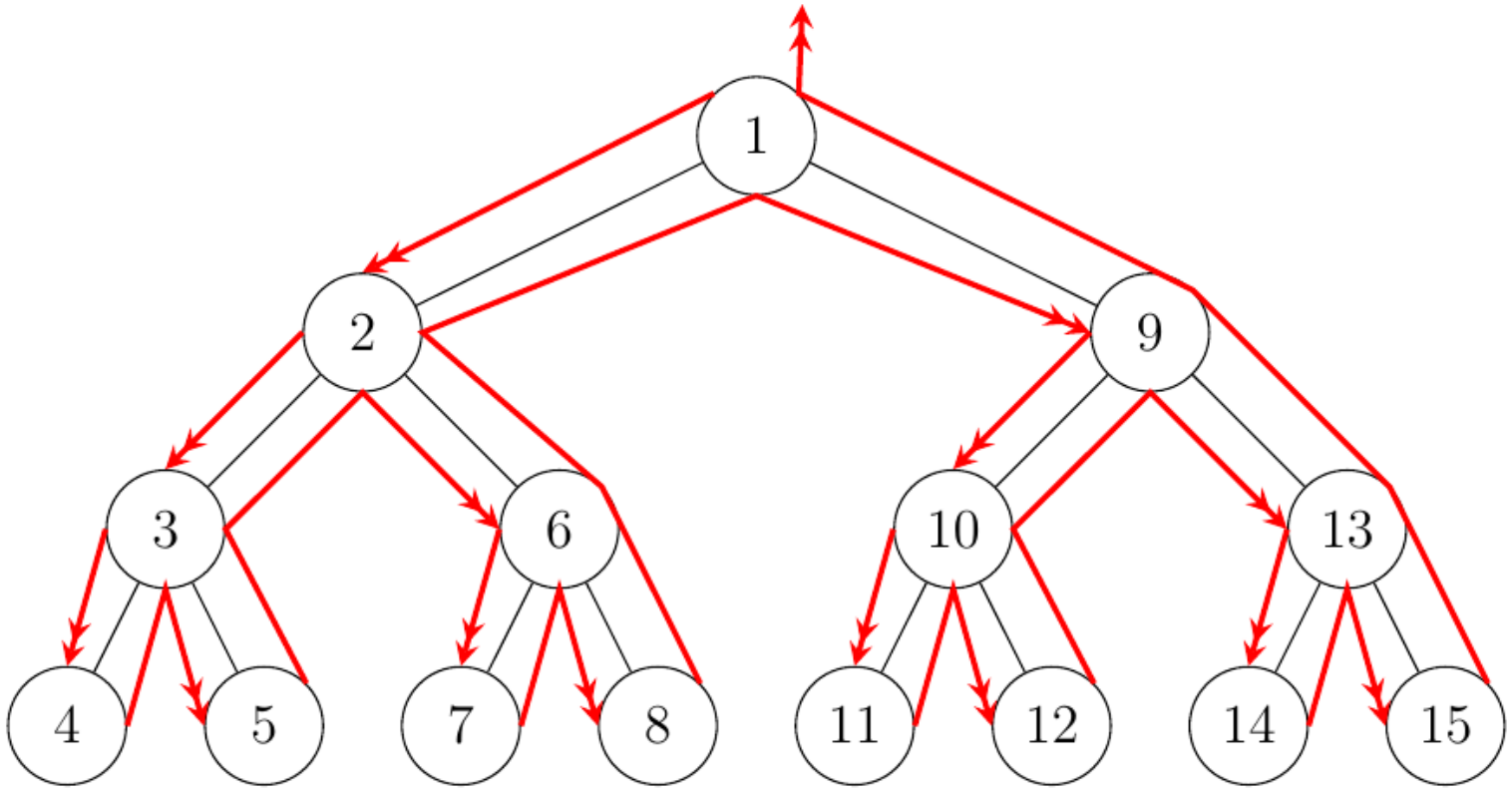
Aufgabe 11.1 Baumtraversierung - `preNext(v)`

```
public Node preNext(Node v) {
    if (hasleftChild(v))
        return leftChild(v);
    else if (hasrightChild(v))
        return rightChild(v);
    else {
        Node p = v;
        Node q;
        while (!isRoot(p)) {
            q = p;
            p = parent(p);
            if (hasrightChild(p) && rightChild(p) != q)
                return rightChild ( p ) ;
        }
        return null;
    }
}
```

Aufgabe 11.1 Baumtraversierung - `postNext(v)`

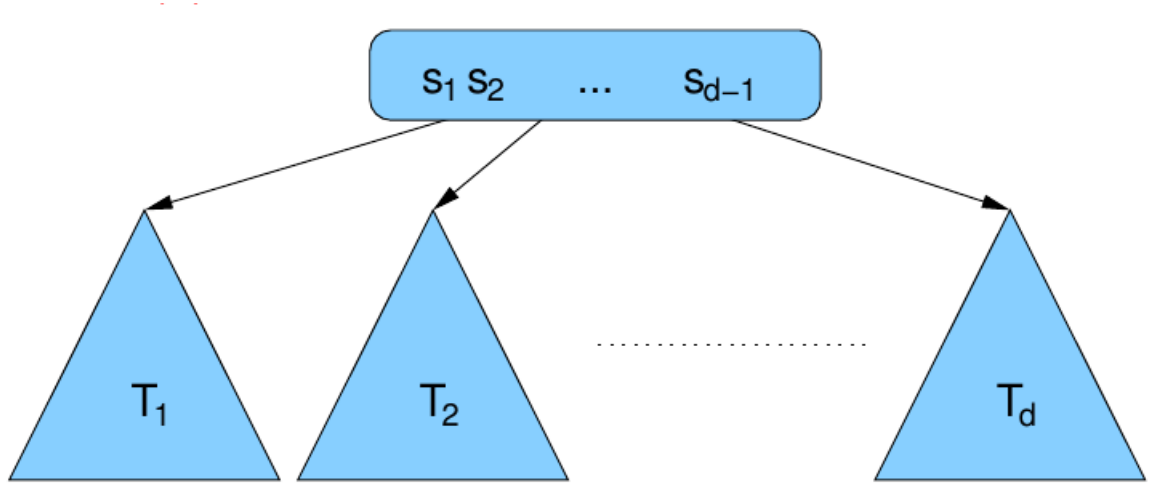
```
public Node postNext(Node v) {
    if (!isRoot(v)) {
        Node p = parent(v);
        if (hasrightChild(p)) {
            if (v == rightChild(p))
                return p;
            else {
                Node c = rightChild(p);
                while (isInternal(c)) {
                if (hasleftChild(c))
                    c = leftChild(c);
                else
                    c = rightChild(c);
                }
                return c;
            }
        }
        else return p;
    } else return null ;
}
```

Aufgabe 11.1 Baumtraversierung - `preNext(v)` Laufzeit



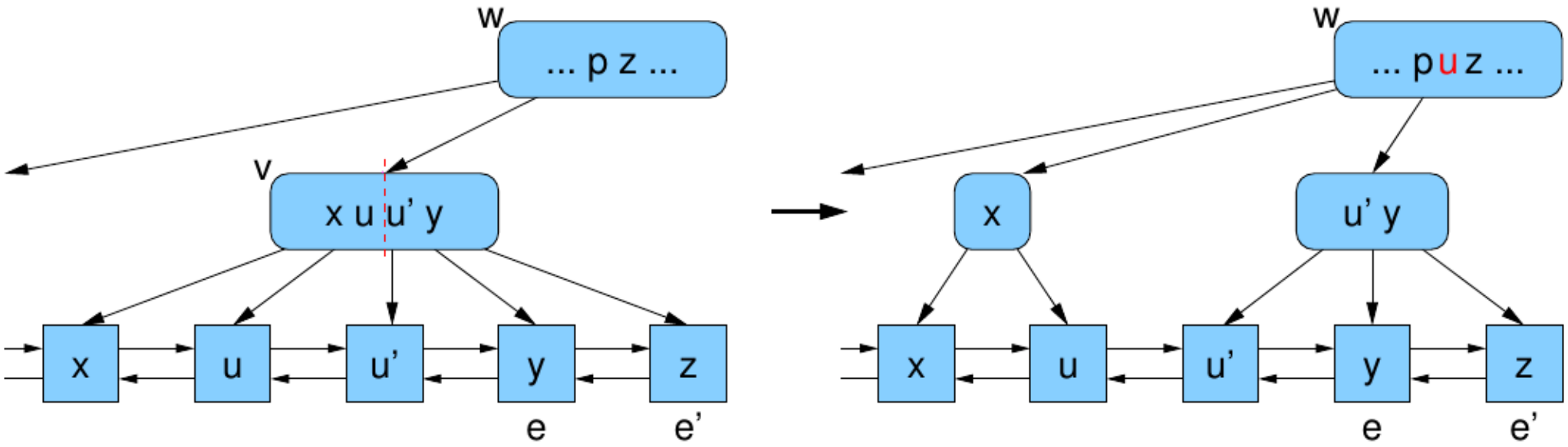
(a, b)-Bäume

- **Form-Invariante**: alle Blätter haben die selbe Tiefe
- **Grad-Invariante**: für alle internen Knoten gilt $a \leq d(v) \leq b$
  Spezialfall Wurzel: $2 \leq d(v) \leq b$
- (a, b)-Suchbaum-Regel:
  $\forall k \in T_1 : s_1 \geq k$ und
  $\forall k \in T_2 : s_1 < k$

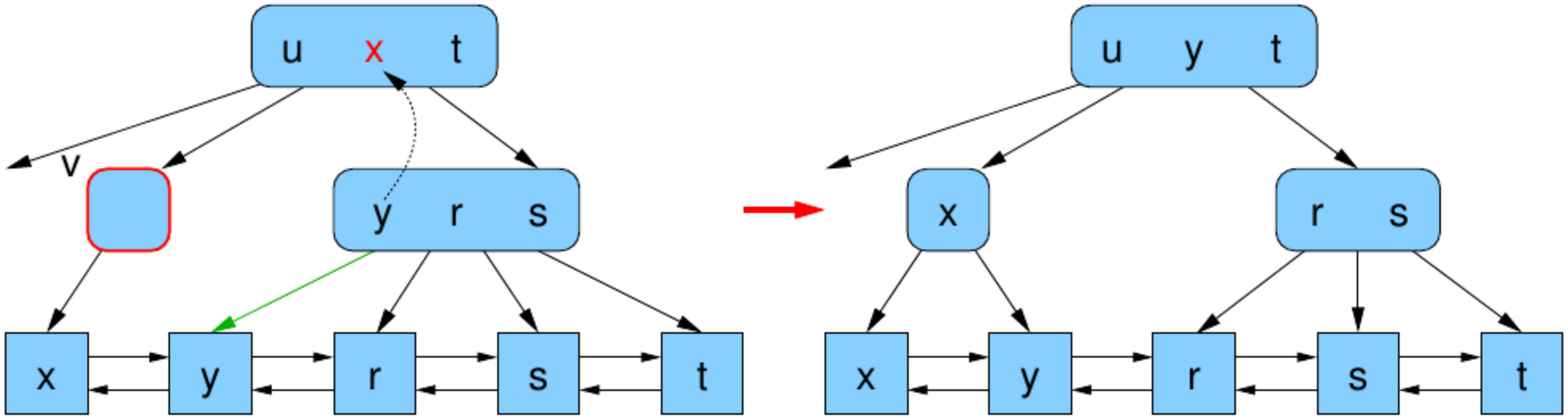


$d(v)$ ist die Anzahl an ausgehenden Kanten am Knoten v

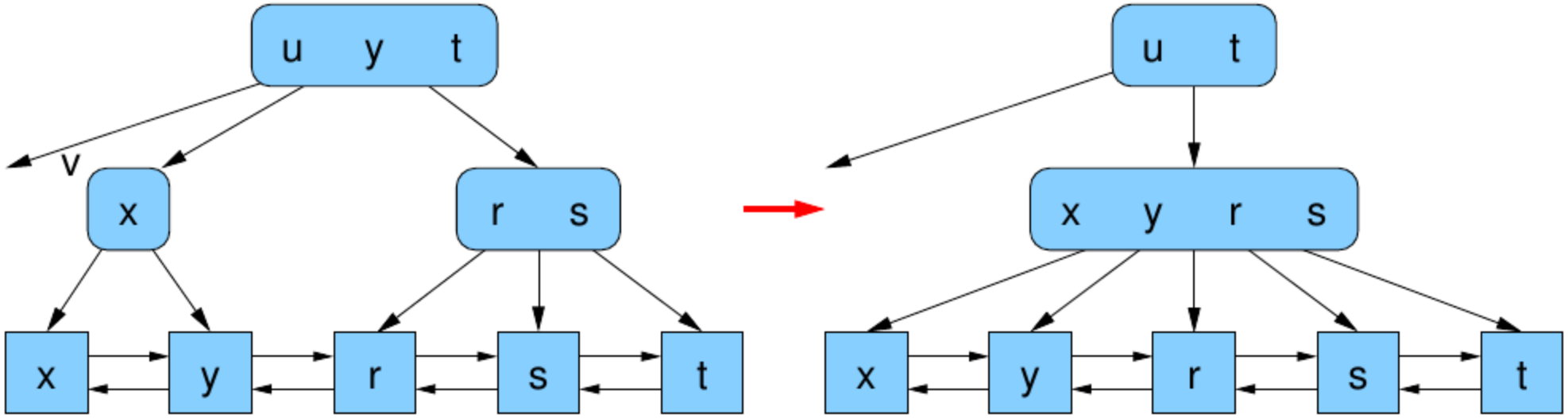
(a, b)-Bäume: Insert



(a, b)-Bäume: Remove - Steal



(a, b)-Bäume: Remove - Merge

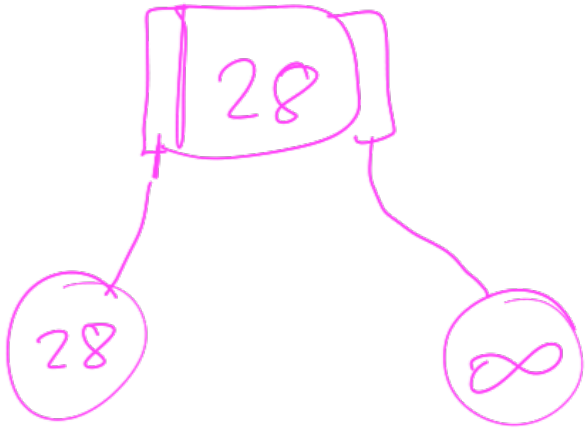


Aufgabe 11.2 ABBaumbaumaßnahmen I

> Führe auf einem leeren $(2, 3)$-Baum die folgenden Operationen aus:
>
> insert: $[19, 11, 28, 38, 37, 30, 7, 59, 41]$
>
> remove: $[7, 37, 59, 41, 11, 19, 30, 38]$
>
> - Zeichne den Baum nach jedem Schritt, Blattknoten müssen nicht gezeichnet werden.
> - Beim Aufspalten wandert das Element am Index $\lfloor \frac{b}{2} \rfloor$ nach oben.
> - Beim Löschen wird immer der symmetrische Vorgänger verwendet.
> - Es wird zunächst beim linken Nachbarn gestohlen.
> - Beim Verschmelzen wird zunächst der linke Nachbar betrachtet.

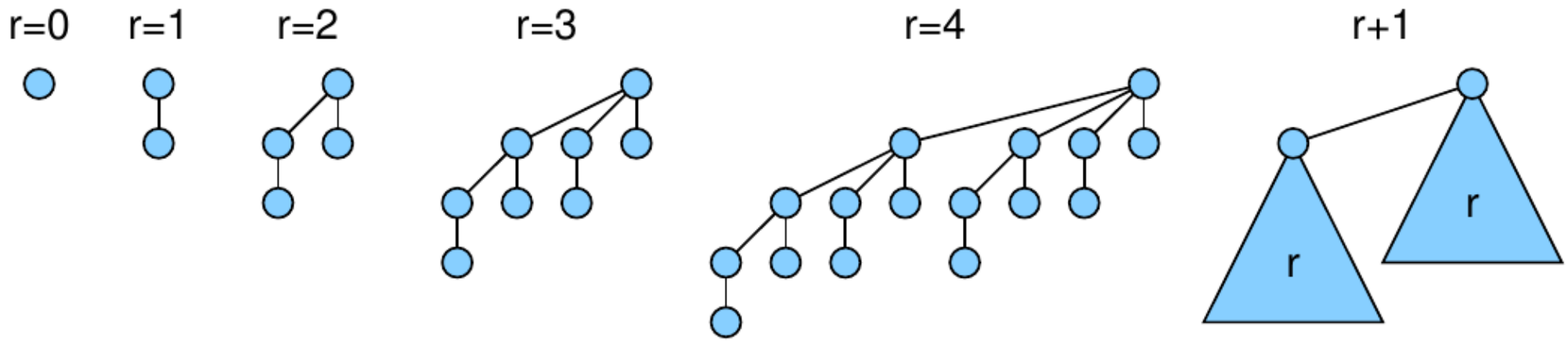
Aufgabe 11.2 ABBaumbaumaßnahmen I

insert: $[19, 11, 28, 38, 37, 30, 7, 59, 41]$
remove: $[7, 37, 59, 41, 11, 19, 30, 38]$



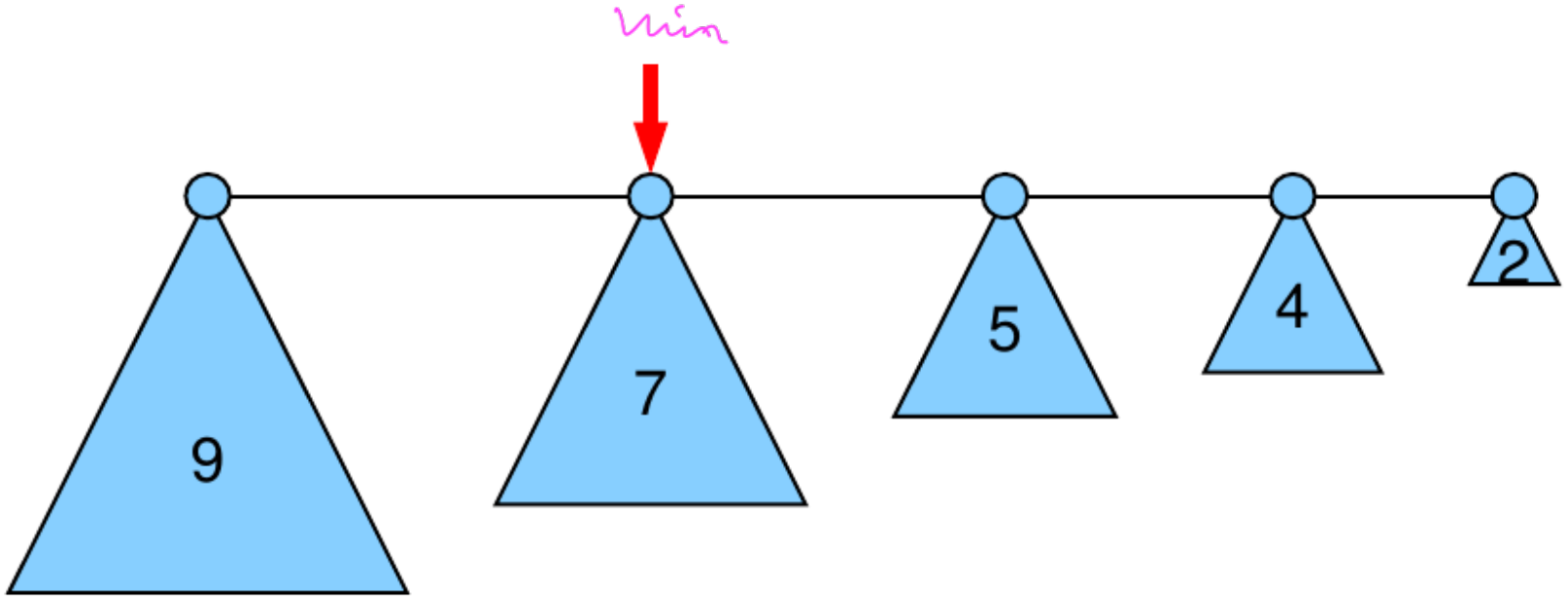
Binomial Bäume

- Jeder Baum hat einen Rang $r \in \mathbb{N}_0$
- Ein Binomial Baum mit $r = 0$ ist ein einzelner Knoten
- Ein Binomial Baum mit $r = n$ hat n Kinder mit jeweils Rang $r - 1, r - 2, \ldots, 0$



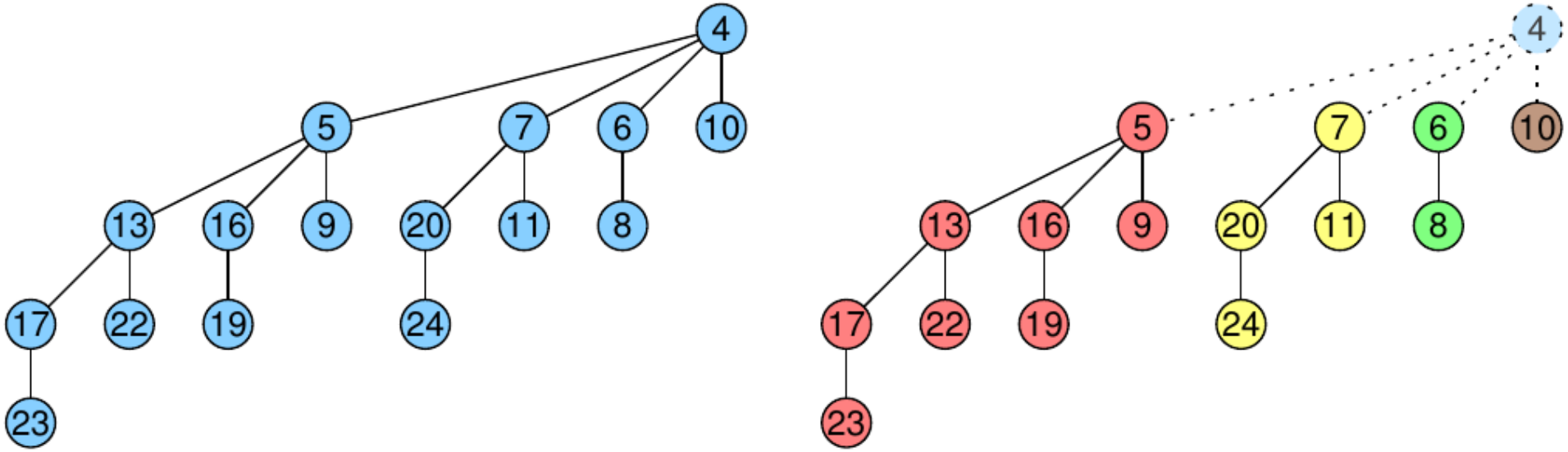
Binomial Heaps

- Menge an Binomial Bäumen (jeder Rang max. einmal vertreten)
- Zeiger auf minimales Element
- Jeder Binomial Baum erfüllt die **Min-Heap-Eigenschaft**

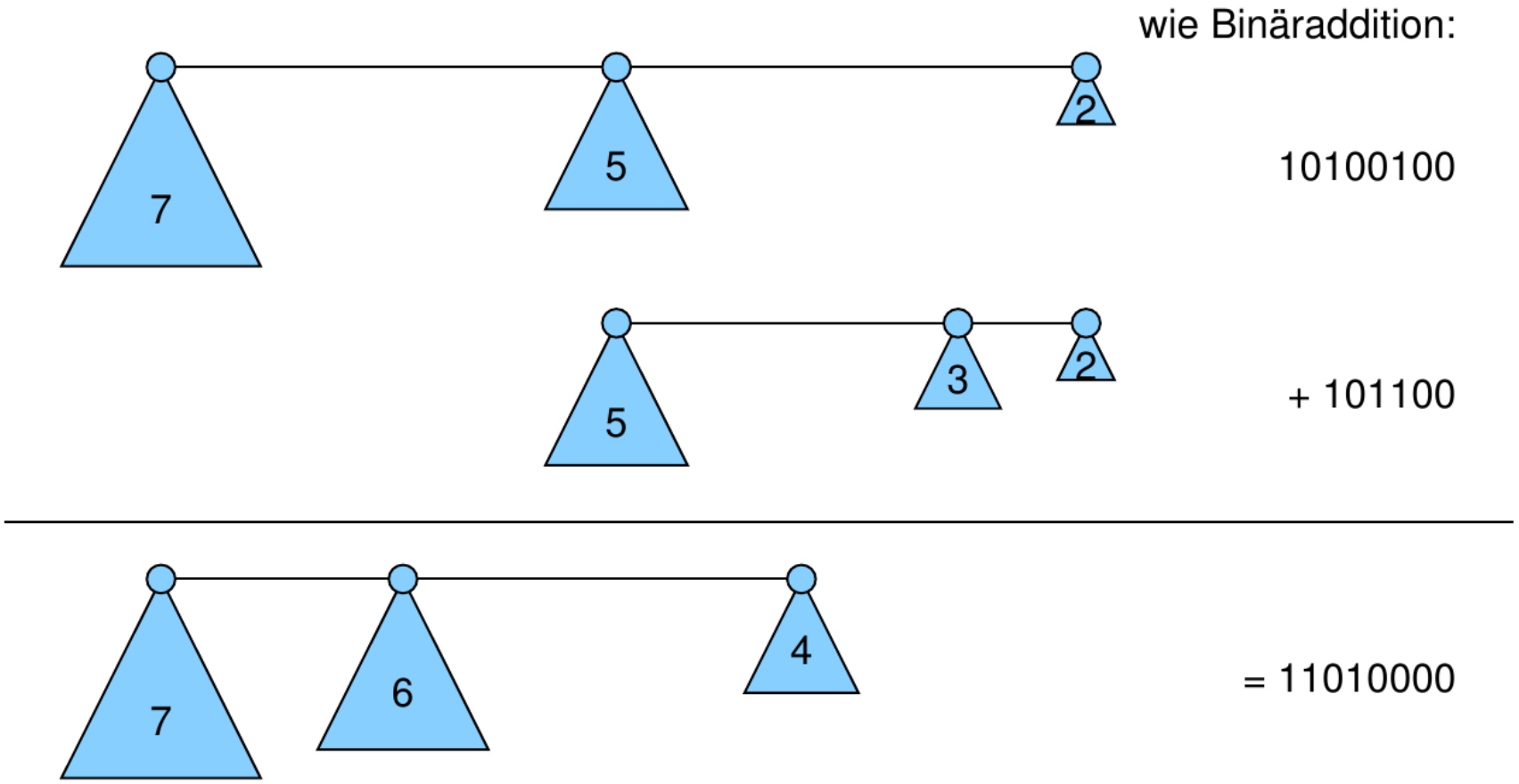


Binomial Heaps - deleteMin

- Minimales Element immer in der Wurzel des Baumes hinter Zeiger.
- Entfernen der Wurzel eines Baums mit Rang r erzeugt r Bäume mit Rang $r-1, r-2, \ldots, 0$.



Binomial Heaps - Merge

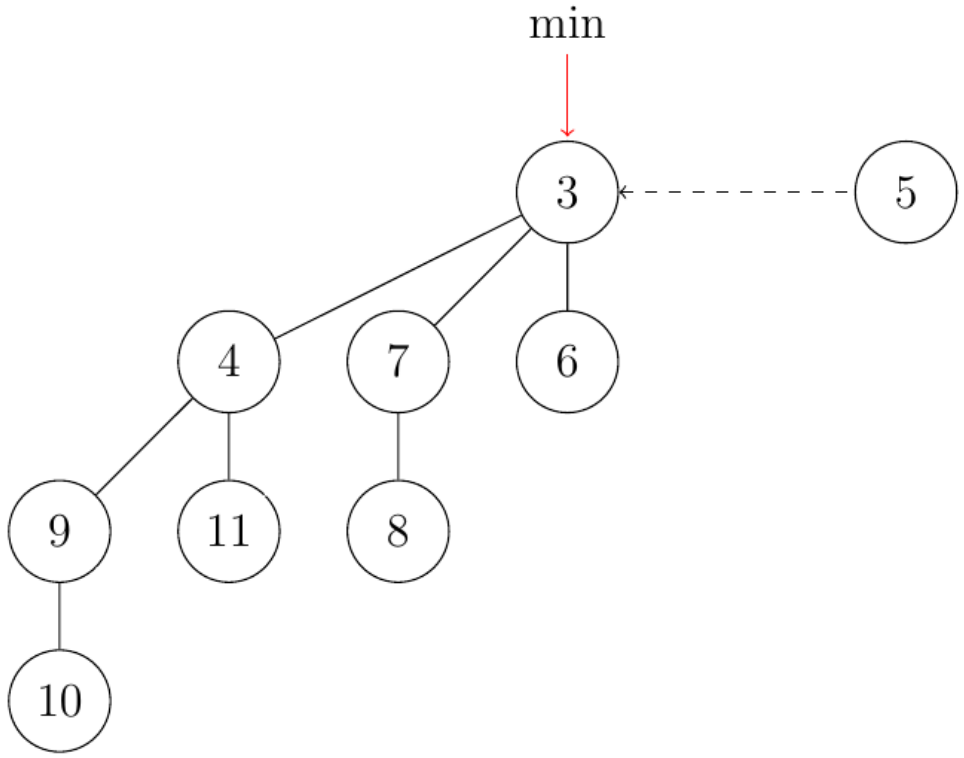


Binomial Heaps vs Binary Heaps

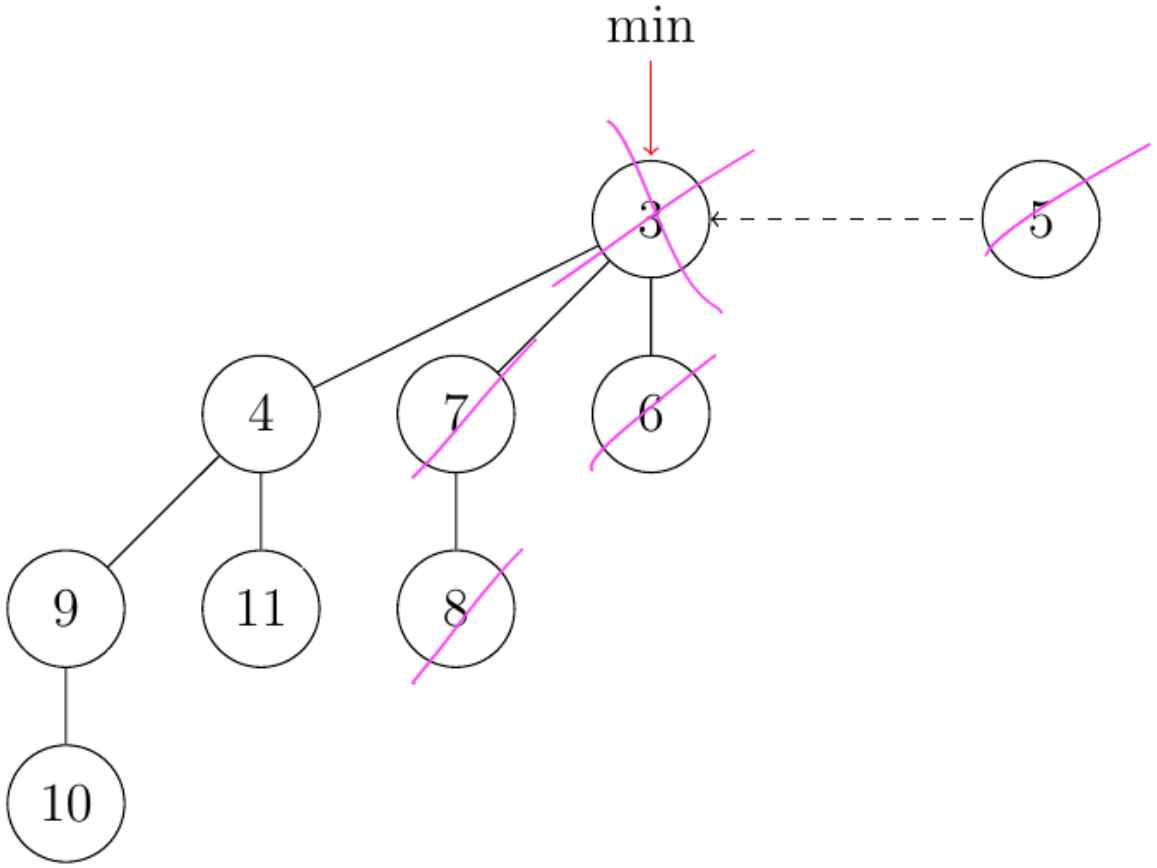
Operation	Binomial Heap	Binary Heap
`insert`	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
`min`	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`deleteMin`	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
`decreaseKey`	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
`merge`	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$

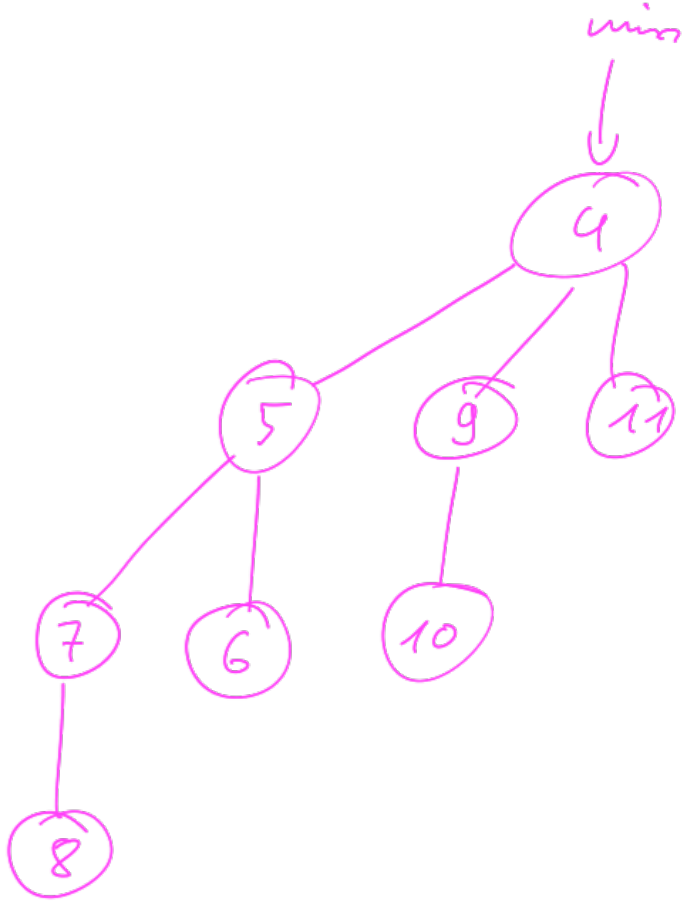
Aufgabe 11.3 Rückblick I: Binomialheaps

> Führe auf dem gezeigten Binomial Heap drei `deleteMin`-Operationen aus. Zeichne nach jeder der Operationen den entstandenen Binomial Heap.

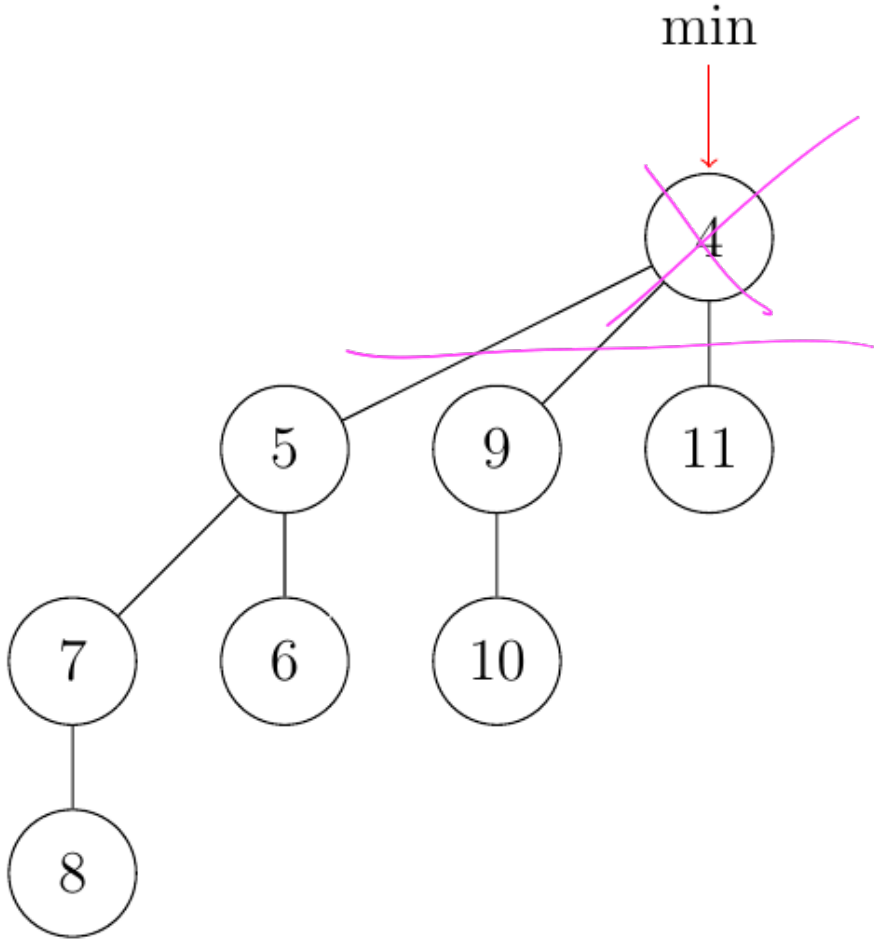


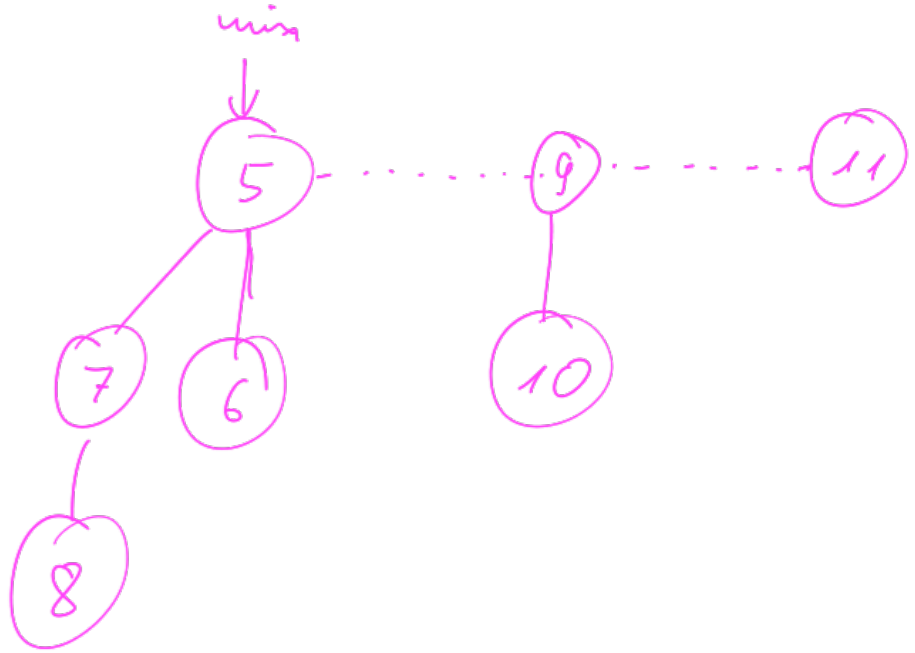
deleteMin 1



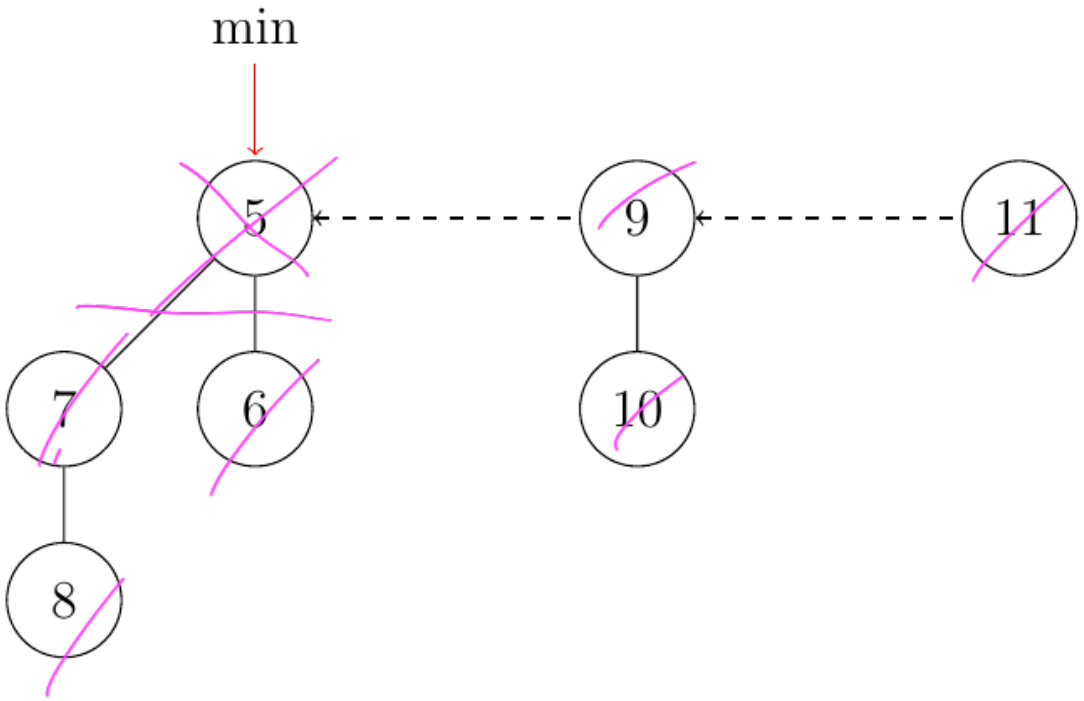


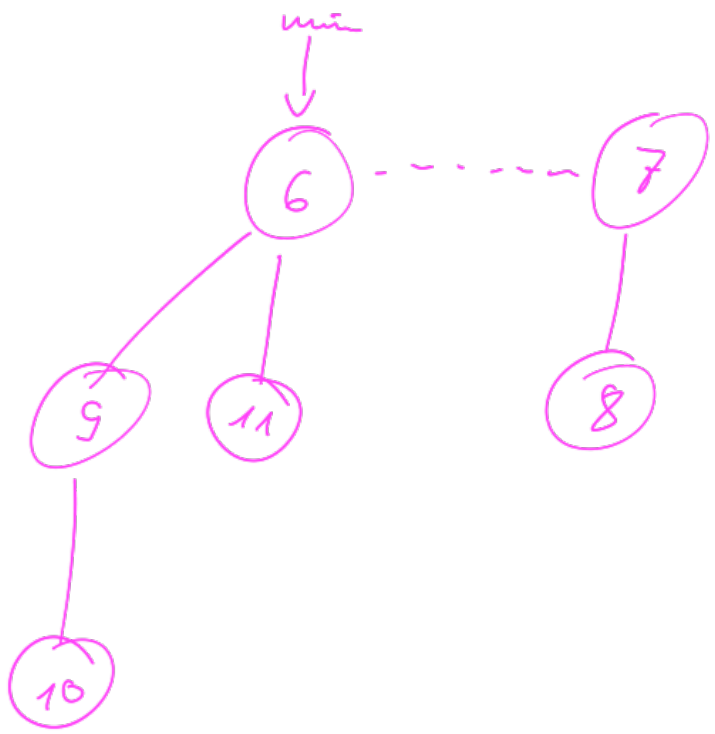
deleteMin 2





deleteMin 3





Binärbaum? AVL? RB? AB?

- **Binary Tree**:
  $\Rightarrow$ für Random Inserts (keine Rebalancierung)
- **AVL Tree**:
  $\Rightarrow$ für sortierte Inserts mit vielen Lookups (teure Rotationen)
- *Red-Black Tree*:
  $\Rightarrow$ für viele Inserts (billigerer Ausgleich)
  Beispiele: Java TreeSet / TreeMap; Linux Kernel; stdlibc++
- **(A)-B Tree**:
  $\Rightarrow$ für Hintergrundspeicher / Cache Effizienz
  Beispiele: DBMS; g++
- Radix-Tree (Trie)

Old but gold: Vergleichs-Paper (Ben Pfaff) | Sedgewick VL zu RB- und B-Trees | Sedgewick Buch: Algorithmen (TUM-UB, ~Seite 196)

Warum Binärbäume gefährlich sind

- Cache Verhalten: viele Random Reads, Kinder über Memory verteilt
- Höhe im Worst-Case: $\text{AVL} = 1.44 \cdot \log n$, $\text{RB-Tree} = 2 \cdot \log n$, $\text{Binary Tree} = n$
- Branching Verhalten: 50/50 Chance für jeden Kinder Branch $\Rightarrow$ Branch Predictor 😠