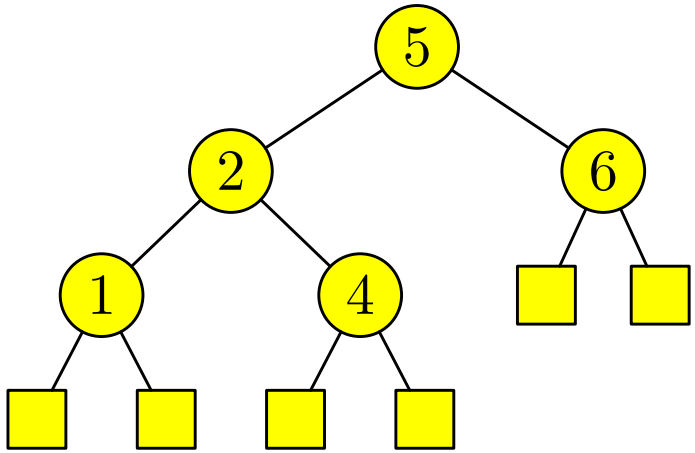
Binäre Suchbäume



- Als assoziatives Array geeignet
- Schlüssel aus geordneter Menge
- Linke Kinder kleiner, rechte Kinder größer als Elternknoten
- Externe und interne Knoten
- Externe Knoten zu einem Sentinel zusammenfassen?

Binäre Suchbäume

Java

```
public class Searchtree<K extends Comparable<K>, D>
                        extends Dictionary<K, D> {
  protected Searchtreenode<K, D> root;
```

Java

```
class Searchtreenode<K extends Comparable<K>, D> {
  K key;
  D data;
  Searchtreenode<K, D> left, right, parent;
```

Binäre Suchbäume – Suchen

Java

```
public D find(K k) {
  if(root ≡ null) return null;
  Searchtreenode⟨K, D⟩ n = root.findsubtree(k);
  return n ≡ null ? null : n.data;
}
```

Im Gegensatz zu Liste:
Zusätzlicher Test auf **null**.

Binäre Suchbäume – Suchen

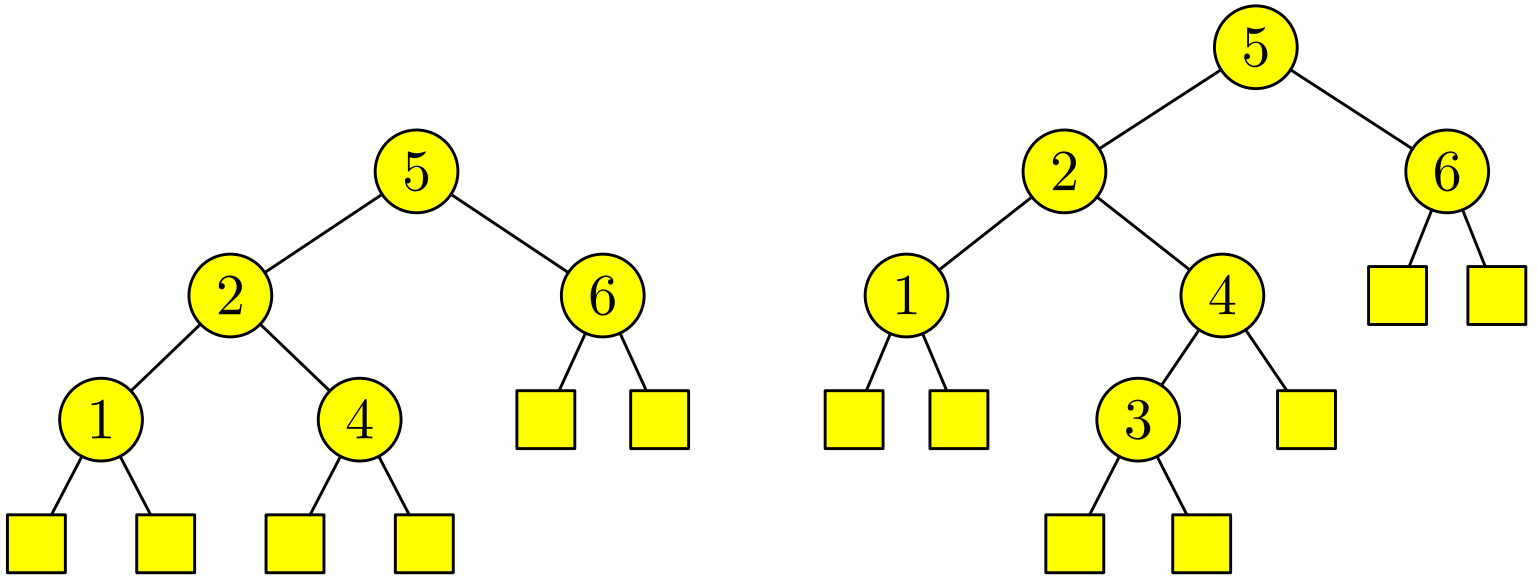
Java

```
Searchtreenode<K, D> findsubtree(K k) {
  int c = k.compareTo(key);
  if(c > 0) return right ≡ null ? null : right.findsubtree(k);
  else if(c < 0) return left ≡ null ? null : left.findsubtree(k);
  else return this;
}
```

Wieder zusätzlicher Test auf **null**.
Durch Sentinel verhindern!

Besser: Iterativ statt rekursiv.

Binäre Suchbäume – Einfügen



Wo fügen wir 3 ein?

Wie fügen wir es ein?

In den richtigen externen Knoten!

Binäre Suchbäume – Einfügen

Java

```
public void insert(K k, D d) {
  if(root ≡ null) root = new Searchtreenode⟨K, D⟩(k, d);
  else root.insert(new Searchtreenode⟨K, D⟩(k, d));
}
```

Binäre Suchbäume – Einfügen

Java

```
public void insert(Searchtreenode<K, D> n) {
  int c = n.key.compareTo(key);
  if(c < 0) {
      if(left ≠ null) left.insert(n);
      else {left = n; left.parent = this; }
  }
  else if(c > 0) {
      if(right ≠ null) right.insert(n);
      else {right = n; right.parent = this; }
  }
  else copy(n);
}
```

Binäre Suchbäume – Löschen

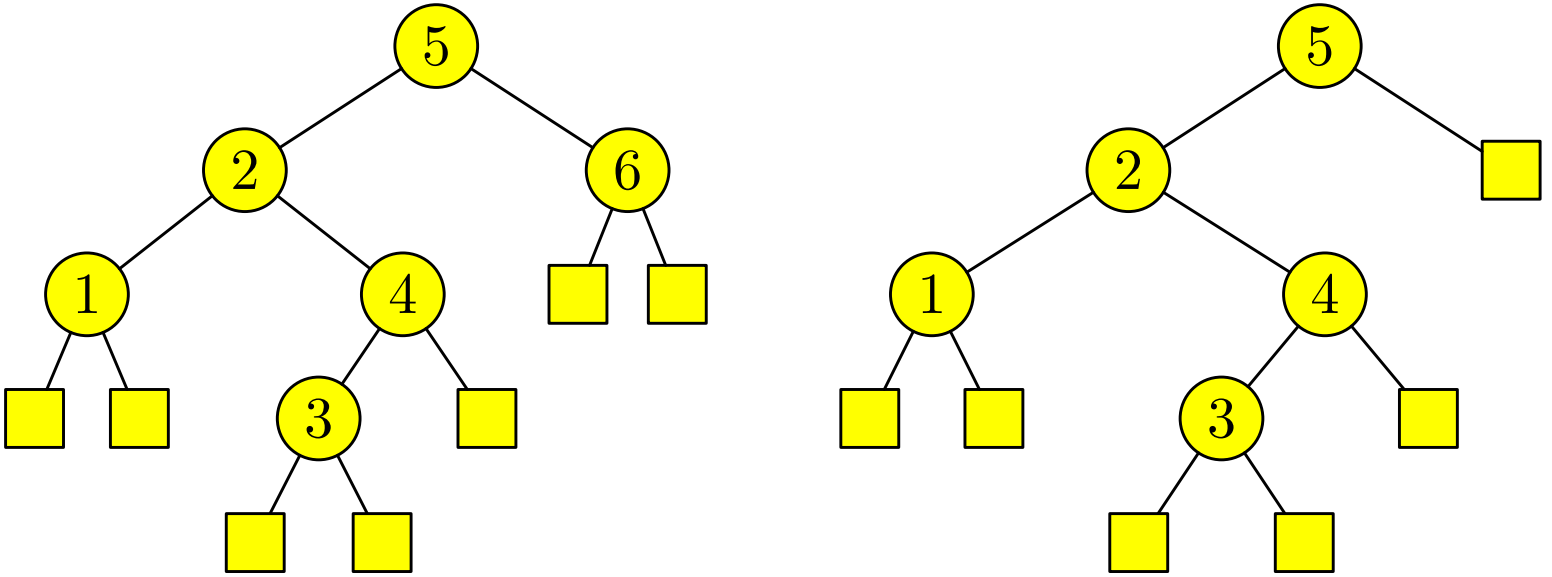
Beim Löschen unterscheiden wir drei Fälle:

- Blatt (einfach)
- Der Knoten hat kein linkes Kind (einfach)
- Der Knoten hat ein linkes Kind (schwierig)

Damit sind alle Fälle abgedeckt!
Warum kein Fall: Kein rechtes Kind?

Binäre Suchbäume – Löschen

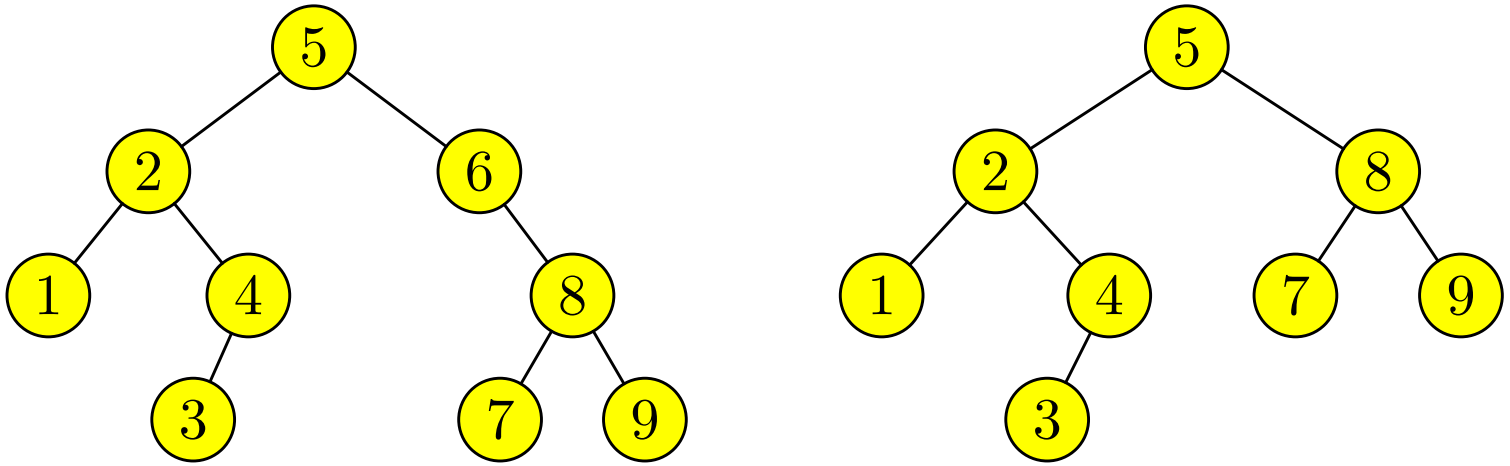
Löschen eines Blatts:



Ein Blatt kann durch Zeigerverbiegen gelöscht werden.

Binäre Suchbäume – Löschen

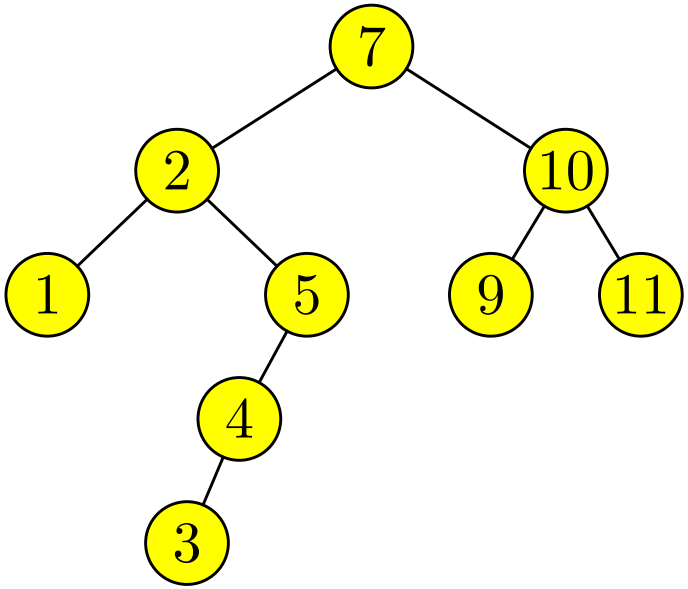
Löschen eines Knotens ohne linkes Kind:



Wir können kopieren oder Zeiger verbiegen.

Binäre Suchbäume – Löschen

Löschen eines Knotens mit linkem Kind:



1. Finde den größten Knoten im linken Unterbaum
2. Kopiere seinen Inhalt
3. Lösche ihn

Binäre Suchbäume – Löschen

In der Klasse Searchtree<K,D>:

Java

```
public void delete(K k) {
  if(root ≡ null) return;
  if(root.left ≡ null && root.right ≡ null && root.key ≡ k)
    root = null;
  else {
    Searchtreenode⟨K, D⟩ n = root.findsubtree(k);
    if(n ≠ null) n.delete();
  }
}
```

Binäre Suchbäume – Löschen

Java

```
void delete() {
  if(left ≡ null && right ≡ null) {
    if(parent.left ≡ this) parent.left = null;
    else parent.right = null; }
  else if(left ≡ null) {
    if(parent.left ≡ this) parent.left = right;
    else parent.right = right;
    right.parent = parent; }
  else {
    Searchtreenode⟨K, D⟩ max = left;
    while(max.right ≠ null) max = max.right;
    copy(max); max.delete();
  }
}
```

Binäre Suchbäume – Löschen

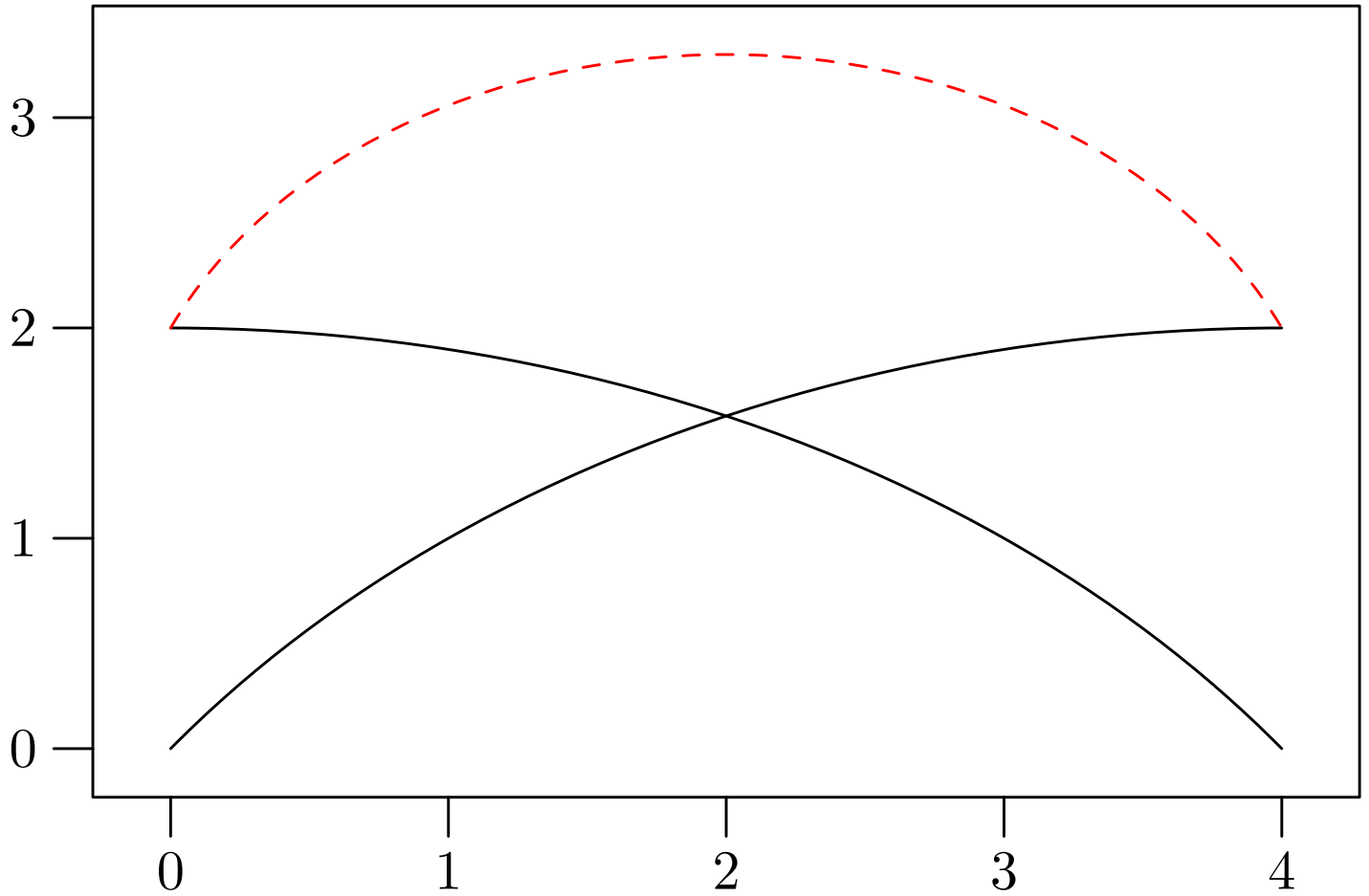
In Searchtree<K,D> jetzt korrekt:

Java

```
public void delete(K k) {
  if(root ≡ null) return;
  if(root.key.equals(k))
    if(root.left ≡ null && root.right ≡ null) {
      root = null; return;
    }
    else if(root.left ≡ null) {
      root = root.right; root.parent = null; return;
    }
  Searchtreenode⟨K, D⟩ n = root.findsubtree(k);
  if(n ≠ null) n.delete();
}
```

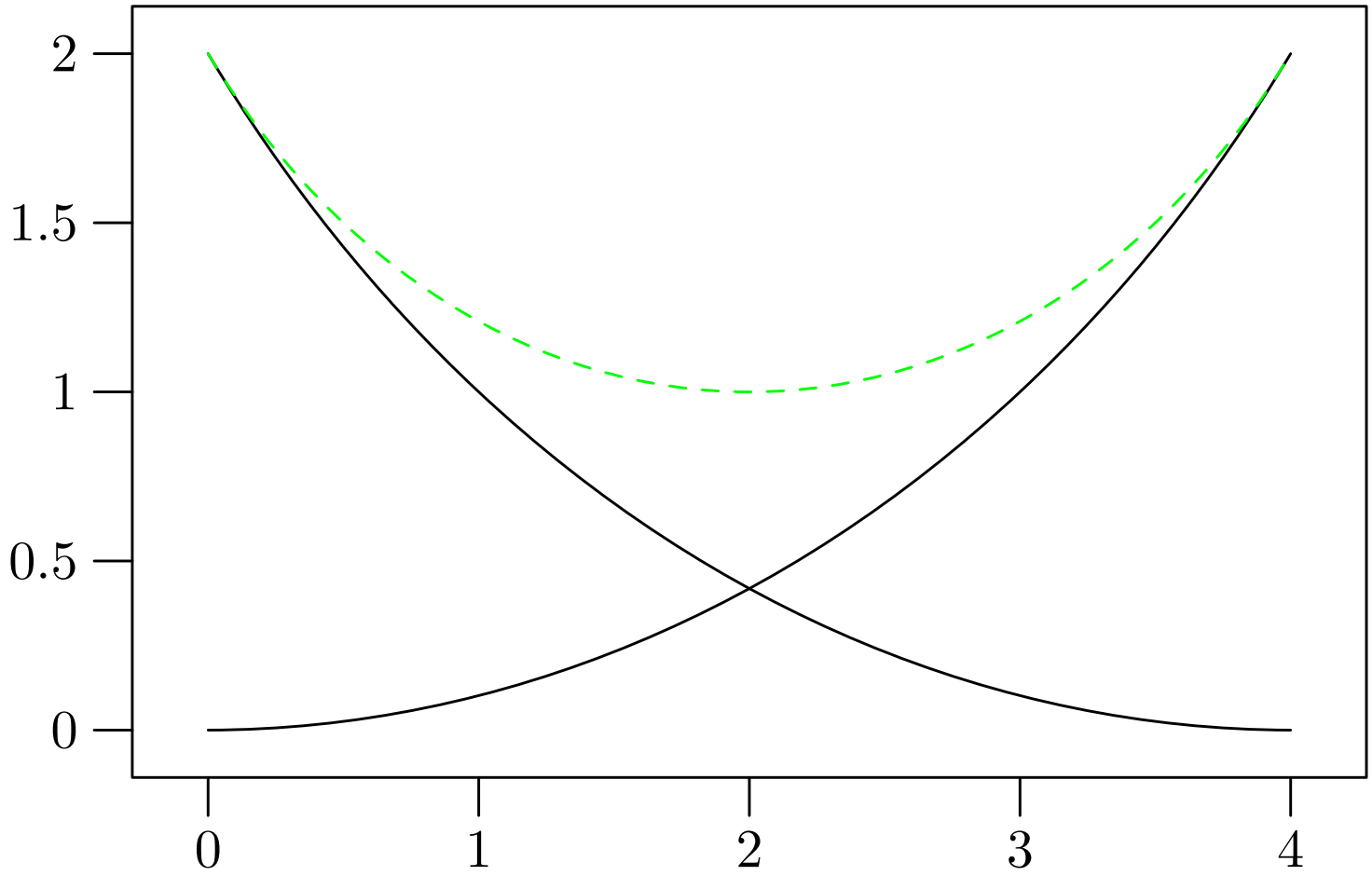
Binäre Suchbäume – Analyse

Eine kleine Vorbemerkung:



Summe schlechte Näherung des Maximums.

Binäre Suchbäume – Analyse



Summe gute Näherung des Maximums.

Die Kurven sind steiler.

Binäre Suchbäume – Analyse

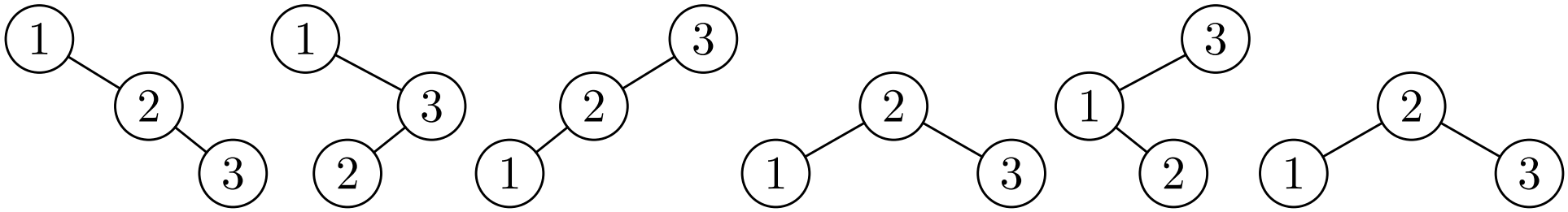
Wir fügen die Knoten $1, \ldots, n$ in zufälliger Reihenfolge in einen leeren Suchbaum ein.

Sei T_n die Höhe dieses Suchbaums.

Wir interessieren uns für $E(T_n)$.

Wir betrachten erst einmal T_0, T_1, T_2 und T_3:

- $E(T_0) = 0$
- $E(T_1) = 1$
- $E(T_2) = 2$
- $E(T_3) = 8/3$



Allgemeiner Fall:

Die Wurzel des Baums enthält $W \in \{1, \ldots, n\}$.

$$\Pr[W = k] = 1/n \text{ für } k \in \{1, \ldots, n\}$$

Wie sieht der Rest des Baums aus, falls $W = k$?

In den linken Teilbaum wurden $\{1, \ldots, k-1\}$ in zufälliger Reihenfolge eingefügt.

Seine Höhe ist T'_{k-1}.

Die Höhe des rechten Teilbaums ist T''_{n-k}.
Die Gesamthöhe ist $T_n = \max\{T'_{k-1}, T''_{n-k}\} + 1$.

Die Gesamthöhe ist $T_n = \max\{T'_{k-1}, T''_{n-k}\} + 1$.

$$E(T_n) = \frac{1}{n} \sum_{k=1}^{n} E(\max\{T'_{k-1}, T''_{n-k}\} + 1)$$

Wir können das Maximum durch die Summe abschätzen:

$$E(T_n) \leq \frac{1}{n} \sum_{k=1}^{n} \Big(E(T_{k-1}) + E(T_{n-k}) + 1 \Big)$$

Leider zu grob! Führt zu einer schlechten Abschätzung.

Problem:

$E(\max\{X, Y\}) \leq E(X) + E(Y)$ korrekt, aber zu ungenau.

$E(\max\{X, Y\}) \leq \max\{E(X), E(Y)\}$ genau genug, aber zu unkorrekt.

Führe neue Zufallsvariablen ein:

$$\hat{T}_n = 2^{T_n}, \quad \hat{T}'_n = 2^{T'_n}, \quad \hat{T}''_n = 2^{T''_n}$$

$$ET_n = \frac{1}{n} \sum_{k=0}^{n-1} E\left(\max\{T'_k, T''_{n-k-1}\} + 1\right)$$

$$E\hat{T}_n = \frac{1}{n} \sum_{k=0}^{n-1} E\left(2^{\max\{T'_k, T''_{n-k-1}\}+1}\right)$$

Die Kurven sind jetzt steiler!

Vereinfachen wir diese Rekursionsgleichung zunächst:

$$E\,\hat{T}_n = \frac{1}{n}\sum_{k=0}^{n-1} E\left(2^{\max\{T'_k, T''_{n-k-1}\}+1}\right) =$$

$$= \frac{1}{n}\sum_{k=0}^{n-1} 2E\left(\max\{2^{T'_k}, 2^{T''_{n-k-1}}\}\right) = \frac{2}{n}\sum_{k=0}^{n-1} E\left(\max\{\hat{T}'_k, \hat{T}''_{n-k-1}\}\right)$$

$$\leq \frac{2}{n}\sum_{k=0}^{n-1} E\left(\hat{T}'_k + \hat{T}''_{n-k-1}\}\right) =$$

$$= \frac{2}{n}\sum_{k=0}^{n-1}\left(E\,\hat{T}_k + E\,\hat{T}_k\}\right) = \frac{4}{n}\sum_{k=0}^{n-1} E\,\hat{T}_k$$

Diese Rekursionsgleichung läßt sich mit Standardmethoden lösen.
$\longrightarrow$ Vorlesung Analyse von Algorithmen

$$E\,\hat{T}_n = \frac{4}{n}\sum_{k=0}^{n-1} E\,\hat{T}_k$$

Wir „lösen“ diese Gleichung nicht.

Wir zeigen nur, daß $E\,\hat{T}_n \leq (n+3)^{\underline{3}} = (n+3)(n+2)(n+1)$:

$n = 1$: $E\,\hat{T}_1 = 2 \leq (1+3)^{\underline{3}}$

$n > 1$:

$$E\,\hat{T}_n \leq \frac{4}{n}\sum_{k=0}^{n-1} E\,\hat{T}_k \overset{\text{I.V.}}{\leq} \frac{4}{n}\sum_{k=0}^{n-1}(k+3)^{\underline{3}} = (n+3)^{\underline{3}}.$$