Karlsruher Institut für Technologie

Tutorium Algorithmen 1

08 · Suchbäume · 17.6.2024
Peter Bohner Tutorium 3

Wiederholung: Binäre Suche

- Ziel: Element in sortierter Folge finden
- Idee: Element mit Median vergleichen und dadurch eine Hälfte ausschließen
- Laufzeit: $O(\log n)$

Beispiel: 6 suchen

$$\langle 1, 7, 9, 50, 300, 700, 701 \rangle$$

7 Elemente: Median an Stelle 3
$\Rightarrow 50 > 6$

$$\langle 1, 7, 9 \rangle$$

3 Elemente: Median an Stelle 2
$\Rightarrow 6 < 7$

$$\langle 1 \rangle$$

$\Rightarrow$ 6 war nicht in der Folge

wichtig: $\Theta(1)$ Zugriff auf Median
$\Rightarrow$ Nur mit Arrays und nicht mit Listen umsetzbar

- Einfügen in sortiertes Array benötigt $O(n)$

Aufgabe

Ein sortiertes Array ohne Duplikate wurde um k Stellen nach rechts rotiert. Geben Sie einen Algorithmus in Pseudocode an, der dieses k in $\Theta(\log n)$ bestimmt (3 min)

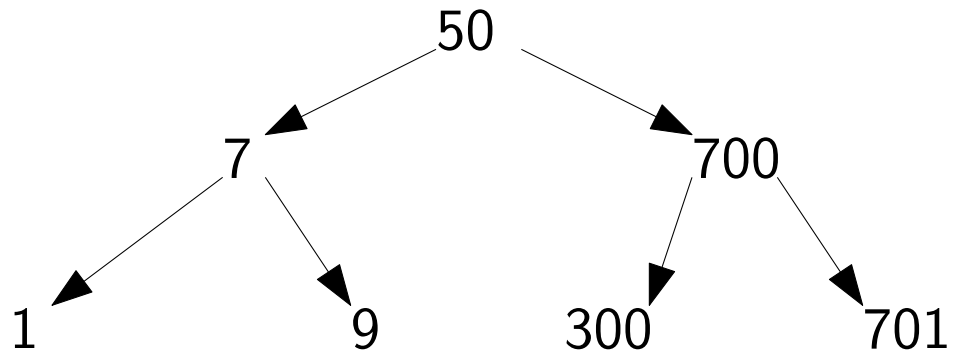
Beispiel: $\langle 3, 4, 1, 2 \rangle \Rightarrow k = 2 \quad \langle 3, 4, 5, 2 \rangle \Rightarrow k = 3$

```
findK(Array A)
  l, r ← 0, n − 1
  while l + 1 < r do
    m ← ⌊(l+r)/2⌋
    if A[l] > A[m] then
      r ← m
    else
      l ← m
    then
    if A[l] < A[r] then return l
    return r
```

Balancierte binäre Suchbäume

- Binärbaum $\Leftrightarrow$ Jeder Knoten max. 2 Kindknoten
- Idee: Median statt head/tail
- Median der kleineren Elemente statt prev
- Median der größeren Elemente statt next

$\langle 1, 7, 9, 50, 300, 700, 701 \rangle$



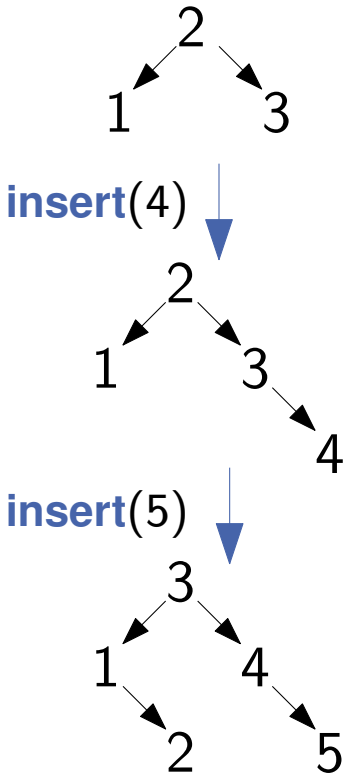
Höhe $\approx \log n \Rightarrow$ Jeder Knoten ist in $O(\log n)$ erreichbar $\Rightarrow$
Suchen benötigt $O(\log n)$ Schritte

Balancierte binäre Suchbäume II

- Einfügen: Baum $\mapsto$ Folge $\mapsto$ Folge (mit e) $\mapsto$ Baum (mit e)

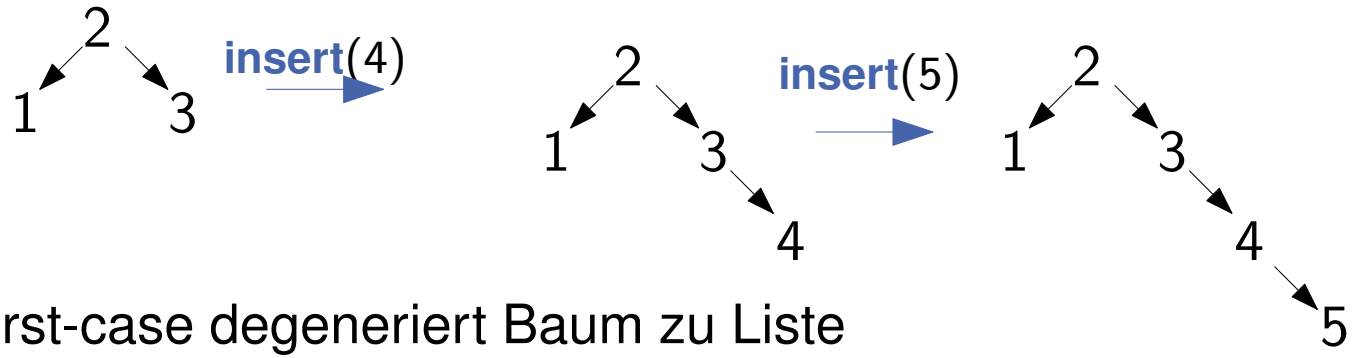
$\Rightarrow$ Einfügen ist teuer ($O(n)$)

$\Rightarrow$ für uns nicht besser als sortiertes Array



Idee: wir speichern nicht Median, sondern irgendein kleineres Element

- binärer Suchbaum (ohne das balanciert)
- Suchen geschieht analog
- Einfügen dort, wo Element erwartet wird



$\Rightarrow$ im worst-case degeneriert Baum zu Liste

$\Rightarrow$ Suchen/Einfügen im worst-case in $O(n)$

- im worst-case schlechter als sortierte Liste

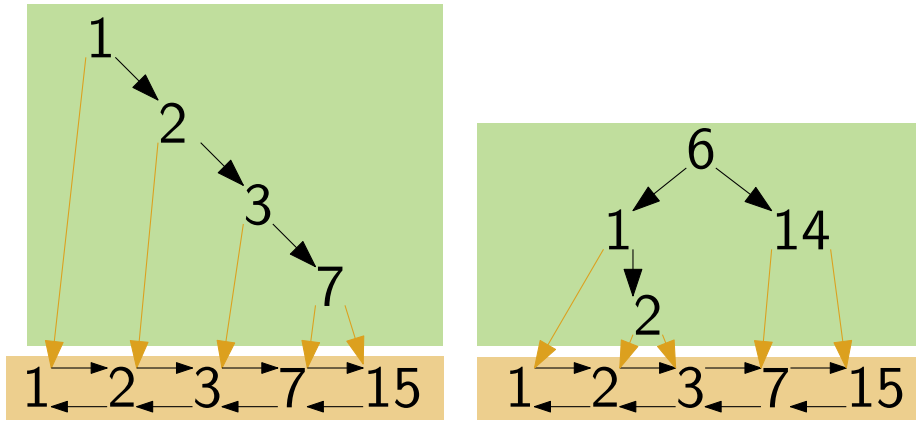
Aufgabe

Beschreiben Sie einen Algorithmus, der in $O(n)$ aus einem Binärbaum eine sortierte Liste erstellt (3min)

Wir erstellen rekursiv aus beiden Kind-Bäumen eine Liste.
Für einen leeren Baum ist die Liste leer.
Wir fügen das Element des Knotens in die linke Liste ein.
Wir verschmelzen die linke Liste mit der rechten Liste.
$\Rightarrow$ Jeder Knoten wird einmal betrachtet & Aufwand pro Knoten ist konstant $\Rightarrow \in O(n)$

Spaltschlüssel

- **oben:** Suchbaum
  - Elemente im linken Teilbaum sind kleiner **gleich** Schlüssel
- **unten:** verkette Liste
- Trennung zwischen Nutz- und Navigationsdaten
- Schlüssel im Suchbaum müssen **nicht** in Liste sein



Beschreiben Sie einen Algorithmus, der in $O(n)$ einen balancierten Binärbaum (mit Spaltschlüsseln) zu einer sortierten Liste erstellt. $n = 2^k$, $k \in \mathbb{N}_0$ (3min)

Lösung

Beschreiben Sie einen Algorithmus, der in $O(n)$ einen balancierten Binärbaum (mit Spaltschlüsseln) zu einer sortierten Liste erstellt. $n = 2^k$, $k \in \mathbb{N}_0$ (3min)

Wir speichern pro Teilbaum Median und größtes Element

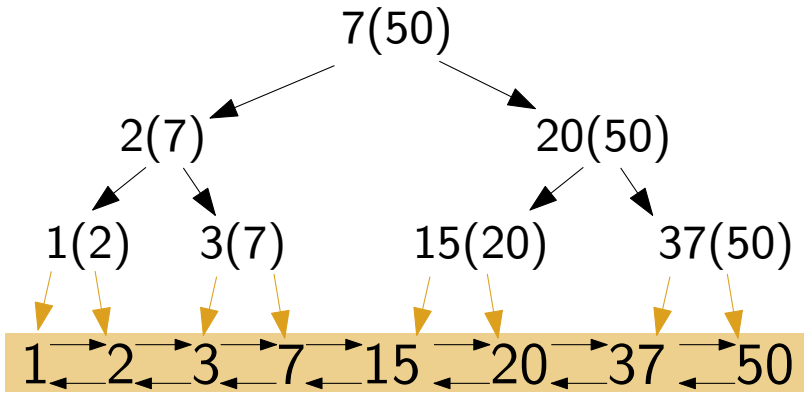
Wir bauen den Baum von der tiefsten zur höchsten Schicht

Schreibweise: *Median(gr. Element)*

Wir erstellen jede Schicht, indem wir das Maximum des linken Kindes als Median und das Maximum des rechten Kindes als neues Maximum verwenden

Die gerade Knoten auf einer Ebene sind die linken Kinder und die ungeraden die rechten Kinden

Für die tiefste Schicht sind die Elemente der Liste die Kinder. Dabei ist der Median/Maximum jeweils der Wert des Listeneintrags



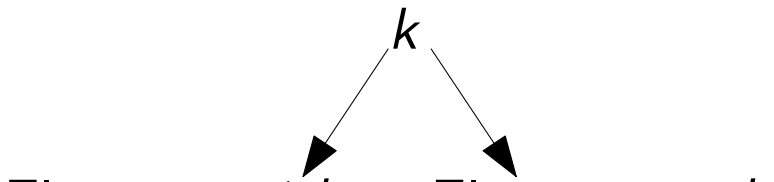
$\Rightarrow$ Median wird Spaltschlüssel

Laufzeit: $T(n) = \begin{cases} T(n/2) + n, n > 0 \\ 1, n = 0 \end{cases}$

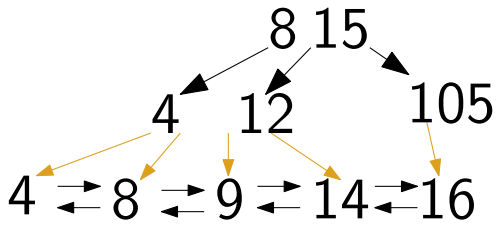
$\Rightarrow T \in O(n)$

Spaltschlüssel II

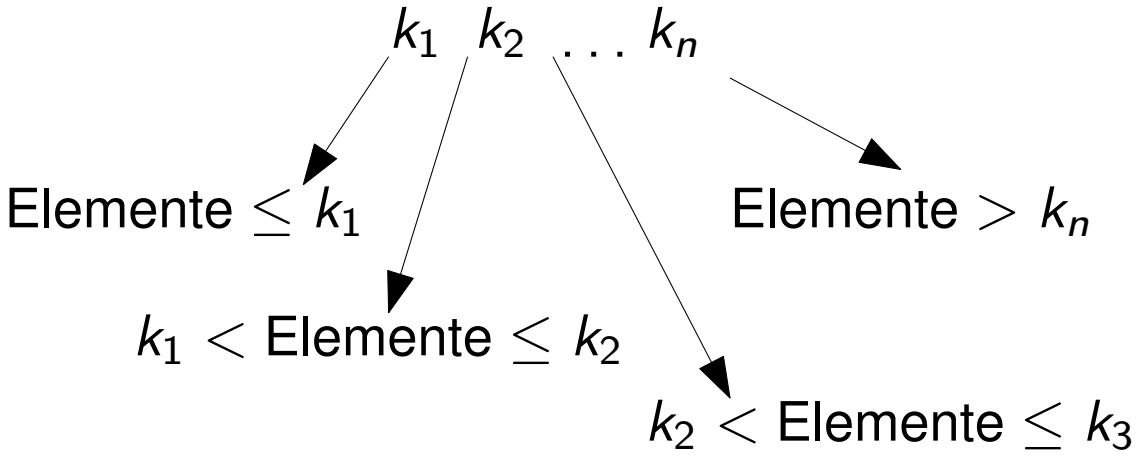
Für Binärbäume



Beispiel: $n = 3$



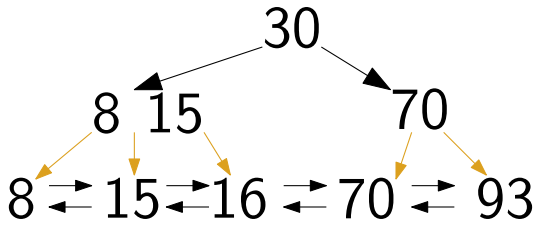
Für Bäume mit bis zu n Kindern

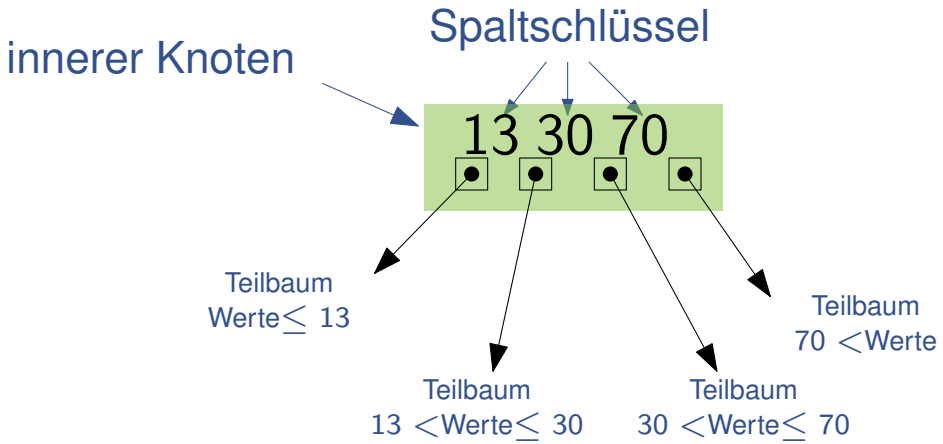


(a, b)-Bäume

- Parameter: $2 \leq a \leq (b+1)/2$
- $a \leq$ Anzahl Kindknoten $\leq b$
  - für Wurzel nur $\leq b$
- Baum balanciert sich selber
  - Suchen/Einfügen/Entfernen $\in O(\log n)$

Beispiel: $(2, 3)$-Baum

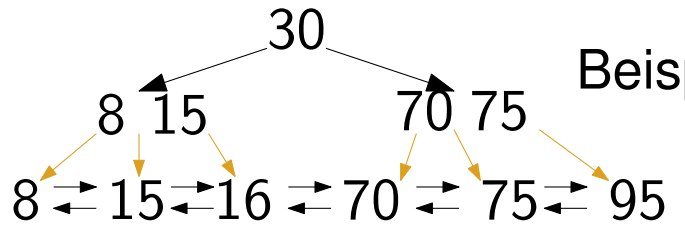




Einfügen in (a, b)-Baum

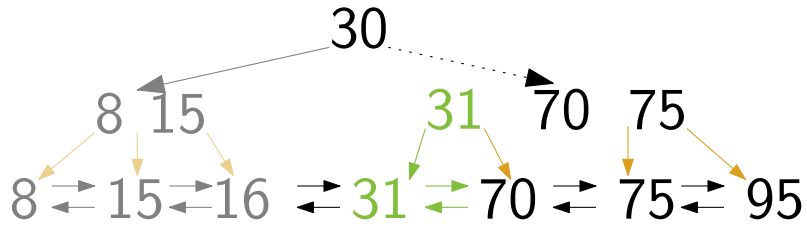
- Pfad in Baum suchen
- Neues Element in Liste einfügen
- Neuen Spaltschlüssel einfügen
- ggf. neu balancieren
  - überfüllten Knoten aufspalten
  - überflüssigen Spaltschlüssel nach oben geben

$\Rightarrow$ balancieren ggf. rekursiv

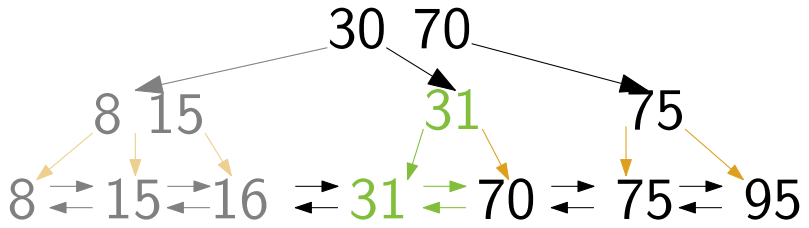


Beispiel: **insert**(31)

$(2, 3)$-Baum



Wurzel könnte jetzt überfüllt sein

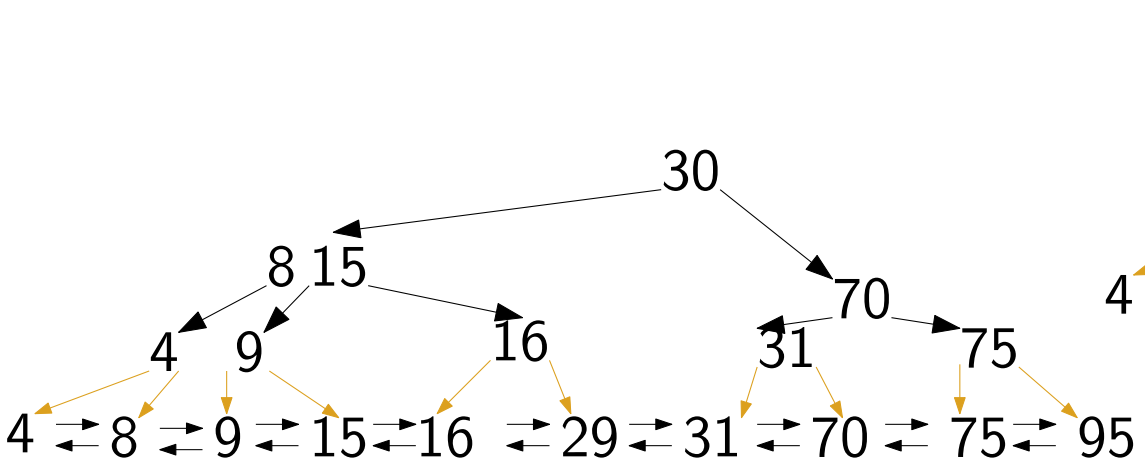


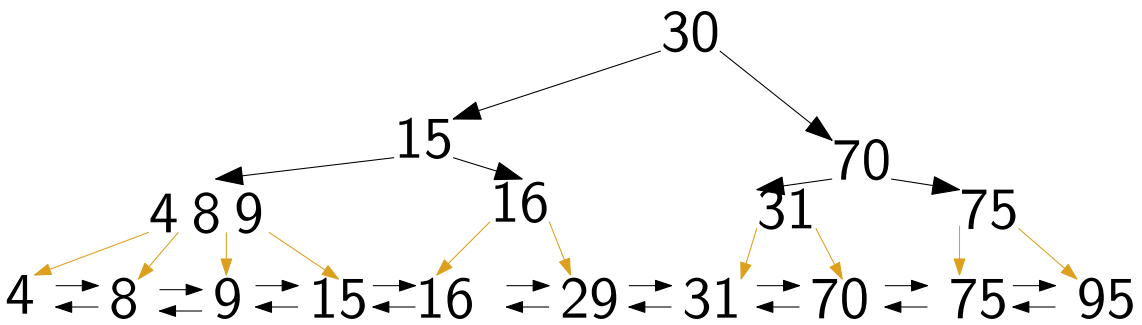
Aufgabe

- Pfad in Baum suchen
- Neues Element in Liste einfügen
- Neuen Spaltschlüssel einfügen
- ggf. neu balancieren
  - überfüllten Knoten aufspalten
  - überflüssigen Spaltschlüssel nach oben geben

$\Rightarrow$ balancieren ggf. rekursiv

Aufgabe: Fügen sie 29, 4 und 9 in den $(2, 3)$-Baum ein.
Wie kann man das Einfügen rückgängig machen (entfernen)? (5min)





Entfernen

- Pfad in Baum suchen
- Element aus Liste entfernen
- Spaltschlüssel entfernen
- ggf. neu balancieren
  - unterbesetzten Knoten mit Nachbarn verschmelzen
    - Bei Wurzel nicht möglich (daher nur $\leq b$ viele Kinder)
  - Spaltschlüssel von oben holen
  - ggf. überfüllten Knoten aufspalten
  - ggf. überflüssigen Spaltschlüssel nach oben geben

$\Rightarrow$ balancieren ggf. rekursiv

Zusammenfassung

Suchbäume

- balancierte Binärbäume sind teuer beim Einfügen
- unbalancierte (Binär-)bäume werden ggf. zu Listen

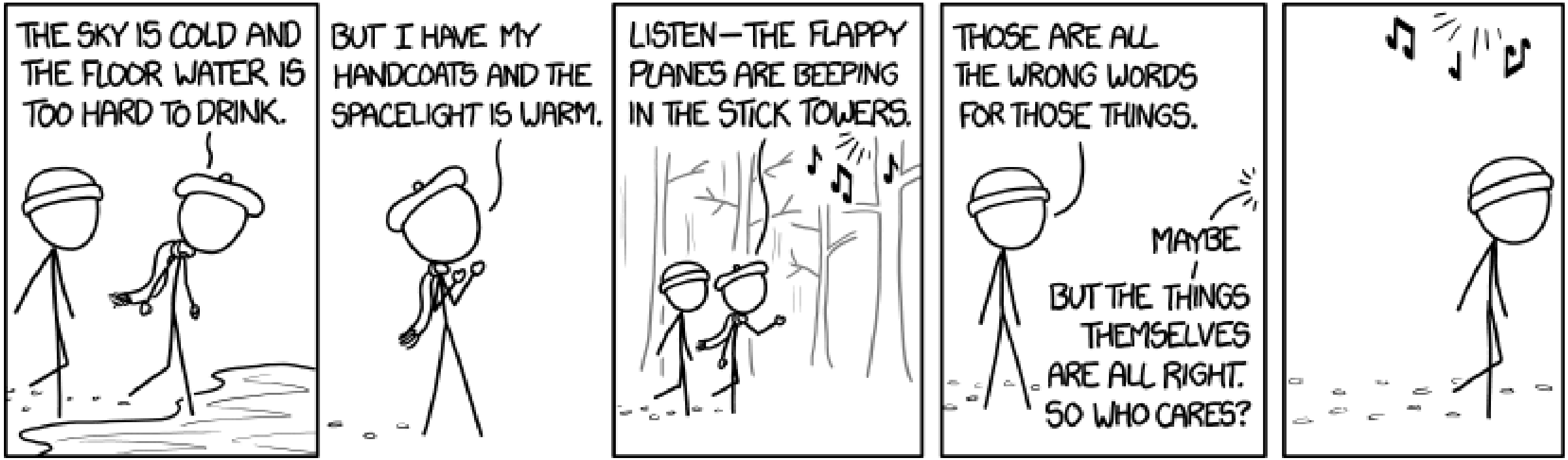
- (a, b)-Bäume
  - Balancieren sich selber in $O(\log n)$
  - Bis zu b-Kinder (mehr Vergleiche), dafür schnelles Einfügen
  - Neue Knoten entstehen oben und nicht unten

Fragen?

Karlsruher Institut für Technologie

Fragen!

Ende



https://xkcd.com/1322/