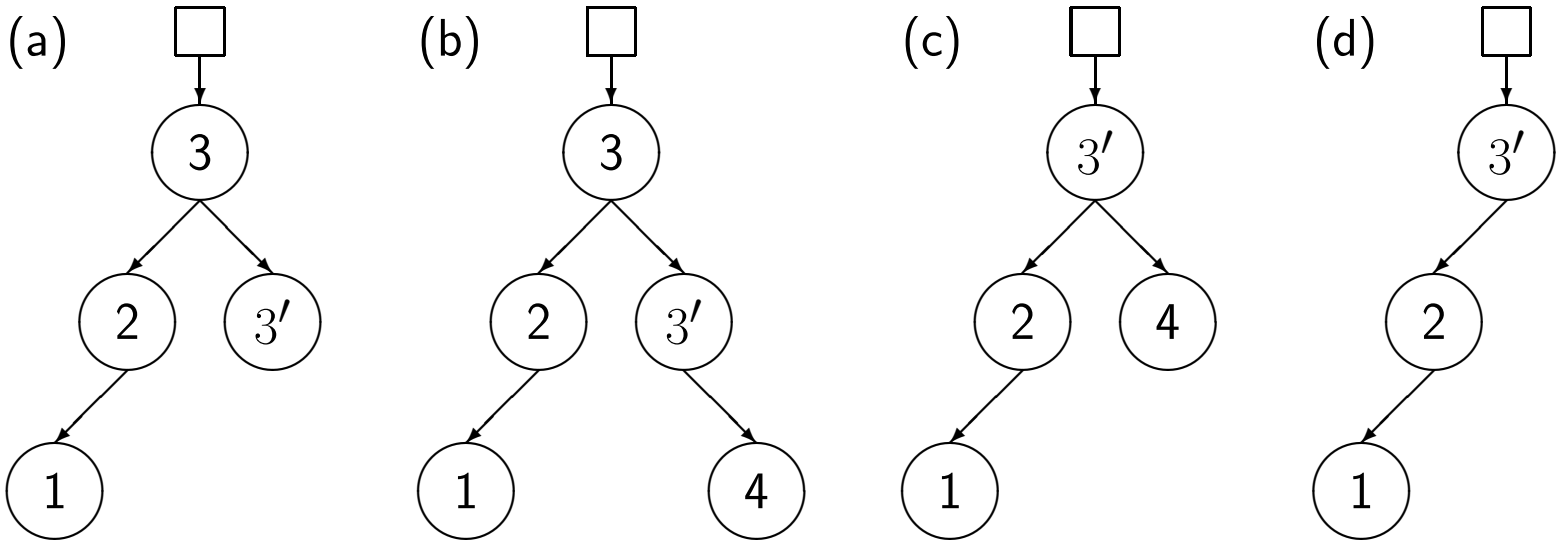
Binärer Baum



Aufgabe: Baum versus Liste

Such Sie in Gruppen zu zwei bis drei Personen Antworten auf folgende Fragen:

1. Welche Vorteile haben Bäume gegenüber Listen?
2. Welche Vorteile haben Listen gegenüber Bäumen?

Zeit: 2 Minuten

Algorithmische Kosten

Kostenabschätzungen

grobe Abschätzung von Laufzeit und Speicherverbrauch

konstante Faktoren vernachlässigt
→ geschätzte Kosten unabhängig von Hardware, Compiler, ...

Anzahl der Elemente in Datenstruktur berücksichtigt

Abschätzung maximaler und durchschnittlicher Kosten

Kosten typisch für Algorithmen bzw. Datenstrukturen

Beispiele für Laufzeit-Kosten

Operation	Durchschnitt	Maximum
Einfügen in Liste:	$O(1)$	$O(1)$
Suche in Liste:	$O(n)$	$O(n)$
Löschen aus Liste:	$O(n)$	$O(n)$
Einfügen in Baum:	$O(\log(n))$	$O(n)$
Suche in Baum:	$O(\log(n))$	$O(n)$
Löschen aus Baum:	$O(\log(n))$	$O(n)$

$O(1)$ = konstant (sehr niedrig)

$O(\log(n))$ = logarithmisch

$O(n)$ = linear

$O(n^2)$ = quadratisch

$O(n^k)$ = polynomial

$O(2^n)$ = exponentiell (sehr hoch)

Beispiele für Speicher-Kosten

Kosten für Liste und Baum entspricht Anzahl der Knoten: $O(n)$

Kosten für Operationen auf Liste und Baum:

- iterativ: $O(1)$
- rekursiv: gleich den Kosten für Laufzeit $O(\log(n))$ bis $O(n)$
- dieser Unterschied kaum relevant da höhere Kosten $O(n)$ entscheidend

Ausgewogenheit zwischen Laufzeit und Speicher wichtig

Beispiel: Sortiertes Ausgeben

Verkettete Liste: (Sortieren notwendig)

Liste aufbauen: $n \cdot O(1) = O(n)$

Sortieren: $O(n^2)$ für Bubble Sort

Ausgeben: $O(n)$

insgesamt: $\max(O(n), O(n^2), O(n)) = O(n^2)$

Binärer Baum: (schon bei Einfügen sortiert)

Baum aufbauen (Ø): $n \cdot O(\log(n)) = O(n \cdot \log(n))$
Baum aufbauen (maximal): $n \cdot O(n) = O(n^2)$

Ausgeben: $O(n)$

insgesamt (Ø): $\max(O(n \cdot \log(n)), O(n)) = O(n \cdot \log(n))$
insgesamt (maximal): $\max(O(n^2), O(n)) = O(n^2)$