

Vorbemerkungen über binäre Bäume

$\mathcal{B}$  : Menge der binären Bäume, rekursiv definiert durch die Regeln:

- $\square$  ist ein binärer Baum
- sind  $t_\ell, t_r$  binäre Bäume, so ist auch  $t = \langle \circ, t_\ell, t_r \rangle$  ein binärer Baum
- nur das, was durch die beiden vorigen Regeln erzeugt werden kann, ist ein binärer Baum

Übliche Darstellung:

Bäume mit Wurzel, wobei jeder Knoten zwei oder keinen Knoten als (geordnete) Nachfolger hat.

- zwei Nachfolger: mit  $\circ$  bezeichnete inneren Knoten,
- kein Nachfolger: mit  $\square$  bezeichneten äusseren Knoten (Blätter).

Lineare Codierung von Binärbbäumen (Wörter über dem Alphabet  $\{\circ, \square\}$ )

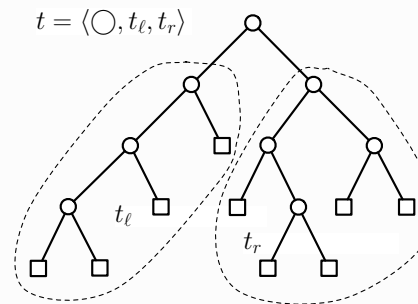
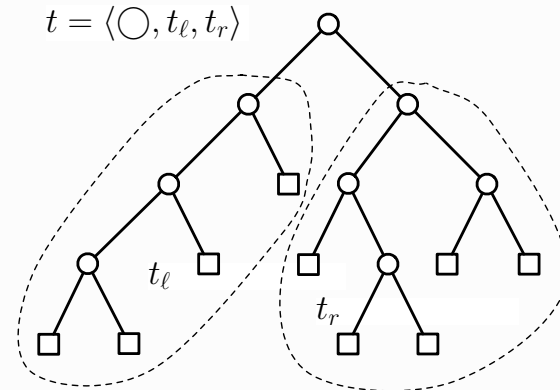
$$\text{code}(\square) = \square \quad \text{code}(\langle \circ, t_\ell, t_r \rangle) = \circ \text{code}(t_\ell) \text{code}(t_r)$$

Entsprechend: Generierung mittels einer kontextfreien Grammatik in BNF  $G_{\mathcal{B}}$

- Variablensymbol  $B$  (auch Startsymbol)
- Terminalsymbolen  $\square$  und  $\circ$
- Produktionen

$$B \rightarrow \square \mid \circ BB$$

- $\square$  ist ein binärer Baum, der nur aus einem einzigen (äusseren) Knoten besteht;
- $t = \langle \circ, t_\ell, t_r \rangle$  ist ein Baum mit Wurzel  $\circ$ , an die zwei binäre Bäume  $t_\ell$  und  $t_r$  als linker bzw. rechter Teilbaum angehängt sind.



$$\text{code}(t_\ell) = \circ \circ \circ \square \square \square \square,$$

$$\text{code}(t_r) = \circ \circ \square \circ \square \square \circ \square \square$$

$$\text{code}(t) = \circ \text{code}(t_\ell) \text{code}(t_r)$$

$$= \circ \circ \circ \circ \square \square \square \square \circ \circ \square \square \square \square \square \square$$

Bemerkungen:

- streicht man in jedem von  $G_{\mathcal{B}}$  erzeugten Wort das letzte Symbol und identifiziert man

$$\circ \sim \text{"linke Klammer"} \quad \square \sim \text{"rechte Klammer"},$$

so entsprechen diese Wörter genau den wohlgeformten Klammersausdrücken entsprechender Länge.

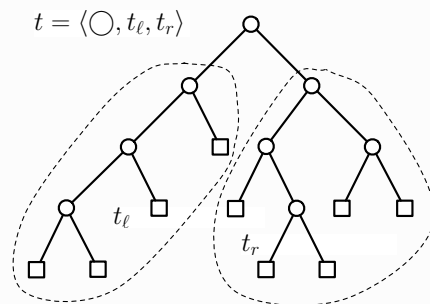
- die von  $G_{\mathcal{B}}$  erzeugte Sprache ist kontextfrei, aber nicht regulär
- die Anzahl der binären Bäume mit  $n$  inneren (und  $n + 1$  äusseren) Knoten ist

$$c_n = \frac{1}{n+1} \binom{2n}{n} \in \Theta\left(\frac{4^n}{n^{3/2}}\right)$$

5

- $h(t)$  = Höhe ("height") von  $t$ :

$$h(t) = \begin{cases} 0 & \text{falls } t = \square \\ 1 + \max\{h(t_\ell), h(t_r)\} & \text{falls } t = \langle \circ, t_\ell, t_r \rangle \end{cases}$$



$$i(t) = 8, e(t) = 9, s(t) = 17, h(t) = 4$$

7

Parameter für Binärbäume (und auch für andere Typen von Bäumen), die induktiv definiert oder beschrieben werden:

- $i(t)$  = Anzahl der inneren ("internal") Knoten von  $t$ :

$$i(t) = \begin{cases} 0 & \text{falls } t = \square \\ 1 + i(t_\ell) + i(t_r) & \text{falls } t = \langle \circ, t_\ell, t_r \rangle \end{cases}$$

- $e(t)$  = Anzahl der äusseren ("external") Knoten von  $t$ :

$$e(t) = \begin{cases} 1 & \text{falls } t = \square \\ e(t_\ell) + e(t_r) & \text{falls } t = \langle \circ, t_\ell, t_r \rangle \end{cases}$$

- $s(t) = i(t) + e(t)$  = Grösse ("size") von  $t$ :

$$s(t) = \begin{cases} 1 & \text{falls } t = \square \\ 1 + s(t_\ell) + s(t_r) & \text{falls } t = \langle \circ, t_\ell, t_r \rangle \end{cases}$$

6

Für alle binären Bäume  $t$  gelten folgende Aussagen:

- $e(t) = i(t) + 1$ , und damit  $s(t) = 2 \cdot i(t) + 1 = 2 \cdot e(t) - 1$

Beweis

$$e(\square) - i(\square) = 1 - 0 = 1$$

$$\begin{aligned} e(\langle \circ, t_\ell, t_r \rangle) - i(\langle \circ, t_\ell, t_r \rangle) &= e(t_\ell) + e(t_r) - (i(t_\ell) + i(t_r) + 1) \\ &= e(t_\ell) - i(t_\ell) + e(t_r) - i(t_r) - 1 \\ &= 1 + 1 - 1 = 1 \end{aligned}$$

8

2.  $h(t) \leq i(t)$

Beweis

$$\begin{aligned}
 h(\square) &= 0 = i(\square) \\
 h(\langle \circ, t_\ell, t_r \rangle) &= 1 + \max\{h(t_\ell), h(t_r)\} \\
 &\leq 1 + \max\{i(t_\ell), i(t_r)\} \\
 &\leq 1 + i(t_\ell) + i(t_r) \\
 &= i(\langle \circ, t_\ell, t_r \rangle)
 \end{aligned}$$

9

3.  $e(t) \leq 2^{h(t)}$ , also  $\log e(t) \leq h(t)$

Beweis

$$\begin{aligned}
 e(\square) &= 1 = 2^0 = 2^{h(\square)} \\
 e(\langle \circ, t_\ell, t_r \rangle) &= e(t_\ell) + e(t_r) \\
 &\leq 2^{h(t_\ell)} + 2^{h(t_r)} \\
 &\leq 2 \cdot 2^{\max\{h(t_\ell), h(t_r)\}} \\
 &= 2^{1+\max\{h(t_\ell), h(t_r)\}} \\
 &= 2^{h(\langle \circ, t_\ell, t_r \rangle)}
 \end{aligned}$$

10

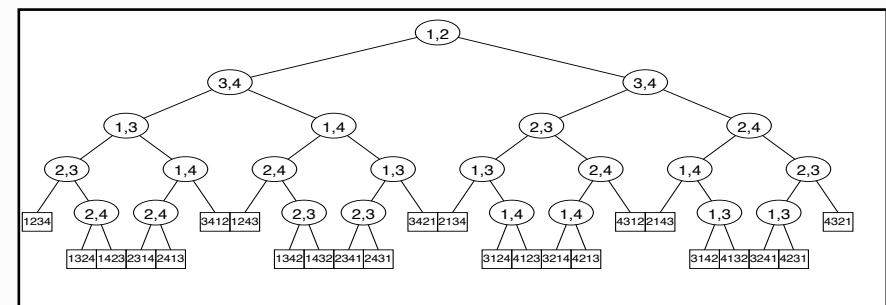
Untere Schranke für das Sortieren

Vergleichsbasierte Sortialgorithmen können mittels beschrifteter binärer Bäume (Entscheidungsbäume) veranschaulicht werden:

- Eingabe ist Liste  $(x_1, x_2, \dots, x_n)$  der Länge  $n$ , wobei die  $x_i$  paarweise verschieden (Permutationsmodell)
- innere Knoten sind mit Paaren  $(i, j)$  ( $1 \leq i < j \leq n$ ) beschriftet
- äussere Knoten (Blätter) sind mit Permutationen von  $\{1, 2, \dots, n\}$  beschriftet
- jeder Weg von der Wurzel zu einem Blatt identifiziert schrittweise die Struktur der Eingabe anhand der Inversionen:
  - an einem mit  $(i, j)$  beschrifteten Knoten
    - gehe weiter in den linken Teilbaum, falls  $x_i < x_j$
    - gehe weiter in den rechten Teilbaum, falls  $x_i > x_j$
  - an einem Blatt: die Permutation stellt die Größenrelationen der Eingabe dar

11

Beispiel: mergesort



12

- für jeden binären Baum  $t$  gilt

$$e(t) \leq 2^{h(t)}$$

wobei  $e(t)$  = Anzahl der Blätter,  $h(t)$  = Höhe, also

$$\log e(t) \leq h(t)$$

- Operiert ein Sortieralgorithmus  $\mathcal{A}$  auf Eingaben der Länge  $n$ , so muss der zugehörige Entscheidungsbaum  $t_{\mathcal{A}}(n)$  mindestens  $n!$  Blätter haben, denn alle  $n!$  möglichen Permutationen müssen identifiziert werden können.
- Somit gilt für die Anzahl  $V_{\mathcal{A}}(n)$  im worst-case

$$V_{\mathcal{A}}(n) = h(t_{\mathcal{A}}(n)) \geq \lceil \log e(t_{\mathcal{A}}(n)) \rceil \geq \lceil \log n! \rceil$$

13

#### average-case Analyse

- Jeder Entscheidungsbaum für ein Sortierverfahren auf inputs der Länge  $n$  hat genau  $n!$  Blätter
- mittlere Höhe eines Binärbaumes

$$\bar{h}(t) = \frac{1}{e(t)} \sum_{b \in E(t)} h(b, t)$$

wobei  $E(t)$  = Menge der Blätter von  $t$ ,  $h(b, t)$  = Höhe des Blattes  $b$  in  $t$

- Induktiv:

$$\bar{h}(\square) = 0$$

$$\bar{h}(\langle \circ, t_\ell, t_r \rangle) = \frac{e(t_\ell)}{e(t)} \bar{h}(t_\ell) + \frac{e(t_r)}{e(t)} \bar{h}(t_r) + 1$$

15

- Erinnerung: STIRLINGS Formel

$$n! = \left(\frac{n}{e}\right)^n \sqrt{2\pi \cdot n} \left(1 + O\left(\frac{1}{n}\right)\right)$$

- Theorem (GA, Th. 2.22): Jeder vergleichsbasierte Sortieralgorithmus benötigt auf Folgen der Länge  $n$  im worst-case mindestens

$$n \log n - n \log e + O(\log n) \approx n \log n - 1.44n \quad \text{Vergleiche}$$

14

- Lemma (GA, Lemma 2.24): Für jeden Binärbaum gilt

$$\bar{h}(t) \geq \log e(t)$$

- Theorem (GA, Theorem 2.26): Jeder vergleichsbasierte Sortieralgorithmus benötigt auf Folgen der Länge  $n$  im average-case mindestens

$$n \log n - n \log e + O(\log n) \approx n \log n - 1.44n \quad \text{Vergleiche}$$

16

## Bemerkungen

- Sortieralgorithmen, die  $\Theta(n \log n)$  Vergleiche für inputs der Länge  $n$  benötigen (im worst-case bzw. average-case), sind *asymptotisch optimal*
- *Mergesort* und *Heapsort* sind asymptotisch optimale Sortierverfahren im worst-case (und daher auch im average-case)
- *Selectionsort* (*Bubblesort*) und *Insertionsort* sind weder im worst-case, noch im average-case, asymptotisch optimal
- *Quicksort* ist nur im average-case ein asymptotisch optimales Sortierverfahren
- Es gibt andere Sortieralgorithmen (z.B. *Bucketsort*) mit asymptotisch linearer Laufzeit: die sind aber nicht vergleichsbasiert oder machen einschränkende Annahmen über die Natur bzw. Verteilung der zu sortierenden Elemente