

## Wahrscheinlichkeit eines Vergleichs

Nehmen wir an, dass die sortierte Reihenfolge des vorliegenden zu sortierenden Feldes  $a[1, \dots, n]$  die Folge  $s_1, \dots, s_n$  ist. Also

$$s_1 < s_2 < \dots < s_{n-1} < s_n$$

Wie wahrscheinlich ist es, dass für  $i < j$  das Element  $s_i$  irgendwann im Ablauf unseres Algorithmus mit  $s_j$  verglichen wird?

Offensichtlich ist ein solcher Vergleich nur möglich, solange  $s_i$  und  $s_j$  zusammen in einem zu sortierenden Teil-Array liegen.

Wird ein Pivot  $s_\mu$  mit  $\mu < i$  oder  $\mu > j$  gewählt, ändert sich nichts an der Situation –  $s_i$  und  $s_j$  bleiben im gleichen Teil-Array. Wird ein Pivot  $s_\mu$  mit  $i < \mu < j$  gewählt, wandern die beiden in verschiedene Teil-Arrays und werden nie miteinander verglichen. Nur bei  $s_i$  oder  $s_j$  als Pivot werden die beiden verglichen.

## Resultat für average case

Damit ist klar: die Wahrscheinlichkeit, dass  $s_i$  und  $s_j$  im Verlauf des Sortiervorgangs verglichen werden, ist  $\frac{2}{j-i+1}$ .

Hieraus ergibt sich die erwartete Anzahl von Vergleichen, indem man für alle in Frage kommenden Vergleiche – also für alle Paare  $(i, j)$  mit  $1 \leq i < j \leq n$  – die Wahrscheinlichkeiten dieser Vergleiche aufsummiert.

Der Erwartungswert für die Anzahl Vergleiche  $V(n)$  ergibt sich damit zu

$$V(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} \leq \sum_{i=1}^{n-1} 2 \cdot (H_n - 1) \leq 2n \cdot (H_n - 1)$$

$H_n - 1$  kann durch  $\ln n$  beschränkt werden, daher erhalten wir:

$$V(n) \leq 2n \ln n < 1.39 n \log n.$$

## Heapsort

Ein weiteres schnelles In-place Sortierverfahren ist *Heapsort*. Dabei wird erst das Feld  $a[1, \dots, n]$  in einen **Heap** verwandelt, d.h. es erfüllt für alle  $i$  die Bedingungen

$$a[i] \leq a[2i] \text{ und } a[i] \leq a[2i + 1]$$

(natürlich nur, falls  $2i \leq n$  bzw.  $2i + 1 \leq n$ )

Die bildliche Vorstellung ist hier ein Binärbaum mit Wurzel  $a[1]$ , als Nachfolgeknoten zur Wurzel:  $a[2]$  und  $a[3]$ , etc.

Die Tiefe dieses Baums ist  $\lceil \log n \rceil$ , und im Baum ist jeder Knoten im Wert kleiner (oder gleich) beiden Nachfolgeknoten.

Also befindet sich das Minimum eines Heaps immer in  $a[1]$ .

Damit wird so sortiert:

Tausche  $a[1]$  mit  $a[n]$ , betrachte ab jetzt nur noch  $a[1, \dots, n - 1]$ . „Repariere“ den Heap bei  $a[1]$  und fahre entsprechend fort.

## Die Einsink-Prozedur reheap

Um einen Heap zu erstellen oder eine „gestörte“ Heapbedingung (bei  $a[1]$ ) zu *reparieren*, benötigen wir die Einsink-Prozedur, die auch **reheap** genannt wird. Sie wird rekursiv so realisiert:

**reheap**( $i$ ):

```
if  $2i + 1 > n$  then                                     ← Sonderfall!
  if  $2i = n$  then
    if  $a[i] > a[2i]$  then swap( $a[i], a[2i]$ )
  else
    if  $a[2i] \leq a[2i + 1]$  then                         ← Normalfall!
      if  $a[i] > a[2i]$  then
        swap( $a[i], a[2i]$ ); reheap( $2i$ )
      else
        if  $a[i] > a[2i + 1]$  then
          swap( $a[i], a[2i + 1]$ ); reheap( $2i + 1$ )
```

## Die Heap-Erstellung

Wir lösen zunächst eine kleine Rechenaufgabe:

Es gilt  $\sum_{i=0}^{\infty} \frac{1}{2^i} = 2.$

Das folgt aus  $\sum_{i=0}^k \frac{1}{2^i} = 2 - \frac{1}{2^k}.$

Die rote Gleichung folgt mit Induktion – der Beweis hierzu war an der Tafel.

Sie ist aber auch eine gute Übungsaufgabe...

Wir entnehmen dem Pseudo-Code der Einsink-Prozedur, dass beim Einsinken eines Elements von Höhe  $h$  maximal  $2h$  Vergleichsoperationen ausgeführt werden müssen.

Die Blätter (Höhe 0) müssen nicht einsinken. Auf Höhe 1 gibt es maximal  $n/2$  viele Elemente, auf Höhe 2 maximal  $n/4$  viele, etc.

Also ist **obere Schranke** für Anzahl Vergleiche der Heaperstellung:

$$2 \cdot 1 \cdot \frac{n}{2} + 2 \cdot 2 \cdot \frac{n}{4} + 2 \cdot 3 \cdot \frac{n}{8} + \dots \leq 2n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} = 4n \in O(n).$$

## Aufwand für Heapsort

Im wesentlichen muss nach Heap-Erstellung jedes der  $n$  Elemente einmal von ganz oben nach ganz unten sinken, d.h. man braucht für den Sortier-Teil maximal  $2n \log n$  Vergleiche.

Damit gilt für die Gesamtzahl  $V(n)$  der Vergleiche bei Heapsort:

$$V(n) \leq 2n \log n + O(n)$$

Die gute Nachricht: Diese Schranke gilt für den worst case, d.h. anders als bei Quicksort erreichen wir hier *immer* die bis auf Konstanten optimale Laufzeit  $O(n \log n)$ .

Die schlechte Nachricht: Die Konstante 2 vor dem  $n \log n$  gilt auch im average case, d.h. im Mittel ist dieses Verfahren dem Quicksort-Verfahren deutlich unterlegen, also praktisch nicht konkurrenzfähig.