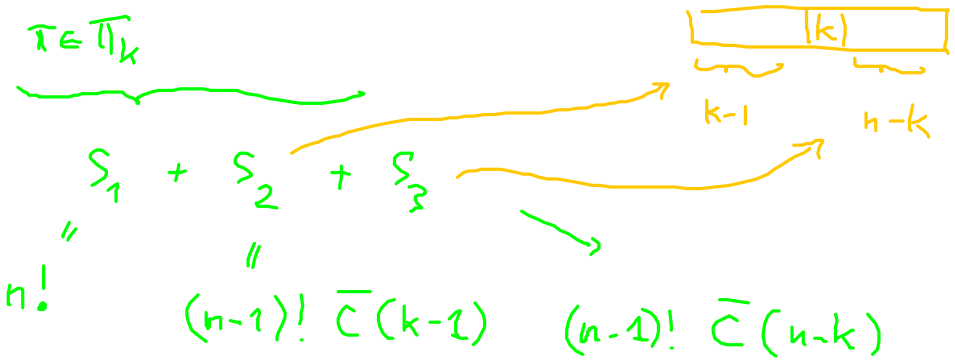
VL diese Woche: D_1, D_2, D_3

UE " T_5

Quicksort

$$\bar{C}(n) = \frac{1}{n!} \sum_{k=1}^{n} \sum_{\pi \in \Pi_k} C(\pi)$$



$$= \frac{1}{n!} \sum_{k=1}^{n} \left[n! + (n-1)!\ \bar{C}(k-1) + (n-1)!\ \bar{C}(n-k) \right]$$

$$= n + \frac{1}{n} \sum_{k=1}^{n} \bar{C}(k-1) + \frac{1}{n} \sum_{k=1}^{n} \bar{C}(n-k)$$

$\bar{C}(n-1) + \cdots + \bar{C}(0)$

$\bar{C}(0) + \bar{C}(1) + \cdots + \bar{C}(n-1)$

$= 0 \qquad = 0$

$$= n + \frac{2}{n} \sum_{k=1}^{n} \bar{C}(k-1)$$

$$\bar{C}(n) \leq n + \frac{2}{n} \sum_{k=2}^{n-1} \bar{C}(k)$$

$$\bar{C}(2) = 1$$

Rekursionsgleichung für den mittleren Aufwand von Quicksort

Lemma: Für die Lösung $T(n)$ der Rekursionsgleichung

$$T(n) = n + \frac{2}{n} \sum_{k=2}^{n-1} T(k) \quad \text{mit} \quad T(0) = T(1) = 0, \; T(2) = 1$$

gilt: $T(n) \leq 2 \cdot n \cdot \ln n$

Beweis: vollst. Ind. nach n

Ansatz: $T(n) \leq c \cdot n \ln n$

↑ bestimmen im Beweis

Ind. Anf.

$n = 2 \qquad T(2) = 1$

$c \cdot n \ln n = c \cdot 2 \cdot \ln(2) \approx c \cdot 1{,}39$ } Ind. Anfang erfüllt für $c \geq 1$

Ind. Vor. für $k = 2, 3, \ldots, n-1$ sei $T(k) \leq c \cdot k \cdot \ln k$

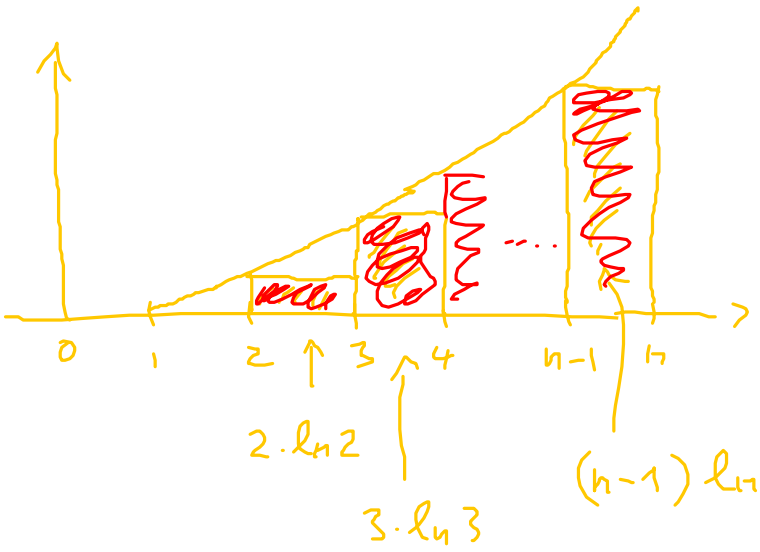
Schluss auf n

$$T(n) \overset{\text{Def}}{=} n + \frac{2}{n} \sum_{k=2}^{n-1} T(k) \overset{\text{IV}}{\leq} n + \frac{2}{n} \sum_{k=2}^{n-1} c \cdot k \ln k$$

$$= n + \frac{2c}{n} \sum_{k=2}^{n-1} k \ln k$$

? wie berechnen ? $\stackrel{\wedge}{=}$ rote Fläche

Trick aus Analysis: Betrachte Fkt $f(x) = x \cdot \ln x$



$$\text{rote Fläche} \leq \int_2^n x \ln x \, dx$$

$$= \frac{x^2}{2} \ln x \Big|_2^n - \int_2^n \frac{x}{2} \, dx$$

$$= \frac{n^2}{2} \ln n - 2 \ln 2 - \left(\frac{n^2}{4} - 1 \right)$$

$$1 - 2 \ln 2 \leq 0$$

$$\leq \frac{n^2}{2} \ln n - \frac{n^2}{4}$$

$$T(n) \leq n + \frac{2c}{n} \left[\frac{n^2}{2} \ln n - \frac{n^2}{4} \right]$$

$$= n + \frac{2c}{2} n \ln n - \frac{2c}{4} n$$

$$= c \cdot n \ln n + \underbrace{n - \frac{c}{2} n}$$

$$=: R(n)$$

$R(n) \leq 0$

$\Leftrightarrow n \leq \frac{c}{2} n$

$\Leftrightarrow 2 \leq c \qquad n \geq 2$

$\leq c \cdot n \ln n$ für $c \geq 2$ □

Satz: Für mittlere Anzahl von Vergleichen $\bar{C}(n)$ beim Sortieren beliebiger n-elem. Arrays gilt

$$\bar{C}(n) \in O(n \log n)$$

Beweis: $\bar{C}(n) \leq A(n) \leq 2 \cdot n \ln n = 2n \frac{\log n}{\log e}$

$$= \underbrace{\frac{2}{\log e}} \cdot n \log n \in O(n \log n)$$

$\approx 1{,}386$

mittlere
Anzahl der Zuweisungen $\bar{A}(n)$ ähnlich $A(n) \in O(n \log n)$

10.5 Heapsort

Heap (Priority Queue) = abstrakte Datenstruktur
mit folgenden Kennzeichen:

Wertebereich: Menge von Werten eines homogenen Komponententyps
Alle Komponenten besitzen einen Schlüssel
[bei Sortieren: Werte ≙ Array Komp. vec[i]
Schlüssel ≙ vec[i].key]

Operationen:

1. Einfügen einer Komponente
2. Zugriff auf Komp. mit größtem Schlüssel
3. Entfernen der Komp. mit größtem Schlüssel
4. Ändern des Schlüssels einer Komponente

Grobstruktur von Heapsort:

1. Gegeben sei Array vec mit n Komponenten
2. Initialisiere den Heap mit Komponenten von vec — $O(n)$
   (alle einfügen)
3. for i := n-1 downto 0 do — n mal
   3.1 Greife auf das größte Element im Heap zu — $O(1)$
   3.2 Weise diese Komponente der Array Komp. vec[i] zu

3.3. Entferne das größte Element im Heap $O(1)$

$O(\log n)$

$\Sigma\ O(n \log n)$

Klar das dies also das Array sortiert

ähnelt Selection Sort mit zusätzlicher Datenstruktur "Heap"

Hier speziellen Heap implementieren

- Initialisierung $O(n)$
- Zugriff auf größte Element in $O(1)$
- Entfernen des größten in $O(\log n)$

[Indem Schlüssel nicht benötigt]

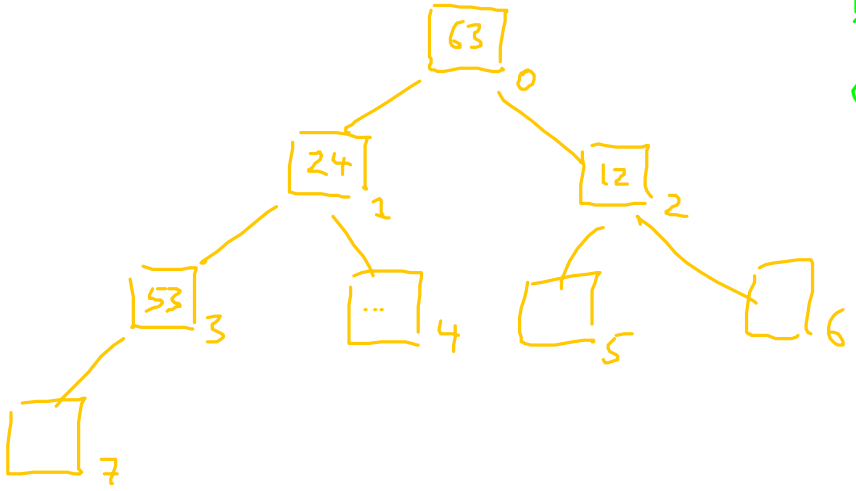
SATZ Mit dieser Heap - Implementation benötigt Heapsort $O(n \log n)$ Aufwand (Vergleiche und Zuweisungen)

Implementation des Heaps:

Hauptidee: Array als Baum verstellen

63	24	12	53	72	18	44	35
0	1	2	3	4	5	6	7

Baum:



Baum schichtenweise voll machen

Lemma: a) vec[0] ist Wurzel des Baumes

b) linker Sohn von vec[i] ist vec[2i+1] (falls vorhanden)

c) rechter ---- ... vec[2i+2] (---)

d) nur Knoten vec[i] mit $i \leq \lfloor \frac{n}{2} \rfloor - 1$ haben Söhne

Dieser Baum erfüllt Heapeigenschaft (ist ein Heap)

wenn gilt vec[i].key $\geq$ vec[2i+1].key falls $2i+1 < n$

... $\geq$ vec[2i+2].key $2i+2 < n$

Vater $\geq$ Söhnen

Heap = Array als Baum mit Heapeigenschaft

Lemma: Erfüllt vec die Heapeigenschaft, so gilt

a) vec[0].key ist der größte Schlüsselwert

b) Entlang eines jeden Weges von Wurzel zu Blatt

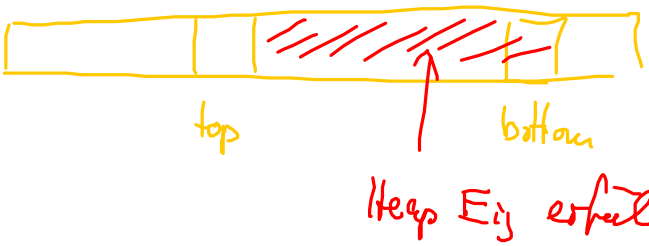
sind die Schlüssel absteigend sortiert

Wie Heapeigenschaft herstellen?

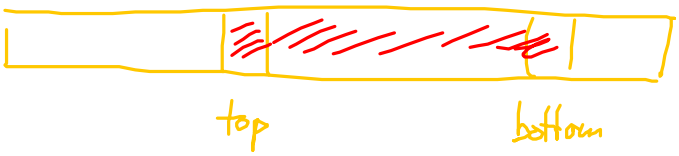
Methode heapify()

Voraussetzung: Heapeigenschaft gilt im Bereich

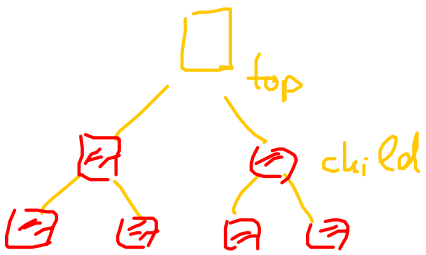
vec[top+1] .... vec[bottom]



Aufruf von heapify(vec, top, bottom) stellt dann die Heapeigenschaft im Bereich von top .... bottom her



Baumvorstellung



heapify():

1. Ermittle größeren der beiden Söhne von top, dies sei child
   (falls kein Sohn existiert, so ist Heap Ei. erfüllt
   falls nur ein Sohn existiert, so nehme diesen

2. Vergleiche vec[child].key mit vec[top].key
   Falls child den größeren Wert hat, so tausche
   vec[child] und vec[top]

3. Wende heapify rekursiv auf child an
   [Aufruf heapify(vec, child, bottom)]