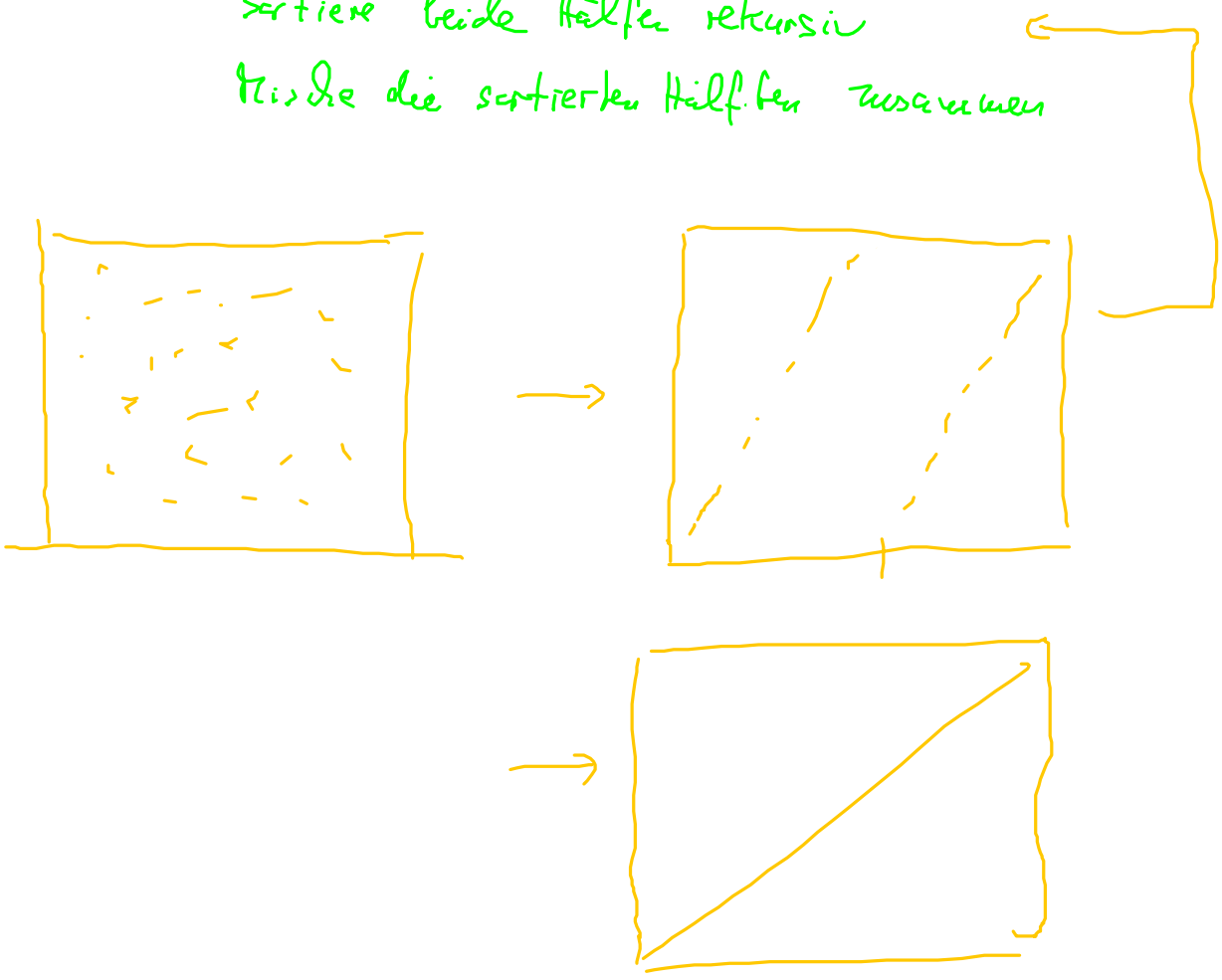
Weiter mit Sortieralgorithmen

Bisher am besten: Mergesort

↓

rekursiver Sortieralgo

Idee: Teile Array in 2 Hälften
Sortiere beide Hälften rekursiv
Mische die sortierten Hälften zusammen



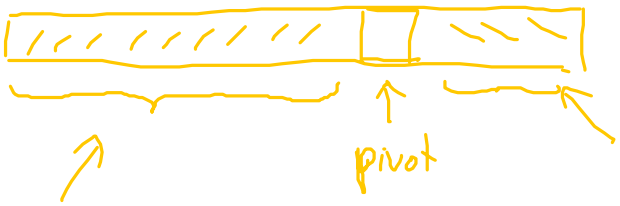
10.4 Quicksort "King" unter den Sortieralgos

rekursiv, anders als Mergesort

1. Ges. Array vec mit n Komponenten

2. Wähle eine beliebige Komponente vec[pivot]

3. Zerlege vec in 2 Teile

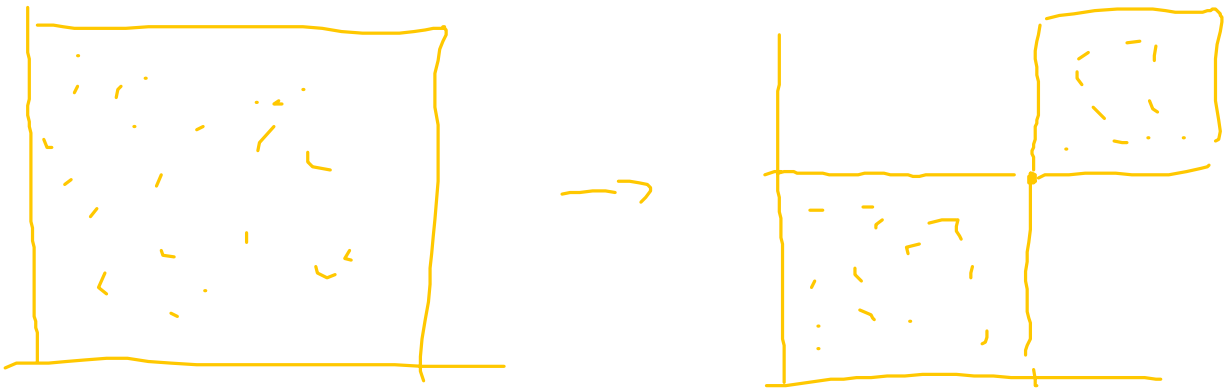


hier ist
vec[i].key > vec[pivot].key

hier ist
vec[i].key ≤ vec[pivot].key

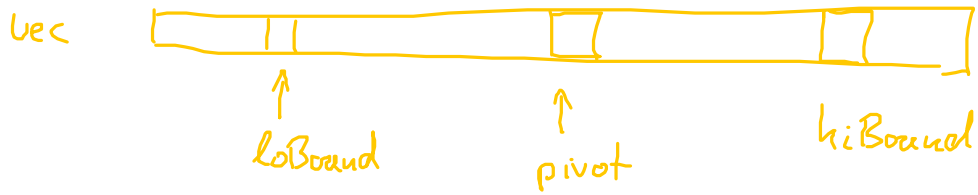
4. Sofern ein Teilbereich aus mehr als 1 Komponente besteht, so wende Quicksort auf ihn per Rekursion an

Korrektheit folgt sofort durch Induktion, wenn die Zerlegung korrekt funktioniert



Zerlegung

muss für beliebige Teilbereiche des Arrays gemacht werden,
dafür Grenzen lo und hi angeben

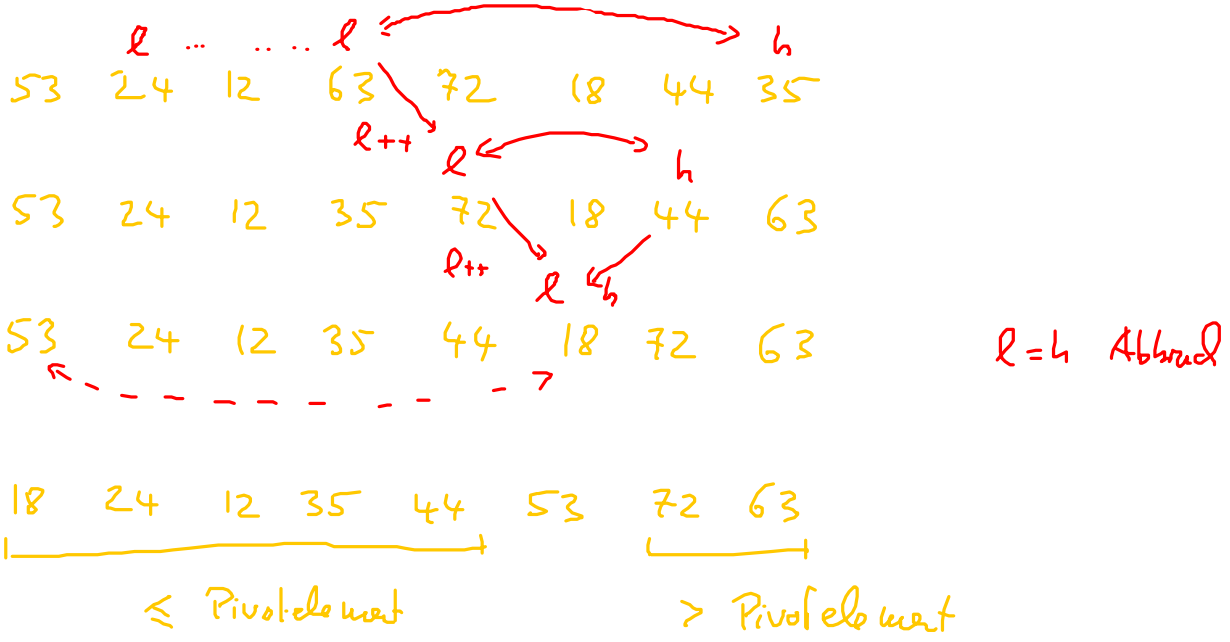


1. Wähle pivot im Bereich lo ... hi
2. Tausche vec[pivot] mit vec[lo]

3. Richte zusätzliche Indexzeiger loSwap hiSwap
   l r
4. Inkrementiere l solange
   bis vec[l].key > vec[loBound].key
   ↖ Wert Pivotelement
5. Dekrementiere h solange
   bis vec[h].key ≤ Wert Pivotelement
6. Falls l < h, so vertausche vec[l], vec[h]
   l++; h--;
7. Wiederhole Schritte 4 - 6 solange bis l ≥ h
8. Tausche vec[loBound] und vec[h]

pivot
↓

Bsp.: 63 24 12 53 72 18 44 35



Warum steht nach Austausch von vec[lo Bound] und vec[h] das Pivotelement an der richtigen Stelle?

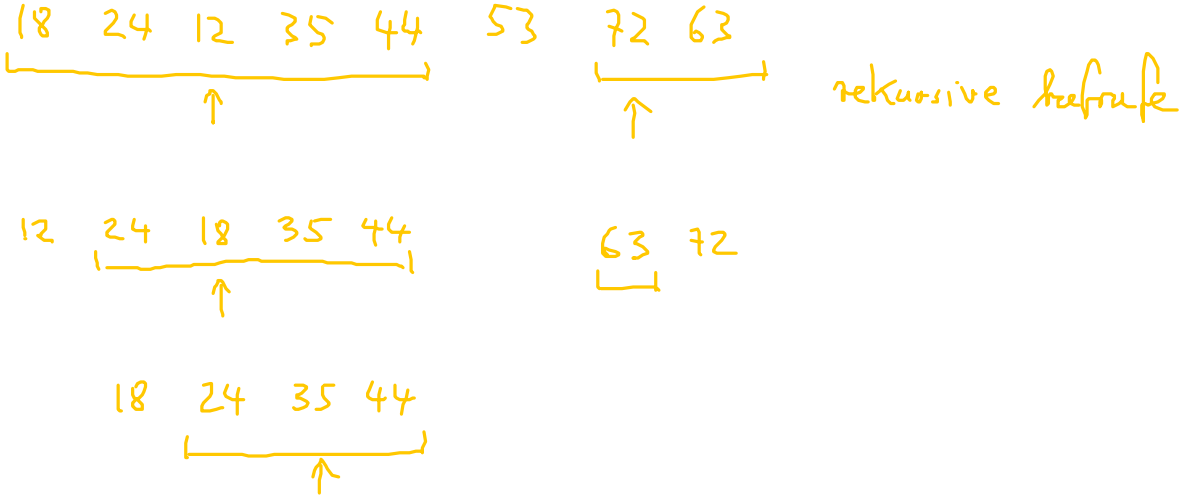
h wird dekrementiert solange vec[h].key ≥ Pivotelement.

Für Indices j > h war entweder vec[j].key > Pivotel.

oder nach Austausch steht dort (wegen vec[l].key > Pivot) ein Element ≥ Pivot

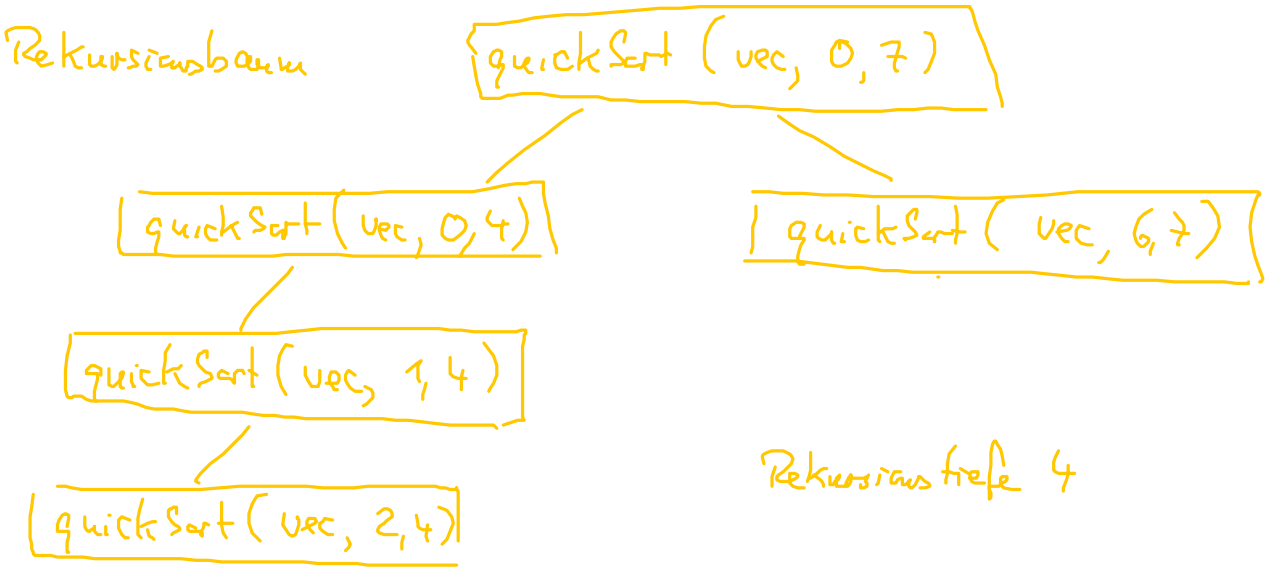
analog: Links von l stehen nur Elemente ≤ Pivotelement

Fortsetzung Beispiel



24 35 44

Rekursionsbaum



Rekursionstiefe 4

Rekursionsbaum kann nur Liste entarten, wenn das Pivotelement immer das kleinste oder das größte ist

⇒ Rekursionstiefe ist dann n

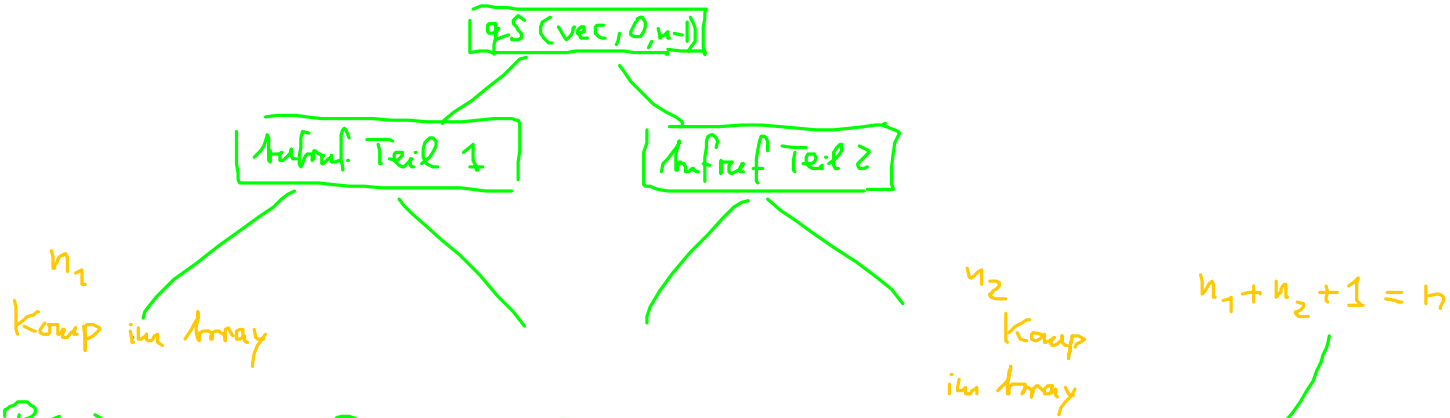
SATZ: Anzahl $R(n)$ rekursiver Aufrufe erfüllt $R(n) \leq n$

Beweis durch Induktion:

I.A. $n = 1$ kein rekursiver Aufruf $R(1) = 0$

I.V. Beh. sei richtig für $1, 2, 3, \ldots n-1$

Schluss auf n:



$$R(n) = 1 + R(n_1) + R(n_2)$$

$$\overset{IV}{\leq} 1 + n_1 - 1 + n_2 - 1$$

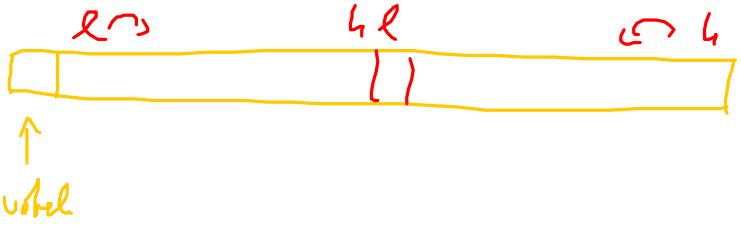
$$= n_1 + n_2 - 1$$

$$< n - 1 \quad \square$$

Vergleiche

für Zerlegung in die beiden Teilarrays

bei n Komponenten



jede Komponente wird mit Pivotelement verglichen

maximal $n-1$ Vergleiche

pivot = kleinstes Element

wieder zerlegen

n-2

Im Wost Case $n-1 + n-2 + n-3 + \dots + 1$ Vergleiche

$= \frac{n(n-1)}{2} \in \Omega(n^2)$

so schlecht wie Bubblesort

Fkt $C(n) : \mathbb{N} \to \mathbb{N}$

$C(n) = \frac{n(n-1)}{2}$

Wost Case erfordert mindestens $\frac{n(n-1)}{2}$ Vergleiche

Warum ist Quicksort dann praktisch so gut?

Grund: Wost Case tritt nur selten auf

Werden Quicksort gerechter mit einer Analyse des mittleren Aufwands

dazu: nehmen an: Werte der Komponenten sind $1, 2, \dots, n$

nehmen an jede mögliche Reihenfolge ("Permutation")

dieses Werk wird an Quicksort übergeben

summieren alle Vergleiche auf

und teilen sie durch Anzahl der Eingaben

$$\overline{C}(n) = \frac{1}{n!} \sum_{\pi \in \Pi} C(\pi)$$

$\overline{C}(n)$: mittlerer Aufwand bei n Komponenten

$C(\pi)$: Aufwand von Quicksort für Permutation π (konkrete Eingabe) gemessen in # Vergleiche

Π: Menge aller Permutationen von $1, \ldots, n$

$n!$ = # der Permutationen von $1, \ldots, n$

Ergebnis $\overline{C}(n) \in O(n \log n)$ $\Rightarrow$ Quicksort im Mittel gut

empirisch besser als Insertsort, Heapsort

Sei $\Pi_k := \{ \pi \in \Pi \mid \text{Pivotelement hat Wert } k \}$

$\Pi = \Pi_1 \cup \Pi_2 \cup \ldots \cup \Pi_n$ ← disjunkte Vereinigung

$n = 3$ $\Pi = \{ 123, 132, 213, 231, 312, 321 \}$

$\Pi_1 = \{ 213, 312 \}$ $\Pi_2 = \{ 123, 321 \}$ $\Pi_3 = \{ 132, 231 \}$

$|\Pi_k| = (n-1)!$

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

hier Pivotelement der Weit k

$\Rightarrow$ besser aus zu rechnen

$$C(\pi) = Z(\pi) + C(\pi_1) + C(\pi_2)$$

↑ Zerlegung

Permutation von $1, \dots, k-1$

Permutation von $k+1, \dots, n$

$$\Rightarrow \sum_{\pi \in \Pi_k} C(\pi) = \sum_{\pi \in \Pi_k} Z(\pi) + \sum_{\pi \in \Pi_k} C(\pi_1) + \sum_{\pi \in \Pi_k} C(\pi_2)$$

$=: S_1$ $=: S_2$ $=: S_3$

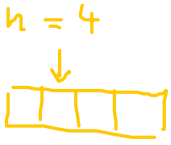
$Z(\pi) \leq n$

$$S_1 \leq \sum_{\pi \in \Pi_k} n = n \underbrace{(n-1)!}_{|\Pi_k|} = n!$$

Beispiel für S_2

$n = 4$ $\quad k = 3$ $\quad \pi_k = \{ 1324, 1342, 2314, 2341$

$4312, 4321 \}$



Zerlegung

jede Permutation π_1 entsteht gleich häufig

$$S_2 = \sum_{\pi \in \pi_k} C(\pi) = \frac{(n-1)!}{(k-1)!} \sum_{\pi_1 \text{ Permutation um } 1,2,\dots,k-1} C(\pi_1)$$

$(n-1)!$ Summanden

$(k-1)!$ Summanden

$$= (n-1)! \boxed{\frac{1}{(k-1)!} \sum_{\pi_1 \text{ Perm um } 1 \dots k-1} C(\pi_1)}$$

$$\bar{C}(k-1)$$

$$= (n-1)!\, \bar{C}(k-1)$$