

Weiter mit Sortieralgorithmen

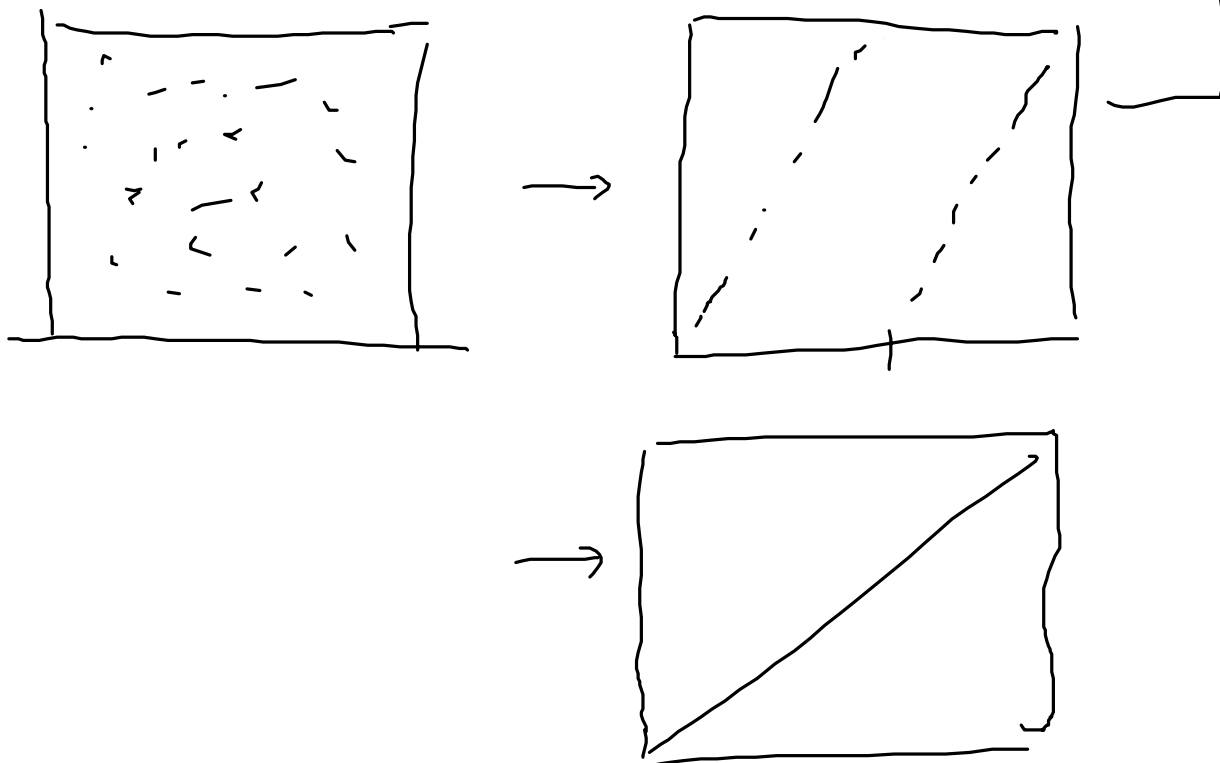
Bisher am besten: Mergesort

rekursiver Sortieralgo

Idee Teile array in 2 Hälften

Sortiere beide Hälften rekursiv

Mische die sortierten Hälften zusammen



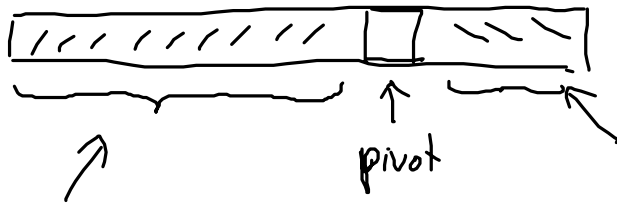
10.4 Quicksort "King" unter den Sortieralgos

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

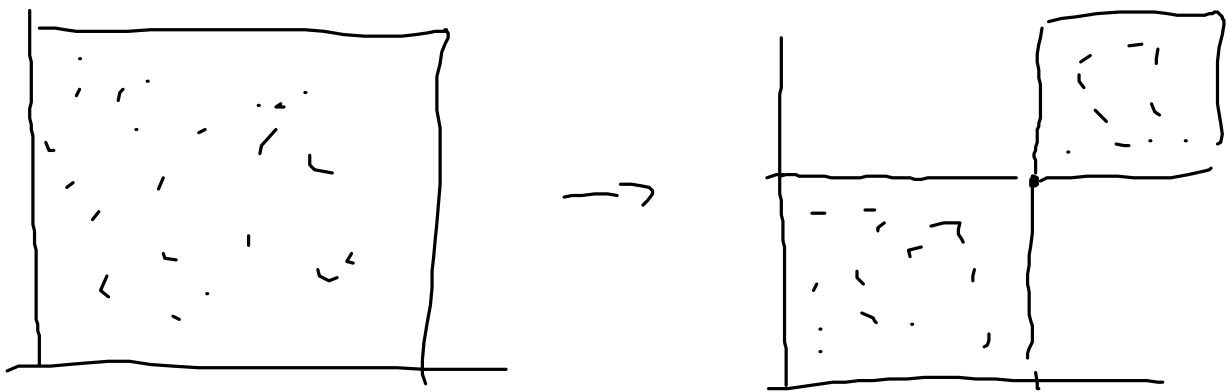
$vec[i].key \leq vec[pivot].key$

hier ist

$vec[i].key > vec[pivot].key$

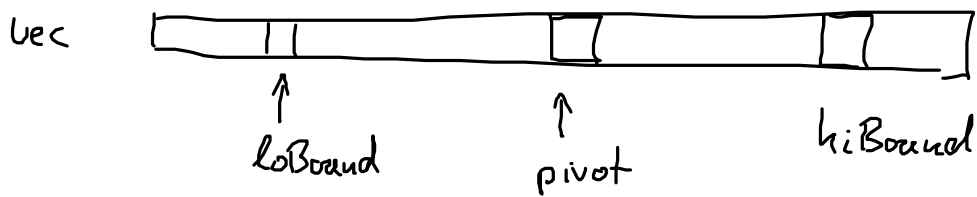
4. Sofern ein Teilbereich aus mehr als 1 Komponente besteht, so werde Quicksort auf ihn per Rekursion 94

Korrektheit folgt sofort durch Induktion, wenn die Zerlegung korrekt funktioniert



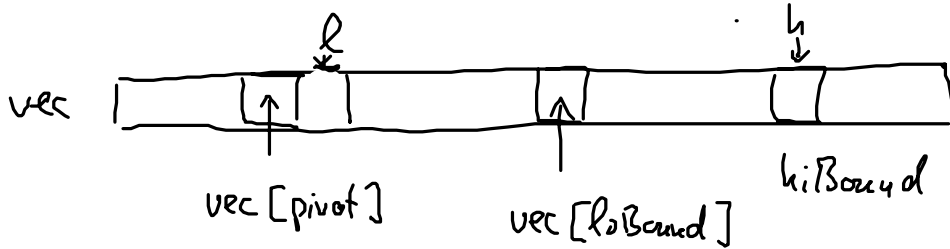
Zerlegung

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



1. Wähle pivot im Bereich $l_0 \dots l_i$

2. Tausche $vec[pivot]$ mit $vec[l_0]$



3. Richte zunächst beide Index zeiger l_{0Swap} l_iSwap
 l r

4. Inkrementiere l solange

bis $vec[l].key > vec[l_0Bound].key$

↖ Wert Pivotelement

5. Dekrementiere h solange

bis $vec[h].key \leq$ Wert Pivotelement

6. Falls $l < h$, so vertausche $vec[l]$, $vec[h]$

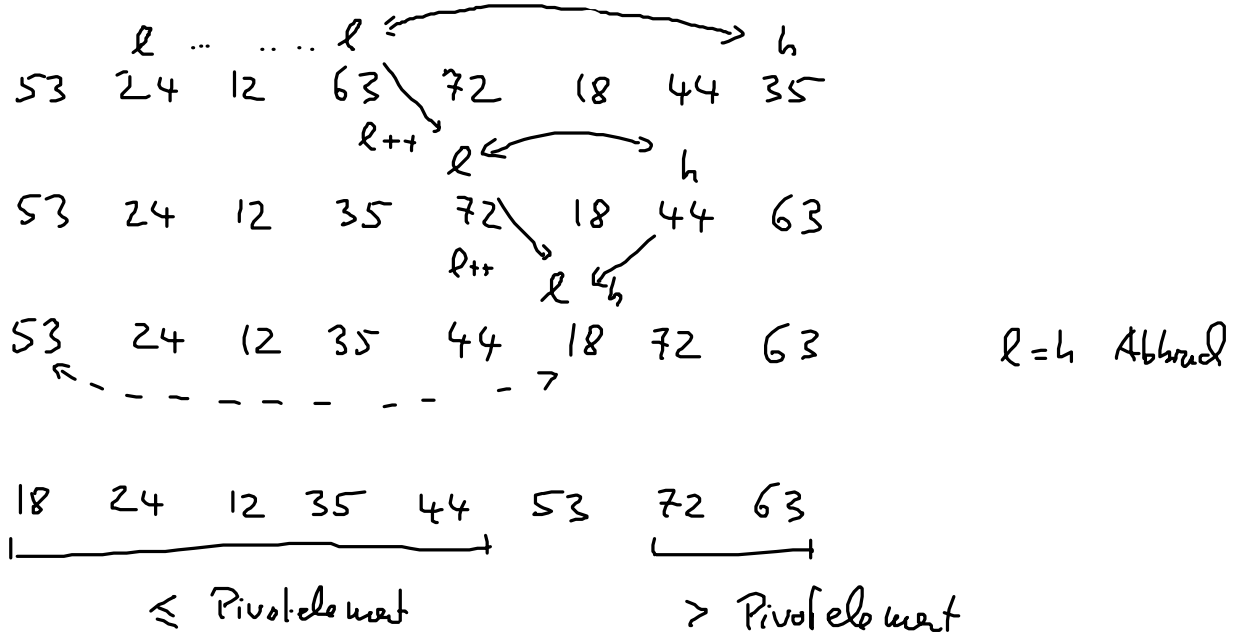
$l++$; $h--$;

7. Wiederhole Schritte 4-6 solange bis $l \geq h$

8. Tausche $vec[l_0Bound]$ und $vec[h]$

pivot
↓

Bsp.: 63 24 12 53 72 18 44 35



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

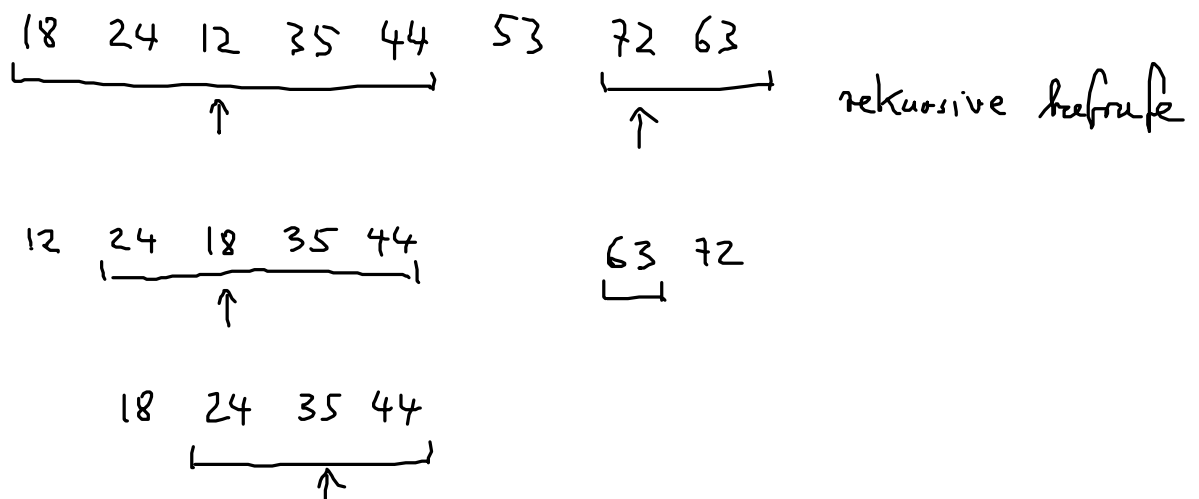
h wird dekrementiert solange $vec[h].key \geq$ Pivotelement.

Für Indizes $j > h$ war entweder $vec[j].key \geq$ Pivot.

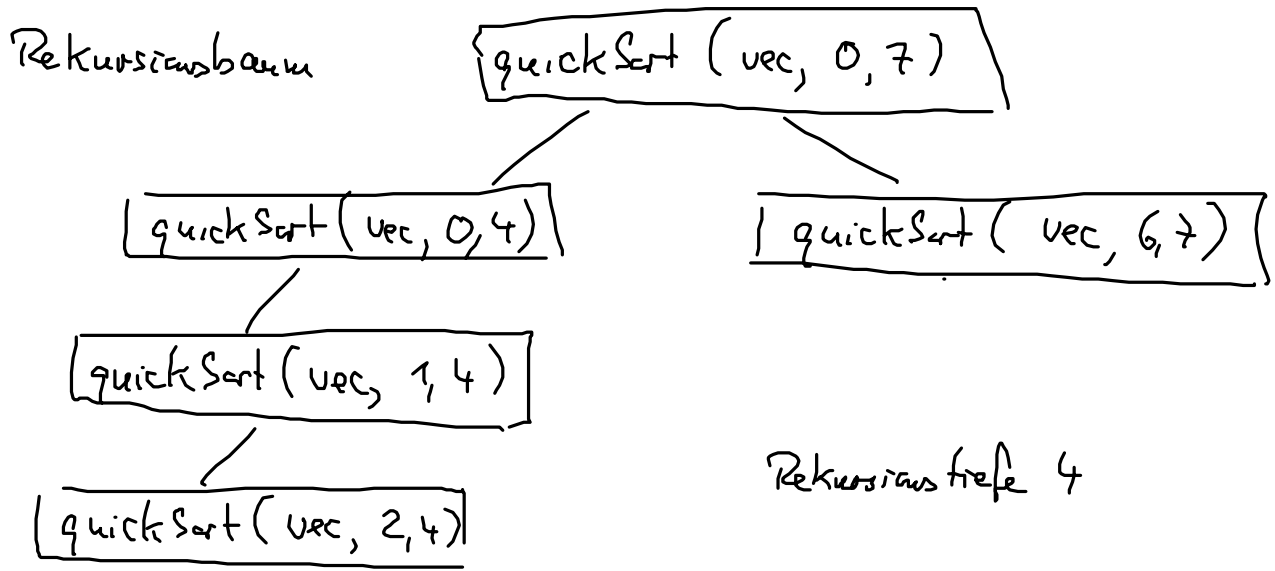
oder nach Austausch steht dort (wegen $vec[l].key >$ Pivot) ein Element \geq Pivot

analog: links von l stehen nur Elemente \leq Pivotelement

Fortsetzung Beispiel



24 35 44
└─┬─┘ └─┬─┘



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

⇒ Rekursionstiefe ist dann n

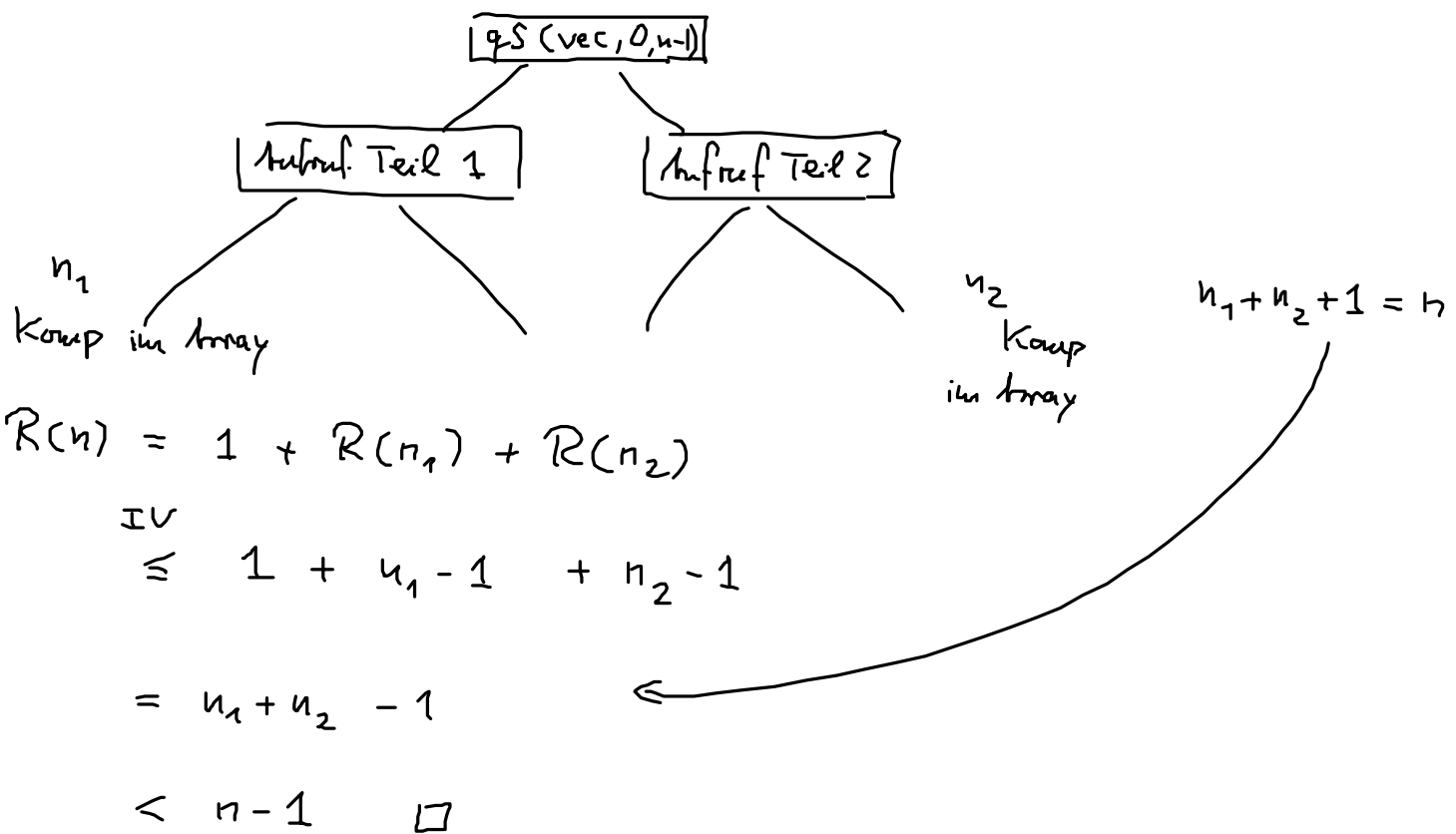
SATZ: Anzahl $R(n)$ rekursiver Aufrufe erfüllt $R(n) \leq n$

Beweis durch Induktion:

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

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

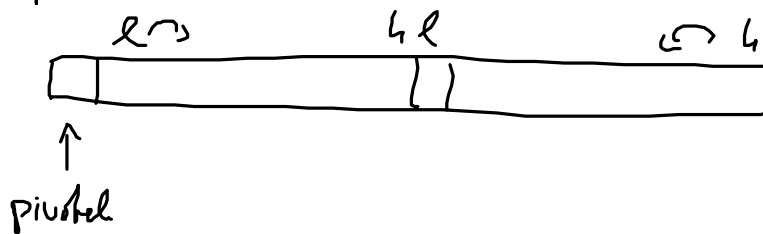
Schluss auf n :



Vergleiche

für Zerlegung in die beiden Teilarrays

bei n Komponenten



jede Komponente wird mit Pivotelement verglichen
maximal $n-1$ Vergleiche



pivot = kleinstes
Element

$n-1$

wieder zerlegen

⋮
n-2
⋮

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

/

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

↑
so schlecht wie Bubblesort

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

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

Worst Case erfordert

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

Warum ist Quicksort dann praktisch so gut?

Grund: Worst Case tritt nur selten auf

Werden Quicksort gerechter mit einer Analyse des wittleren Aufwands

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

nehmen an jede mögliche Reihenfolge ("Permutation")
diese Werte wird an Quicksort übergeben
summieren alle Vergleiche auf
und teilen sie durch Anzahl der Eingaben

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

↑
 mittlerer Aufwand bei n Komponenten

↑
 Menge aller Permutationen von $1, \dots, n$

↑
 Aufwand für Permutation π (konkrete Eingabe) gemessen in # Vergleiche

↑
 von Quicksort

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

Ergebnis $\bar{C}(n) \in O(n \log n)$ \Rightarrow Quicksort im Mittel gut empirisch besser als Bubblesort, Heapsort

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

$\Pi = \Pi_1 \cup \Pi_2 \cup \dots \cup \Pi_n$ \leftarrow disjunkte Vereinigung

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

$\boxed{1|2|3}$

↑

Pivot

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

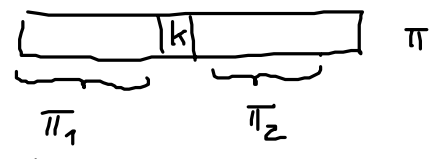
$|\Pi_k| = (n-1)!$

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

hier Pivotelement
den Wert k
=> besser ausrechnen

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

↑
Zerlegung



Permutation von 1, ..., k-1 Permutation von k+1, ..., n

$$\Rightarrow \sum_{\pi \in \Pi_k} C(\pi) = \underbrace{\sum_{\pi \in \Pi_k} Z(\pi)}_{=: S_1} + \sum_{\pi \in \Pi_k} C(\pi_1) + \sum_{\pi \in \Pi_k} C(\pi_2)$$

$\leq n$

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

Beispiel für S_2

