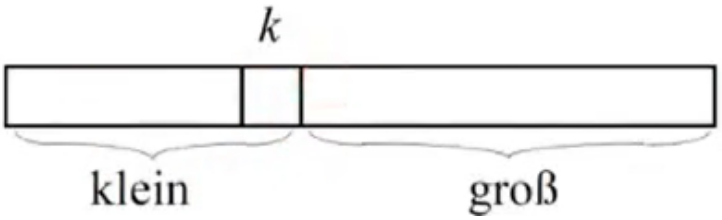
Eigenschaften

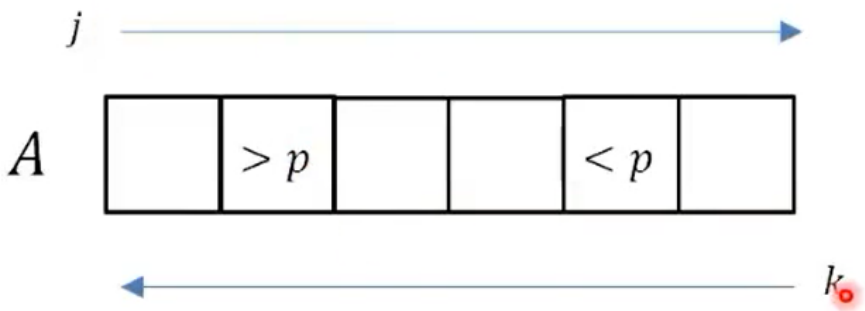
- [[Divide & Conquer]] Algorithmus
- in-place
- Verfahren
  - zerlege Feld in zwei Felder
    - linke Feld mit Elementen $\leq$ Elementen in rechtem Feld
  - Feld weiterzerlegen
    - QuickSort rekursiv aufrufen

```
QUICKSORT(A, von, bis)
1:  IF von<bis THEN
2:          k ← PARTITION(A, von, bis)
3:          QUICKSORT ( A, von, k)
4:          QUICKSORT (A, k+1, bis)
```



Partitioning

- Pivot-Element p
- sortiere Feld, sodass
  - links Zahlen $\leq$ p
  - rechts Zahlen $>$ p
  - Feld wird von beiden Seiten durchlaufen
    - Elemente vertauschen, wenn Bedingung nicht erfüllt
  - treffen sich beide Indizes in der Mitte
    - fertig

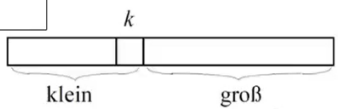


Speichere A so um, dass alle Elemente in A[0..k] kleiner-gleich den Elementen in A[k+1..n-1] sind

```
PARTITION (A, l, r)
1:  p←A[l]
2:  j←l-1, k←r+1
3:  LOOP REPEAT j←j+1 UNTIL A[j]≥p
4:       REPEAT k←k-1 UNTIL A[k]≤p
5:          IF  j<k THEN
6:             vertausche (A[j],A[k])
7:          ELSE
8:             RETURN k
```

p ... Pivotelement (hier: das erste Element)

T(n) ∈ O(n)
S(n) ∈ O(1)

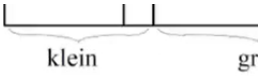


-

Laufzeit

$$T(n) = T(k) + T(n-k) + O(n)$$

linkes Teilfeld — rechtes Teilfeld — PARTITION



- **Bester Fall**: balancierte Aufteilung
  k=n/2 ⇒ **T(n) = O(n*log n)**, S(n) = O(log n)

- **Schlechtester Fall**: A bereits sortiert!
  k=1 ⇒ **T(n) = O(n²)**, S(n) = O(n)

- **Mittlerer Fall**
  T(n) = O(n log n)

-
- Vorteil gegenüber [[MergeSort]]
  - weniger Kopieren => schneller trotz schlechterer T(n)
  - besseres S(n)

Varianten

- Randomisierte Pivotauswahl
  - Wähle Pivotelement als p ← A[random(l,r)]
  - random(l,r) liefert zufällig eine Zahl in {l,...,r} aus einer Gleichverteilung
  - Dadurch wird die Laufzeit **unabhängig von der Inputfolge**.
  - Worst-case Laufzeit ist extrem unwahrscheinlich: $\frac{1}{n!}$
  - ~~erative Variante~~
- Iterative Variante

- Verwendet einen Stapel um Rekursion zu umgehen.
- Hat worst-case Speicherbedarf von O(log n)