# Quicksort Sortieren durch Zerlegen Hoare 1960

Handlungsorientierter Einstieg:
$n$ Personen wollen sich der Größe nach in einer Reihe aufstellen.

Gegeben ist eine Liste mit vergleichbaren, verschiedenen Elementen. Ein Element der Liste (z. B. das erste Element) wird als Vergleichselement ausgewählt. Die übrigen Elemente werden in zwei Teillisten aufgeteilt, deren Elemente kleiner bzw. größer als das Vergleichselement sind. Das Vergleichselement nimmt die Position zwischen den Teillisten ein. Dieser Vorgang wird mit den Teillisten in gleicher Weise fortgesetzt.

$\mathrm{L} = [12, 4, 20, 5, 6, 13, 3, 1, 15]$

$\mathrm{L} = [4, 5, 6, 3, 1] + [12] + [20, 13, 15]$

```
def sort(L):

    kleiner = []
    groesser = []

    if len(L) > 1:
        for x in L:
            if x < L[0]:      #  alternativ  kleiner = [i for i in L if i < L[0]]
                kleiner.append(x)
            if x > L[0]:
                groesser.append(x)
        return sort(kleiner) + [L[0]] + sort(groesser)
    else:                     # falls nur noch ein bzw. kein Element in L ist
        return L
```

```
L=[12,4,20,5,6,13,3,1,15]
print(L)
print(sort(L))
```

Bei der Programmerstellung wird man zunächst
return kleiner + [L[0]] + groesser anstreben.
Ändere das Programm, so dass die Liste auch gleiche Elemente enthalten kann.
$\mathrm{L} = [12, 4, 20, 3, 5, 6, 20, 13, 3, 1, 15]$

```
def sort(L):

    kleiner = []
    gleich = []
    groesser = []

    if len(L) > 1:
        for x in L:
            if x < L[0]:
                kleiner.append(x)
            if x == L[0]:
                gleich.append(x)
            if x > L[0]:
                groesser.append(x)
        return sort(kleiner) + gleich + sort(groesser)
    else:  # falls nur noch ein bzw. kein Element in L ist
        return L


L=[12,4,20,3,5,6,20,13,3,1,15]
print(L)
print(sort(L))
```

komprimiert:

```
def sort(L):
 i=0
 if len(L) > 1:

   return sort([i for i in L if i < L[0]]) + [i for i in L if i == L[0]] + \
          sort([i for i in L if i > L[0]])
 else:
   return L


L=[12,4,20,3,5,6,20,13,3,1,15]
print(L)
print(sort(L))
```

# Quicksort Ablauf

günstiger Fall

L = [20, 110, 7, 10, 80, 2, 150, 14, 25, 100]

[7, 10, 2, 14] [20] [110, 80, 150, 25, 100]

[2] [7] [10, 14] [20] [80, 25, 100] [110] [150]

[2] [7] [10] [14] [20] [25] [80] [100] [110] [150]

ungünstigster Fall

L = [10, 9, 8, 7, 6, 5, 4, 3]

[9, 8, 7, 6, 5, 4, 3] [10]

[8, 7, 6, 5, 4, 3] [9] [10]

[7, 6, 5, 4, 3] [8] [9] [10]

[6, 5, 4, 3] [7] [8] [9] [10]

[5, 4, 3] [6] [7] [8] [9] [10]

[4, 3] [5] [6] [7] [8] [9] [10]

[3] [4] [5] [6] [7] [8] [9] [10]

Die unterstrichenen Zahlen sind jeweils die Pivotelemente.

Im günstigen Fall werden die Längen der Teillisten von Ebene zu Ebene mindestens halbiert. Das heisst, in der ersten Ebene hat eine Teilliste maximal $\frac{n}{2}$ Elemente, in der $k$-ten Ebene maximal $\frac{n}{2^k}$ Elemente.
Wie viele Ebenen sind erforderlich, bis nur noch Teillisten der Länge 1 vorliegen?

$$\frac{n}{2^k} = 1 \quad \Longrightarrow \quad n = 2^k \quad \Longrightarrow \quad k = \log_2(n)$$

Da der Algorithmus pro Ebene nie mehr als $n$ Vergleiche macht, kann man die Laufzeit des Quicksorts im günstigen Fall auf $n \log_2(n)$ Vergleiche nach oben abschätzen.

Ändere das Programm so ab, dass das Pivotelement zufällig gewählt wird.
(Randomisierter Quicksort)