Algorithmen und Programmieren II

Sortieralgorithmen

imperativ

Teil II

Prof. Dr. Margarita Esponda

Freie Universität Berlin

Sortieralgorithmen

Vergleichsalgorithmen
- Bubble-Sort
- Insert-Sort
- Selection-Sort
- Shell-Sort
- Quicksort
- Mergesort
- Heap-Sort

- Counting-Sort
- Radix-Sort
- Bucket-Sort

Bubble-Sort

Einfachster und ältester Sortieralgorithmus

- **In-Place**

  minimaler zusätzlicher konstanter Speicherplatz **O(1)**

- **Stabil**

  die Reihenfolge von gleichen Daten bleibt unverändert

- **zu naiv** und **ineffizient** für das Sortieren von im Speicher zusammenhängenden Informationen
- jedoch eignet er sich für das Sortieren innerhalb **verketteter Listen**.
- quadratischer Aufwand $O(n^2)$

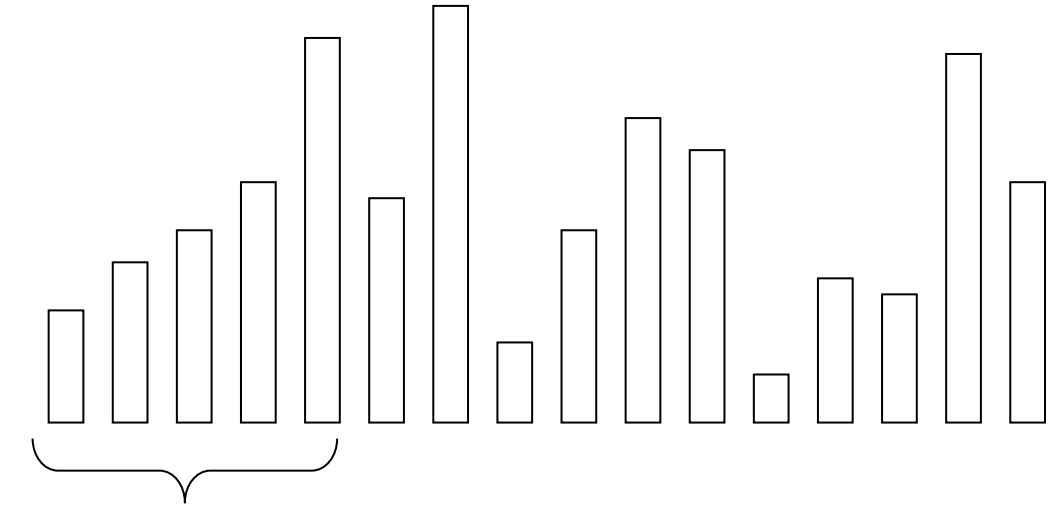
Bubble-Sort

```
def  bubblesort (A):
    swap = True
    stop = len(A)-1
    while swap:
        swap = False
        for i in range(stop):
            if A[i]>A[i+1]:
                A[i], A[i+1] = A[i+1], A[i]
                swap = True
        stop = stop-1
```

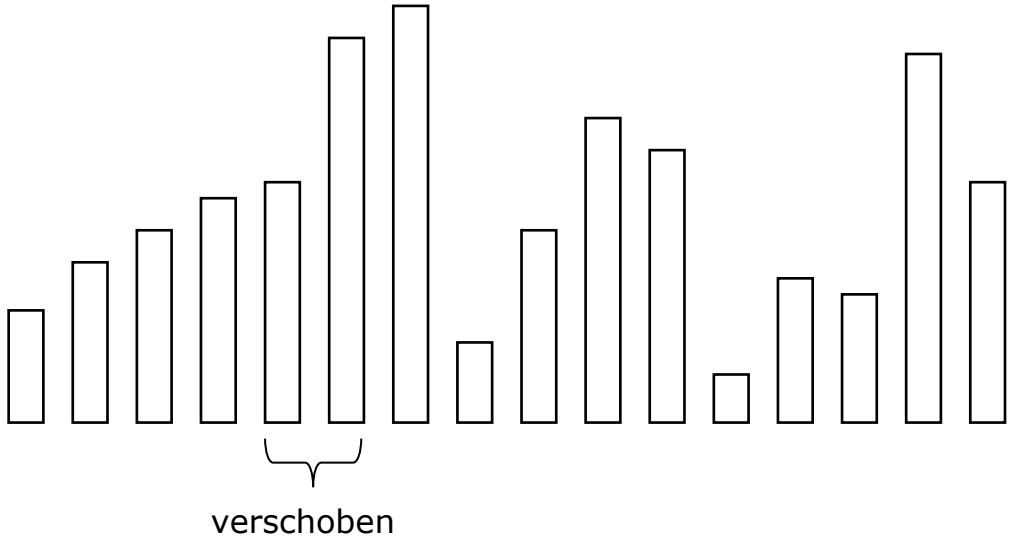
Hilfsvariable für einen linearen Aufwand, wenn die Daten sortiert sind.

Hier wird die Stabilitätseigenschaft des Algorithmus garantiert

Insertion-Sort



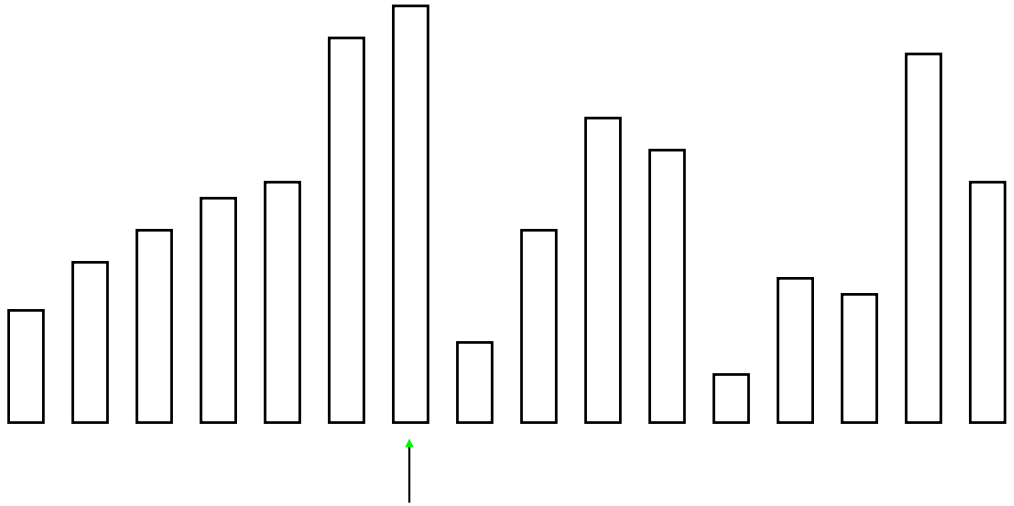
Insertion-Sort



Insertion-Sort

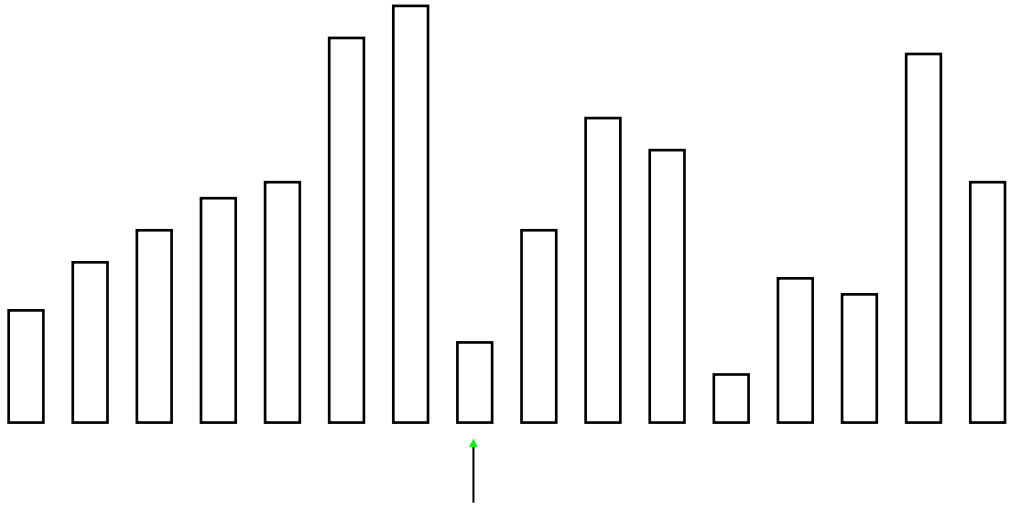
Bester Fall

Größer als alle Elemente auf der linken Seite



Es ist kein weiterer
Vergleich notwendig

Insertion-Sort

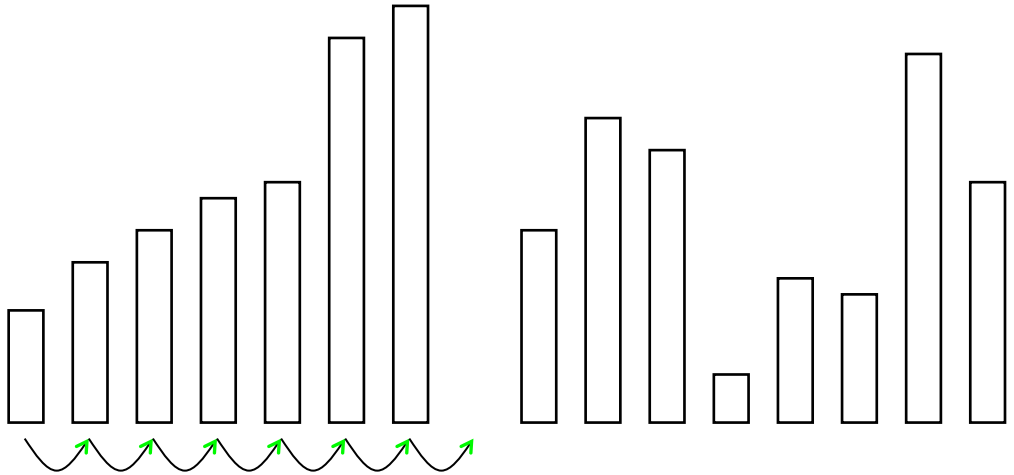


Kleiner als alle Elemente
der linken Seite

Schlimmster Fall

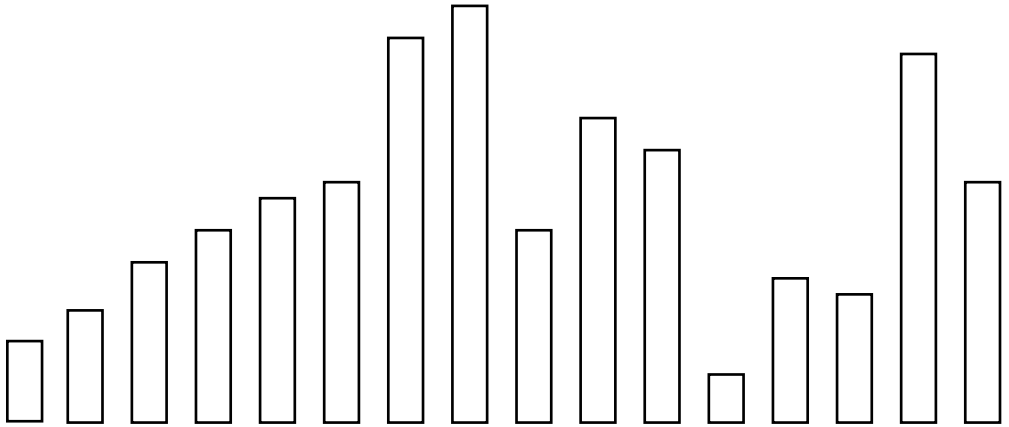
M. Esponda

Insertion-Sort



Alle Elemente müssen verschoben werden

Insertion-Sort



M. Esponda

Insertion-Sort

```
isort :: [ Integer ] -> [ Integer ]

isort [] = []

isort (a:x) = ins a (isort x)


ins :: Integer -> [Integer] -> [Integer]

ins a [] = [a]

ins a (b:y)

        |  a<= b  =  a:(b:y)

        | otherwise = b: (ins a y)
```

Das Problem in Haskell ist vor allem der Speicherverbrauch

Insertion-Sort (imperativ)

Einfacher Sortieralgorithmus

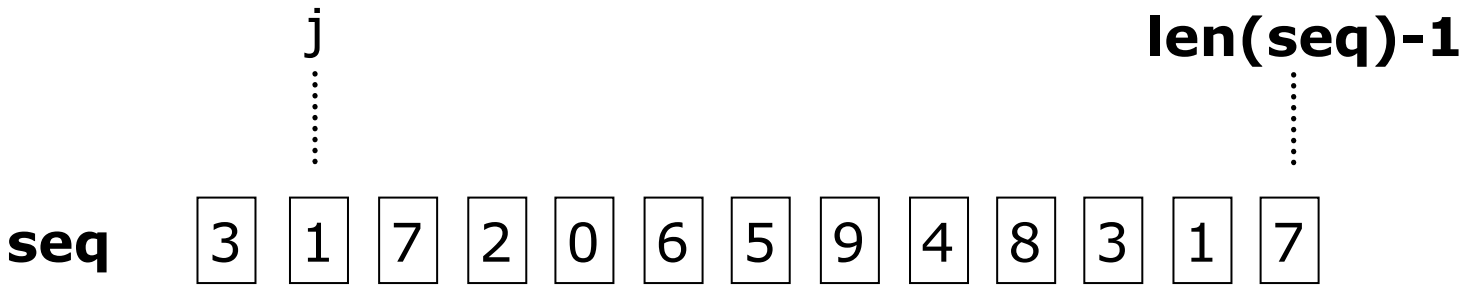
- **In-Place** und kein zusätzlicher Speicherbedarf O(1)
- **Stabil**
- **gut für kleine Mengen** oder **leicht unsortierte Informationen**

```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Eine geeignete Position wird gesucht und die Elemente des sortierten Bereichs verschoben

Die einzusortierende Zahl wird in den gefundenen Platz kopiert

Insertion-Sort (imperativ)

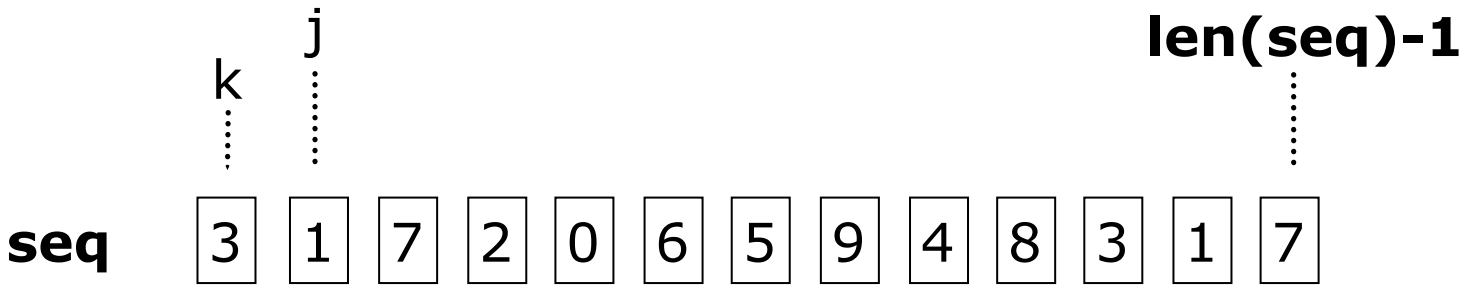


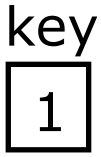

```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

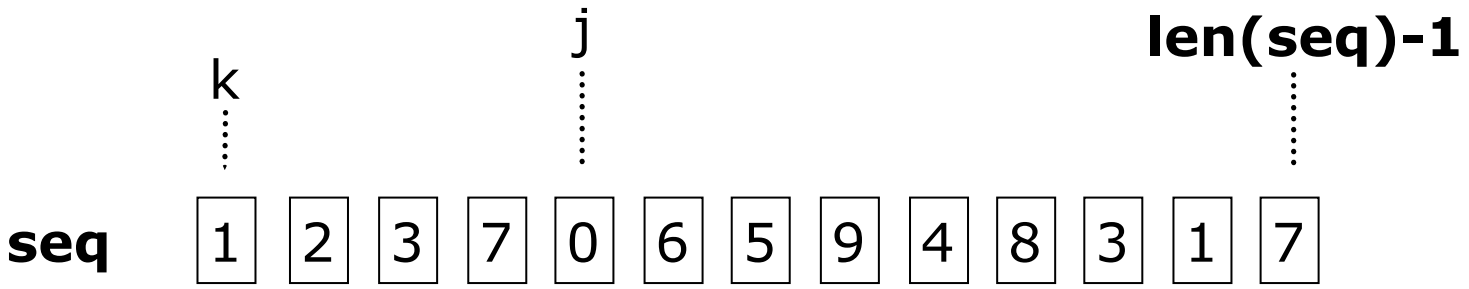
Insertion-Sort (imperativ)






```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)



key

2

```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Laufzeit

	Zeit	Anzahl Max	Anzahl Min
def insertsort(seq):	c_1	1	1
for j in range(1,len(seq)):	c_2	n	n
key = seq[j]	c_3	n-1	n-1
k = j-1;	c_4	n-1	n-1
while k>=0 and seq[k]>key:	c_5	1+2+...+n	n-1
seq[k+1] = seq[k]	c_6	1+2+...+n-1	0
k = k-1	c_7	1+2+...+n-1	0
seq[k+1] = key	c_8	n-1	n-1

Maximale Laufzeit ("worst case")

$$T(n)= c_1 + c_2 n + (c_3+c_4+c_8)(n-1) + c_5 (1+2+..+n) + (c_6+c_7)(1+2+...+(n-1))$$

$$1+2+3+\ldots+n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\frac{n(n+1)}{2} \qquad \frac{(n-1)n}{2}$$

$$T(n) = c_1 + c_2 n + (c_3+c_4+c_8)(n-1) + c_5 n + (c_5+c_6+c_7)(1+2+...+(n-1))$$

$$T(n) = c_1 - c_3 - c_4 - c_8 + (c_2 + c_3 + c_4 + c_8 + c_5)n + (c_5+ c_6+c_7)\left(\frac{(n-1)n}{2}\right)$$

$$T(n) = \underbrace{-(c_1+c_3+c_4+c_8)}_{c} + \underbrace{(c_2+c_3+c_4+c_8+\frac{c_5}{2}-\frac{c_6}{2}-\frac{c_7}{2})}_{b} n + \underbrace{(\frac{c_5}{2}+\frac{c_6}{2}+\frac{c_7}{2})}_{a} n^2$$

$$T(n) = an^2 + bn + c$$

Minimale Laufzeit ("best case")

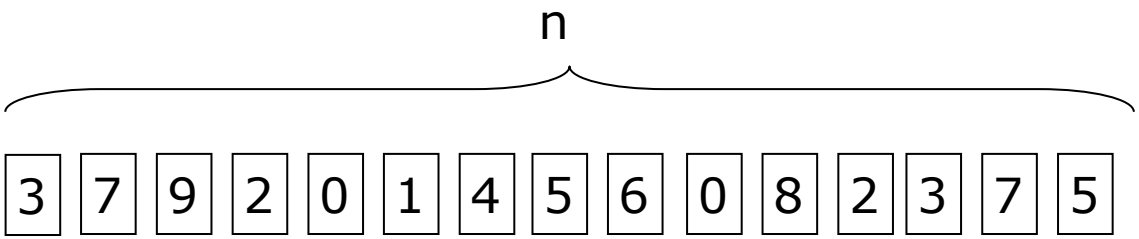
$$T(n) = c_1 + c_2 n + (c_3 + c_4 + c_5 + c_8)(n-1)$$

$$T(n) = \underbrace{(c_1 - c_3 - c_4 - c_5 - c_8)}_{a} + \underbrace{(c_2 + c_3 + c_4 + c_5 + c_8)}_{b}(n)$$

$$T(n) = a + b\,n = \mathbf{O}(n)$$

Insertion-Sort

Eingabe: n Zahlen



Berechnungsschritt: Vergleichsoperation

Im schlimmsten Fall: $T(n) = 1+2+3+...+(n-1)$

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

$$= \tfrac{1}{2}n^2 - \tfrac{1}{2}n$$

$$= c_1 n^2 + c_2 n = \boxed{\mathbf{O}(n^2)}$$

M. Esponda