Sortieren durch Einfügen

Gegeben: eine Folge von ganzen Zahlen.

Gesucht: die zugehörige aufsteigend sortierte Folge.

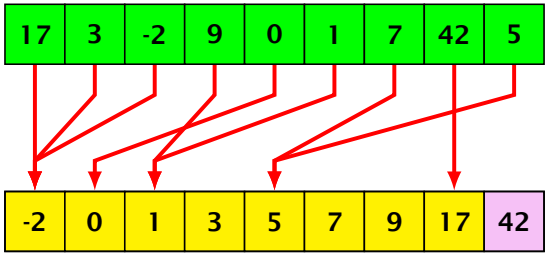
Idee:

- speichere die Folge in einem Feld ab;
- lege ein weiteres Feld an;
- füge der Reihe nach jedes Element des ersten Felds an der richtigen Stelle in das zweite Feld ein!

$\Longrightarrow$ Sortieren durch Einfügen (InsertionSort)

Beispiel

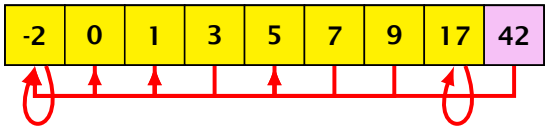
Animation ist nur in der Vorlesungsversion der Folien vorhanden.



Beispiel

Animation ist nur in der Vorlesungsversion der Folien vorhanden.

Wir brauchen kein zweites Array:



Algorithmus: Insertion Sort

```
1  Input: Array A mit n Elementen
2  Output: Array A aufsteigend sortiert
3
4  InsertionSort(A)
5      j = 0;
6      while (j < n)
7          key = A[j];
8          i = j-1;
9          while (i >= 0 && A[i] > key)
10             A[i+1] = A[i];
11             i--;
12         A[i+1] = key;
13         j++;
```

InsertionSort

Laufzeit InsertionSort

Kostenübersicht		
Zeile	*Kosten*	*Anzahl*
5	$\mathcal{O}(1)$	1
6	$\mathcal{O}(1)$	$n+1$
7	$\mathcal{O}(1)$	$n-1$
8	$\mathcal{O}(1)$	t_j
9	$\mathcal{O}(1)$	t_j-1
10	$\mathcal{O}(1)$	t_j-1
11	$\mathcal{O}(1)$	n
12	$\mathcal{O}(1)$	n

```
1  Input: Array A mit Laenge n
2  Output: sortiertes Array
3
4  InsertionSort(A)
5    j = 0;
6    while (j < n)
7      key = A[j];
8      i = j-1;
9      while (i>=0 && A[i]>key)
10       A[i+1] = A[i];
11       i--;
12     A[i+1] = key;
13     j++;
```

t_j bezeichnet Anzahl der Abfragen der while-Bedingung in Zeile 8 für Durchlauf j

Laufzeit InsertionSort

$$\mathcal{O}(1) + (n+1)\mathcal{O}(1) + (n-1)\mathcal{O}(1) + \mathcal{O}(1)\sum_{j=0}^{n-1} t_j +$$
$$\mathcal{O}(1)\sum_{j=0}^{n-1}(t_j-1) + \mathcal{O}(1)\sum_{j=0}^{n-1}(t_j-1) + n\mathcal{O}(1) + n\mathcal{O}(1)$$
$$= \mathcal{O}(1)n + \mathcal{O}(1)\sum_{j=0}^{n-1} t_j$$
$$= \mathcal{O}\left(n + \sum_{j=0}^{n-1} t_j\right)$$

Die Laufzeit hängt stark vom Input ab.

Laufzeit InsertionSort

Best-case:
Wenn das Array sortiert wird ist, ist $t_j = 1$.
$\Longrightarrow$ Laufzeit: $\mathcal{O}(n)$.

Worst-case:
Wenn das Array absteigend sortiert ist, ist $t_j = j+1$.
$\Longrightarrow$ Laufzeit: $\mathcal{O}(n^2)$.

Beobachtung:
Wenn ein Element höchstens h Positionen von seiner Zielposition entfernt ist, dann ist $t_j \leq h+1$.
$\Longrightarrow$ Laufzeit: $\mathcal{O}(hn)$.