Insertionsort

Das Sortierverfahren Insertionsort („Sortieren durch Einfügen“) ist ein stabiles[1] Verfahren und arbeitet bei Verwendung eines Arrays „in-place“[2].

Prinzip von Insertionsort

Vorbemerkung: Der Algorithmus wird für eine aufsteigende Sortierung beschrieben!

Insertionsort entnimmt einem Array im Rahmen einer Zählschleife, beginnend beim zweiten Element des Arrays (Index 1), ein Element (beim ersten Durchlauf das Element mit dem Index 1), speichert dieses Element in einer temporären Speicherstelle, z.B. **`tmpElement`** und merkt sich den Index des entnommenen Elements in einer Zählvariablen, z.B. **`tmpIndex`**.

Nun wird in einer offenen **`while`**-Schleife überprüft, ob **`tmpIndex`** größer 0 ist. Sollte dies „wahr“ sein, wird nun in einer Entscheidung überprüft, ob das Element mit dem vorhergehenden Index (also „**`tmpIndex - 1`**“) größer als das gemerkte Element in **`tmpElement`** ist.

Ist dies der Fall, wird das Element mit dem Index **`tmpIndex - 1`** an die Stelle von **`tmpIndex`** kopiert (quasi „verschoben“) und der gemerkte Index **`tmpIndex`** wird dekrementiert.

Liefert die Vergleichs-Entscheidung den Wert „falsch“, endet die offene **`while`**-Schleife. Nach der Schleife wird dann das gemerkte Element **`tmpElement`** an der Stelle von **`tmpIndex`** eingefügt.

Dann beginnt der Vorgang von vorn mit dem nächsten durch die Zählschleife entnommenen Element.

Im folgenden Beispiel wird der Algorithmus anhand der dargestellten Zahlenfolge mit den Indizes 0 bis 5 visualisiert:

0	1	2	3	4	5	tmpElement	tmpIndex
17	3	15	22	83	67	**3**	**1**

Die äußere Zählschleife beginnt bei dem Element mit dem Index=1 („3“) und speichert dieses Element (orangefarbener Hintergrund) in **`tmpElement`** und merkt sich dessen Index=1 in **`tmpIndex`**.

Nun wird das Element an der Stelle **`tmpIndex - 1`** (hellblauer Hintergrund) mit dem Element **`tmpElement`** verglichen. Da die „17“ größer als die „3“ ist, wird die „17“ an die Stelle **`tmpIndex`** kopiert und **`tmpIndex`** um 1 vermindert. Nun ist **`tmpIndex = 0`**. Damit endet die offene while-Schleife, da deren Bedingung ja „**`tmpIndex > 0`**“ lautet. Hinter der **`while`**-Schleife wird nun **`tmpElement`** („3“) an die Stelle von **`tmpIndex`** (momentan 0) kopiert. Die beiden Elemente haben ihre Plätze getauscht. Die neue Folge sieht nun so aus:

0	1	2	3	4	5	tmpElement	tmpIndex
3	17	15	22	83	18	**15**	**2**

Nun werden durch die äußere Zählschleife das nächste Element „15“ und dessen Index („2“) gespeichert. Wieder wird in der **`while`**-Schleife solange **`tmpIndex > 0`** das davor liegende Element (nun „17“) mit **`tmpElement`** („15“) verglichen und -in diesem Fall- an die Stelle mit dem Index 2 kopiert und **`tmpIndex`** vermindert. Die while-Schleife wird ein zweites Mal durchlaufen und

1 Bei Elementen mit dem gleichen Schlüssel bleibt die Sortierreihenfolge erhalten.

2 Die Sortierung erfolgt im zu sortierenden Array; d.h. es wird kein zweites Array benötigt.

es erfolgt erneut ein Vergleich zwischen **tmpElement** und dem Element mit dem Index **tmpIndex - 1** (also „0“); also „15“ mit „3“. Hier liefert die Bedingung „3“ > „15“ den Wert „falsch“, so dass die **while**-Schleife endet. Anschließend wird tmpElement („15“) an der Stelle **tmpIndex** („1“) eingefügt. Die Zählschleife beginnt den dritten Durchlauf mit dem Index „3“.

0	1	2	3	4	5	tmpElement	tmpIndex
3	15	17	22	83	18	**22**	**3**

Gespeichert wird in **tmpElement** nun die „22“ und in **tmpIndex** der Wert „3“. Da gilt „3 > 0“ wird in der **while**-Schleife nun überprüft, ob „22 > 17“ wahr ist. Da dies aber „falsch“ ist, endet die **while**-Schleife unmittelbar. Auch wenn eigentlich nicht notwendig, wird nun gemäß dem Algorithmus **tmpElement** („22“) an die Stelle von **tmpIndex** („3“) kopiert. Der nächste Index-Wert der Zählschleife ist nun „4“.

0	1	2	3	4	5	tmpElement	tmpIndex
3	15	17	22	83	18	**83**	**4**

Gleiche Prozedur wie zuvor, nur mit anderen Werten. Gespeichert wird in **tmpElement** nun die „83“ und in **tmpIndex** der Wert „4“. Da gilt „4 > 0“ wird in der **while**-Schleife nun überprüft, ob „83 > 22“ wahr ist. Da dies aber „falsch“ ist, endet die **while**-Schleife unmittelbar und **tmpElement** („83“) wird an die Stelle von **tmpIndex** („4“) kopiert. Der nächste Index-Wert der Zählschleife ist nun „5“.

0	1	2	3	4	5	tmpElement	tmpIndex
3	15	17	22	**83**	18	**18**	**5**

tmpElement hat nun den Wert „18“. **tmpIndex** hat den Wert „5“. Da die Bedingung für die **while**-Schleife wahr ist („5 > 0“), wird mittels Vergleich überprüft, ob „83 > 18“ gilt. Da dies wahr ist, wird die „83“ an die Stelle von **tmpIndex** („5“) kopiert und **tmpIndex** dekrementiert („4“).

0	1	2	3	4	5	tmpElement	tmpIndex
3	15	17	**22**	83	83	**18**	**4**

Da für **tmpIndex** immer noch gilt „4 > 0“, wird nun erneut der Wert an **tmpIndex - 1** („22“) mit **tmpElement** verglichen („22 > 18“). erneut wird nun das Element mit **tmpIndex - 1** an die Stelle **tmpIndex** quasi verschoben, so dass die „22“ nun an dem Index „4“ steht. **tmpIndex** wird erneut dekrementiert und hat nun den Wert „3“.

0	1	2	3	4	5	tmpElement	tmpIndex
3	15	**17**	22	22	83	**18**	**3**

Erneut wird der Wert an **tmpIndex - 1** („17“) mit **tmpElement** verglichen („17 > 18“). Der Vergleich schlägt fehl! Die **while**-Schleife endet unmittelbar und **tmpElement** („18“) wird an die Stelle von **tmpIndex** („3“) kopiert, quasi „eingefügt“. Da dies nun der letzte Durchlauf der Zählschleife war, ist das Array nun aufsteigend sortiert!

0	1	2	3	4	5	tmpElement	tmpIndex
3	15	17	18	22	83		

Implementation von Insertionsort

Nachfolgend ist der Algorithmus als statische Methode namens **`Sort(data:T[])`** für eine statische Klasse namens **`Insertionsort`** dargestellt, wobei die zu sortierenden Daten in einem generischen Array namens **`data:T[]`** enthalten sind.

```
public static void Sort(T[] data)
{
    sizeOfArray = data.Length;

    T tmpElement;

    for (int i = 1; i < sizeOfArray; i++)
    {
        int tmpIndex = i;
        tmpElement = data[j];

        while (tmpIndex > 0)
        {
             if (data[tmpIndex - 1].CompareTo(tmpElement) > 0)
             {
                 data[tmpIndex] = data[ tmpIndex - 1];
                 tmpIndex -= 1;
              }
              else
                  break;
        }

        data[tmpIndex] = tmpElement;
    }
}
```

Üblicherweise wird die Bedingung der **`if`**-Entscheidung innerhalb der **`while`**-Schleife mit einer logischen Und-Verknüpfung in den Kopf der **`while`**-Schleife geschrieben, so dass die Entscheidung selbst entfallen kann. Dies sieht dann so aus:

```
while ((tmpIndex > 0) && (data[tmpIndex - 1].CompareTo(tmpElement) > 0))
{
     data[tmpIndex] = data[ tmpIndex - 1];
     tmpIndex -= 1;
}
```

Dadurch reduziert sich der Inhalt der **`while`**-Schleife. Nun wird die **`while`**-Schleife nur dann durchlaufen, wenn beide Teil-Bedingungen erfüllt sind.

Für eine absteigende Sortierung muss lediglich der Vergleichsoperator im Kopf der while-Schleife von „**>**“ gegen „**<**“ ausgetauscht werden.

Aufgaben

1. Informieren Sie sich über die Zeitkomplexität von Insertionsort.
2. Erstellen Sie ein Programm, in dem ein Array von Integer-Zahlen mittels einer Methode **`sortiere(array:int[]):void`** sortiert wird und diese vor und nach der Sortierung ausgegeben wird.
3. Ändern Sie die Methode **`sortiere()`** derart, dass ein zweiter Parameter für die Sortierrichtung angegeben werden kann und entsprechend verwendet wird.