Bubblesort

Bubblesort ist ein sehr einfaches Sortierverfahren für Listen/Arrays, das aber in der Praxis aufgrund seiner schlechten Laufzeit kaum Anwendung findet.

Bubblesort ist ein *stabiles* Sortierverfahren und arbeitet „*in-place*“. Es entsteht ein geringer zusätzlicher Speicherbedarf für das Erzeugen eines temporären Elements für den Tauschvorgang.

Prinzip von Bubblesort

> *„In der Bubble-Phase wird die Eingabe-Liste von links nach rechts durchlaufen. Dabei wird in jedem Schritt das aktuelle Element mit dem rechten Nachbarn verglichen. Falls die beiden Elemente das Sortierkriterium verletzen, werden sie getauscht. Am Ende der Phase steht bei auf- bzw. absteigender Sortierung das größte bzw. kleinste Element der Eingabe am Ende der Liste.*
>
> *Die Bubble-Phase wird solange wiederholt, bis die Eingabeliste vollständig sortiert ist. Dabei muss das letzte Element des vorherigen Durchlaufs nicht mehr betrachtet werden, da die restliche zu sortierende Eingabe keine größeren bzw. kleineren Elemente mehr enthält.*
>
> *Je nachdem, ob auf- oder absteigend sortiert wird, steigen die größeren oder kleineren Elemente wie Blasen im Wasser (daher der Name) immer weiter nach oben, das heißt, an das Ende der Liste. Es werden stets zwei Zahlen miteinander in „Bubbles“ vertauscht.“*[1]

Im Folgenden soll ein einfaches Beispiel mit sieben Integerwerten für eine aufsteigende Sortierung durchgespielt werden. Zu vergleichende Werte sind hervorgehoben und werden, falls erforderlich, getauscht. Nicht mehr vom Vergleich erfasste Felder sind grau hinterlegt.

1. Durchlauf

12	**5**	3	39	17
5	**12**	**3**	39	17
5	3	**12**	**39**	17
5	3	12	**39**	**17**
5	3	12	17	***39***

2. Durchlauf

5	**3**	12	17	***39***
3	**5**	**12**	17	***39***
3	5	**12**	**17**	***39***
3	5	12	17	***39***

3. Durchlauf

3	**5**	12	***17***	***39***
3	**5**	**12**	***17***	***39***
3	5	12	***17***	***39***

4. Durchlauf

3	**5**	***12***	***17***	***39***
3	5	***12***	***17***	***39***

Fertig!

1 Entnommen aus **`https://de.wikipedia.org/wiki/Bubblesort#Prinzip`** abgerufen am 7. Januar 2019

Wie man erkennen kann, ist das Verfahren noch nicht optimal. Eigentlich ist die Menge bereits nach dem zweiten Durchlauf sortiert, so dass während des dritten Durchlaufs keine weiteren Tauschvorgänge auftreten.

Daher kann man das Verfahren eigentlich nach dem dritten Durchlauf vorzeitig beenden. Merkt man sich also während eines Durchlaufs, ob ***kein* Tausch** statt gefunden hat, dann ist kein weiterer Durchlauf vonnöten. Eine weitere Optimierung ist z.B. in der Wikipedia genannt.

Implementation von Bubblesort

In dem folgenden Kasten ist eine rudimentäre Beispielimplementation für eine statische generische Klasse **`Bubblesort`** angegeben. Der Kern, der eigentliche Sortieralgorithmus, fehlt jedoch.

```
public enum Direction
{
    Ascend, Descend
};
public struct BubblesortStatistics
{
    public long Steps;
    public long Exchanges;
}
// Der Datentyp T muss das das Interface
// IComparable implementieren
public static class Bubblesort<T> where T : IComparable
{
    private static Direction direction;
    public static Direction Direction
    {
        get { return direction; }
        set { direction = value; }
    }
    private static BubblesortStatistics statistics;
    static Bubblesort()
    {
        direction = Direction.Ascend;
    }
    public static BubblesortStatistics Sort(T[] data, Direction _direction)
    {
        direction = _direction;
        return Sort(data);
    }
    public static BubblesortStatistics Sort(T[] data)
    {
        // Guard Clause
        if (data == null) throw new ArgumentNullException("data");

        foreach (T item in data)
        {
            if (item == null) throw new NullReferenceException();
        }

        statistics.Steps = 0;         // wird bei jedem inneren Schleifendurchlauf inkrementiert
        statistics.Exchanges = 0;     // wird bei jedem Tauschvorgang inkrementiert

        // Hier ist der Sortieralgorithmus zu implementieren!

        return statistics;
    }
}
```

Aufgaben

1. Informieren Sie sich über die Zeitkomplexität von Bubblesort.
2. Implementieren Sie in einem ***ersten*** Schritt den Bubblesort-Algorithmus für eine aufsteigende Sortierung. Verwenden Sie hierbei **nicht** die Eigenschaft **`direction`**.
3. Ergänzen Sie nun die Möglichkeit einer absteigenden Sortierung. Sie müssen dies durch eine Entscheidung anhand von **`direction`** beim Vergleich entsprechend berücksichtigen.
4. Optimieren Sie die Sortierung! Keine weiteren Tauschvorgänge → vorzeitiges Ende!