Shellsort

Shellsort ist ein instabiles Sortierverfahren, das *in-place* arbeitet. Dieses Sortierverfahren aus dem Jahr 1959 ist nach seinem Entwickler Donald L. Shell benannt[1]. Das Verfahren ist eine Weiterentwicklung des Verfahrens „*Insertion Sort*“. Der Vorteil des Verfahrens ist, das Elemente bei der Sortierung der „Unterarrays“ indirekt über große Distanzen umsortiert werden.

Prinzip von Shellsort

Das zu sortierende Array wird quasi in „Unterarrays“ aufgeteilt und jedes Unterarray wird dann mit *Insertion Sort* sortiert. Das Erzeugen eines solchen „Unterarrays“ erfolgt mittels eines Zahlenwertes, der den Abstand der zu sortierenden Elemente in der Ursprungsfolge vorgibt. Dieser Abstand wird nach jeder Sortierung aller Unterarrays verkleinert, so dass die Unterarrays immer größer werden.Zum Schluß beträgt der Abstand 1.

Die Zeitkomplexität dieses Verfahrens hängt u.a. stark von der Ausgangssituation und der verwendeten Zahlenfolge für die Abstände ab.

Im Folgenden soll eine unsortierte Menge von zehn Zahlen sortiert werden. Für die Bildung der Abstände soll die Original-Zahlenfolge von Shell[2] verwendet werden.

Diese wird wie folgt gebildet: $\lfloor \frac{N}{2^k} \rfloor;\ \ wobei\ \ k \geq 1$ [3]

Die Zahlenfolge berechnet sich damit zu: 5, 2, 1. Der erste Abstand wäre somit 5.

Mit Hilfe von fünf Farben werden nun die Unterarrays gekennzeichnet.

62	83	18	53	7	17	96	86	47	69

Gleichfarbige Elemente werden nun als „Unterarray“ aufgefasst, die durch „Insertion Sort“ sortiert werden. Begonnen wird mit dem gelben „Unterarray“ (62, 17). Danach wird das lilafabene „Unterarray“ (83, 96) mit *Insertion Sort* sortiert usw. Nach dem Durchlauf mit dem Abstand 5 befindet sich das Array im folgenden Zustand:

17	83	18	47	7	62	96	86	53	69

Es wird nun der Abstand 2 verwendet (zwei Farben).

17	83	18	47	7	62	96	86	53	69

Jetzt werden zuerst alle gelben Werte mit *Insertion Sort* sortiert:

7	83	17	47	18	62	53	86	96	69

Nun alle blauen Werte:

7	47	17	62	18	69	53	83	96	86

Jetzt wird der Abstand 1 verwendet. Aus dieser Folge...

7	47	17	62	18	69	53	83	96	86

...entsteht dann mittels *Insertion Sort* die fertig sortierte Folge!

7	17	18	47	53	62	69	83	86	96

1 Siehe **`https://en.wikipedia.org/wiki/Shellsort`** abgerufen am 8. Januar 2019

2 a.a.O.

3 Die verwendeten Klammern mit der fehlenden oberen Ecke nennt man (auch) Gaußklammern. Diese beinhalten eine Abrundungsfunktion insofern, dass einer reellen Zahl die nächst kleinere oder gleiche natürliche Zahl zugeordnet wird.
Siehe hierzu auch **`https://de.wikipedia.org/wiki/Abrundungsfunktion_und_Aufrundungsfunktion`** abgerufen am 08.01.2019

Eine alternative Darstellung dieses Prinzips findet sich in der deutschen Wikipedia[4].

Zu sortieren sei wieder die Zahlenfolge **`62, 83, 18, 53, 07, 17, 96, 86, 47, 69`**.

Nun werden Zeilen mit einer Länge von 5 gebildet und untereinander geschrieben. Diese fünf Spalten bilden dann die jeweiligen fünf „Unterarrays“.

```
62, 83, 18, 53, 07
17, 96, 86, 47, 69
```

Jetzt werden alle Unterarrays mit *Insertion Sort* sortiert. Es entsteht folgender Zustand:

`17, 83, 18, 47, 07` `62, 96, 86, 53, 69`	`17, 83, 18, 47, 07, 62, 96, 86, 53, 69`

Jetzt werden Zeilen mit der Länge 2 gebildet.
Die beiden Spalten werden dann wieder mit *Insertion Sort* sortiert und es entsteht:

`17, 83`
`18, 47`
`07, 62`
`96, 86`
`53, 69`
	`07, 47`
	`17, 62`
	`18, 69`
	`53, 83`
	`96, 86`
		`07, 47, 17, 62, 18, 69, 53, 83, 96, 86`

Der letzte Abstandswert ist 1. Somit wird jetzt eine Zeile mit der Länge 1 gebildet. Im Prinzip schreibt man nun alle Elemente untereinander. Diese „lange“ Spalte wird dann ein letztes Mal mit *Insertion Sort* sortiert und es entsteht das fertig sortierte Array.

Implementation von Insertion Sort

Auf der nächsten Seite ist eine Beispielimplementation in Form einer statischen generischen Klasse namens **`Shellsort`** angegeben. Die Sortierreihenfolge ist „aufsteigend“.

Das Array **`cols:int[]`** gibt die Folge der Abstände vor, die verwendet werden sollen. Sollte der mittels **`foreach`** entnommene Abstandswert größer oder gleich der Länge des Arrays sein, wird zum nächsten Abstandswert übergegangen.

Die for-Schleife beginnt nun

4 Siehe **`https://de.wikipedia.org/wiki/Shellsort#Beispiel`** abgerufen am 8. Januar 2019

```
public enum Direction
{
    Ascend, Descend
};

// basierend auf
// www.iti.fh-flensburg.de/lang/algorithmen/sortieren/shell/shell.htm
public static class Shellsort<T> where T : IComparable
{
    private static int sizeOfArray;

    private static Direction direction;
    public static Direction Direction
    {
        get { return direction; }
        set { direction = value; }
    }

    static Shellsort()
    {
        direction = Direction.Ascend;
    }

    public static void Sort(T[] data, Direction _direction)
    {
        direction = _direction;
        Sort(data);
    }

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

        int i, j, columns;
        // temporäres Datenelement
        T t;

        // Spaltenanzahl
        int[] cols = {1391376, 463792, 198768, 86961, 33936,
                      13776, 4592, 1968, 861, 336, 112, 48,
                      21, 7, 3, 1};

        foreach (int colvalue in cols)
        {
            columns = colvalue;
            // Nur Spaltenanzahlen kleiner der Arraygröße sind von Belang
            for (i = columns; i < sizeOfArray; i++)
            {
                j = i;
                t = data[j];

                while ((j >= columns) && (data[j - columns].CompareTo(t) > 0))
                {
                    data[j] = data[j - columns];
                    j = j - columns;
                }

                data[j] = t;
            }
        }
    }
}
```

Insertion Sort