Der Algorithmusbegriff

Lernen: Merkmale eines Algorithmus wissen, erklären und anwenden können

Ein Algorithmus ist eine Vorschrift zur Lösung eines Problems. Er hat folgende Eigenschaften:

1. Das Verfahren muss in einem endlichen Text eindeutig beschreibbar sein (**Finitheit**).
2. Jeder Schritt des Verfahrens muss tatsächlich ausführbar sein (**Ausführbarkeit**).
3. Das Verfahren darf nur endlich viele Schritte benötigen (**Terminierung**).
4. Der Algorithmus muss bei denselben Voraussetzungen das gleiche Ergebnis liefern (**Determiniertheit**).
5. Die nächste anzuwendende Regel im Verfahren ist zu jedem Zeitpunkt eindeutig definiert (**Determinismus**).

Algorithmen werden in der Informatik mit Programmiersprachen umgesetzt. Das sind Sprachen, mit denen sich Lösungsstrategien sehr gut und eindeutig formulieren lassen. Diese Sprachen unterscheiden sich voneinander in ihrer Syntax und ihren Möglichkeiten. Wie du noch sehen wirst, muss so eine Sprache noch nicht einmal Text enthalten - einfache Symbole oder gar Farben erfüllen auch diesen Zweck.

Bubblesort in der Programmiersprache Python

Du kannst dir hier die Umsetzung von Bubblesort in Python ansehen - einer momentan sehr weit verbreiteten Programmiersprache.

bubblesort.py

```
def bubbleSort(arr):
    # Anzahl der Zahlen ermitteln
    n = len(arr)
    # Erstmal nehmen wir an, dass keine Vertauschungen notwendig waren
    swapped = False
    # Jetzt nehmen wir uns jede Zahl vor und vergleichen sie mit ihrer
Nachbarzahl
    for i in range(n-1):
        # "for" ist eine Schleife, die (n-1)-mal läuft
        # i ist ein Zähler und erhöht sich bei jedem Durchlauf um 1
        # die letzte Zahl lassen wir beim Vergleich aus
        # wir gehen nur bis zur vorletzten (n-1)
        # das sind praktisch die "Durchläufe"
        for j in range(0, n-i-1):

            # bei jedem Durchlauf müssen wir Schritt für Schritt durch
die
            # noch nicht geprüften Zahlen gehen
            # Durchlauf 1 - Schritt 1, Durchlauf 1 - Schritt 2 usw.
```

```
            if arr[j] > arr[j + 1]:
                swapped = True
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

        if not swapped:
            # wenn wir nichts tauschen mussten, sind wir fertig
            return


# Zahlen, die zu sortieren sind
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print("Ausgabe der sortierten Zahlen:")
for i in range(len(arr)):
    print("% d" % arr[i], end=" ")
```

Aufgabe 1 (zusammen, Partnerarbeit) - Ein Problem nach einer Vorgabe lösen

1. Erklärt, warum die Beschreibung von Bubblesort ein Algorithmus ist!
2. Warum ist es kein Problem, wenn einige Zahlen mehrfach vorkommen?
3. Wie viele Schritte braucht man bei Bubblesort im einfachsten Fall (bereits sortierte Zahlen) immer?
4. **Schwierig:** Was könnte man ändern, damit der Bubblesort-Algorithmus nie terminiert?

Bubblesort nutzt eine sehr grundlegende informatische Lösungsstrategie, die sich divide&conquer nennt („teile und bewältige“). Es werden immer nur zwei Zahlen (divide) aus einer großen Menge von Zahlen miteinander verglichen (conquer). Das Problem, welche Zahl größer ist, lässt sich sehr einfach und mit wenig Aufwand lösen.

Bubblesort einmal abstrakt in einer Syntax

Wir nehmen einmal an, dass die Zahlen **3,7,1 und 8** zu sortieren sind.

Durchlauf 1.1: 3 7 1 8 (nichts tauschen)

Durchlauf 1.2: 3 7⇔1 8 (1 gegen 7 tauschen)

Durchlauf 1.3: 3 1 7 8 (nichts tauschen, am Ende angekommen, von vorne)

Durchlauf 2.1: 3⇔1 7 8 (3 gegen 1 tauschen)

Durchlauf 2.2: 1 3 7 8 (nichts tauschen)

Durchlauf 2.3: 1 3 7 8 (nichts tauschen, am Ende angekommen, von vorne)

Durchlauf 3.1: 1 3 7 8 (nichts tauschen)

Durchlauf 3.2: 1 3 7 8 (nichts tauschen)

Durchlauf 3.3: 1 3 7 8 (nichts tauschen, am Ende angekommen und vorher nichts getauscht - wir sind fertig)

Wir brauchen also neun einzelne Vergleiche, um die vier Zahlen zu sortieren. Vergleiche sind die aufwändigste Operation beim Sortieren. Wir müssen in den dritten Durchlauf, da wir noch keinen Durchlauf ohne Tausch hatten. Eine kürzere Schreibweise (**Syntax**) ohne Kommentare könnte so aussehen:

1.1: 3 7 1 8

1.2: 3 7⇔1 8

1.3: 3 1 7 8

2.1: 3⇔1 7 8

2.2: 1 3 7 8

2.3: 1 3 7 8

3.1: 1 3 7 8

3.2: 1 3 7 8

3.3: 1 3 7 8

Aufgabe 2 (alleine) - eine Syntax umsetzen

Folgende Zahlenfolge ist gegeben: 3, 7, 1, 8, 2, 5, 9, 4, 6

1. Schreibe die Abfolge der Durchläufe in der kurzen Schreibweise für diese Zahlenreihe auf.
2. Vergleiche zwischendurch immer mal wieder mit den Lösungen deiner Nachbarn.

1. Durchlauf
1.1: 3 7 1 8 2 5 9 4 6
1.2: 3 7⇔1 8 2 5 9 4 6
1.3: 3 1 7 8 2 5 9 4 6
1.4: 3 1 7 8⇔2 5 9 4 6
1.5: 3 1 7 2 8⇔5 9 4 6
1.6: 3 1 7 2 5 8 9 4 6

1.7: 3 1 7 2 5 8 9⇔4 6
1.8: 3 1 7 2 5 8 4 9⇔6

2. Durchlauf
2.1: 3⇔1 7 2 5 8 4 6 9
2.2: 1 3 7 2 5 8 4 6 9
2.3: 1 3 7⇔2 5 8 4 6 9
2.4: 1 3 2 7⇔5 8 4 6 9
2.5: 1 3 2 5 7 8 4 6 9
2.6: 1 3 2 5 7 8⇔4 6 9
2.7: 1 3 2 5 7 4 8⇔6 9
2.8: 1 3 2 5 7 4 6 8 9

3. Durchlauf
3.1: 1 3 2 5 7 4 6 8 9
3.2: 1 3⇔2 5 7 4 6 8 9
3.3: 1 2 3 5 7 4 6 8 9
3.4: 1 2 3 5 7 4 6 8 9
3.5: 1 2 3 5 7⇔4 6 8 9
3.6: 1 2 3 5 4 7⇔6 8 9
3.7: 1 2 3 5 4 6 7 8 9
3.8: 1 2 3 5 4 6 7 8 9

4. Durchlauf
4.1: 1 2 3 5 4 6 7 8 9
4.2: 1 2 3 5 4 6 7 8 9
4.3: 1 2 3 5 4 6 7 8 9
4.4: 1 2 3 5⇔4 6 7 8 9
4.5: 1 2 3 4 5 6 7 8 9
4.6: 1 2 3 4 5 6 7 8 9
4.7: 1 2 3 4 5 6 7 8 9
4.8: 1 2 3 4 5 6 7 8 9

5. Durchlauf
5.1: 1 2 3 4 5 6 7 8 9
5.2: 1 2 3 4 5 6 7 8 9
5.3: 1 2 3 4 5 6 7 8 9
5.4: 1 2 3 4 5 6 7 8 9
5.5: 1 2 3 4 5 6 7 8 9
5.6: 1 2 3 4 5 6 7 8 9
5.7: 1 2 3 4 5 6 7 8 9
5.8: 1 2 3 4 5 6 7 8 9

5×8 = 40 Vergleiche notwendig!