Ben Siebert

Gymnasium Holthausen

Projektarbeit
Sortieralgorithmen

2023/2024 Q1 PKNW

15. Februar 2024

Inhaltsverzeichnis

1. Einleitung

Sortieralgorithmen sind ein wichtiger Bestandteil der Informatik. Sie dienen allerdings nicht nur zum Sortieren von Daten, sondern auch als Grundlage für andere Algorithmen, denn Sortieralgorithmen sind in der Regel einfach zu implementieren und enthalten viele wichtige Konzepte der Informatik. Zu diesen essenziellen Konzepten gehören Schleifen, Variablen, lineare Datenstrukturen und Rekursion, welche alle in der Informatik häufige Anwendung finden. So bietet der QuickSort-Algorithmus zum Beispiel einen guten Einsteig in die Rekursion und der BubbleSort-Algorithmus in die Komplexitätsanalyse. Ebenfalls helfen Sortieralgorithmen dabei zu verstehen, wie Aufwandsanalysen und die damit verbunde "Big-O Notation" funktionieren.

In dieser Arbeit werden zunächst die Funktionsweisen der Sortieralgorithmen BubbleSort, SelectionSort, InsertionSort und QuickSort erläutert. Zur besseren Veranschaulichung ist für ausgewählte Algorithmen der entsprechende Java-Quellcode im Anhang angegeben. Anschließend werden die Algorithmen im Bezug auf ihren Aufwand analysiert. Im Zuge dessen werden die Algorithmen mit einander verglichen. Abschließend wird anhand von Beispielen erläutert, wie die Funktionsweise von Sortieralgorithmen Schülern anschaulich im Unterricht vermittelt werden kann.

Im Anhang sind die wichtigsten fachspezifischen Begriffe und Konzepte angegeben, die in dieser Arbeit verwendet werden.

Definition Sortieralgorithmus Ein Sortieralgorithmus ist ein Algorithmus, der eine Menge von Elementen in eine bestimmte Reihenfolge bringt. Die Reihenfolge wird durch eine Relation zwischen den Elementen festgelegt. Die Relation kann zum Beispiel eine Ordnungsrelation (kleiner als,

größer als) sein. Sortieralgorithmen können auf alle linearen Datenstrukturen angewendet werden, die eine sequentielle Zugriffsmöglichkeit auf die Elemente bieten. Die im Anhang aufgeführten Java-Quellcodes sind immer auf Arrays angewendet, da Arrays die einfachsten und bekanntesten linearen Datenstrukturen sind.

2. Algorithmen

2.1 Überblick

Es gibt viele verschiedene Sortieralgorithmen, die alle unterschiedliche Eigenschaften aufweisen. Diese sind beispielsweise die Laufzeit und der Speicherbedarf, aber auch die Stabilität und die Anzahl der Vergleiche und Vertauschungen. Die Laufzeit und der Speicherbedarf sind die wichtigsten Eigenschaften eines Sortieralgorithmus, da diese die Effizienz des Algorithmus bestimmen. [1]

Im nachfolgenden Abschnitt werden die vier Algorithmen "BubbleSort", "InsertionSort", "SelectionSort" und "QuickSort" vorgestellt, welche alle auf linearen Datenstrukturen anwendbar sind.

2.2 BubbleSort

Das "BubbleSort"-Verfahren ist ein einfacher Sortieralgorithmus, der nach dem Prinzip des "Vergleichens und Vertauschens" arbeitet. Dabei werden die zu sortierenden Elemente paarweise verglichen und bei Bedarf vertauscht. Dieser Vorgang wird so lange wiederholt, bis die Elemente in der gewünschten Reihenfolge angeordnet sind.
Das "BubbleSort"-Verfahren ist auf alle linearen Datenstrukturen anwendbar, die eine sequentielle Zugriffsmöglichkeit auf die Elemente bieten. [2]

Beispiel

[1] **sorting-algos**: **sorting-algos** (Stand: **sorting-algos**) **sorting-algos**
[2] **informatik-bg**: **informatik-bg** (Stand: **informatik-bg**) **informatik-bg**

1. Durchlauf	3	1	2
2. Durchlauf	1	3	2
3. Durchlauf	1	2	3

Tabelle 2.1: Beispiel für das "BubbleSort"-Verfahren

2.3 SelectionSort

Das "SelectionSort"-Verfahren ist ebenfalls ein einfacher Sortieralgorithmus, der, wie das "BubbleSort"-Verfahren, nach dem Prinzip des "Vergleichens und Vertauschens" arbeitet. Dieses Verfahren ist auf alle linearen Datenstrukturen anwendbar, die eine sequentielle Zugriffsmöglichkeit auf die Elemente bieten. Der Algorithmus arbeitet wie folgt[3]:

- Das erste Element wird mit allen anderen Elementen verglichen.
- Das kleinste Element wird an die erste Stelle gesetzt.
- Das zweite Element wird mit allen anderen Elementen verglichen.
- Das kleinste Element wird an die zweite Stelle gesetzt.
- Dieser Vorgang wird so lange wiederholt, bis die Elemente in der gewünschten Reihenfolge angeordnet sind.

Beispiel

1. Durchlauf	3	4	1	2
2. Durchlauf	1	3	4	2
3. Durchlauf	1	2	4	3
4. Durchlauf	1	2	3	4

Tabelle 2.2: Beispiel für das "SelectionSort"-Verfahren

[3] **selection-sort**: **selection-sort** (Stand: **selection-sort**) **selection-sort**

2.4 InsertionSort

Das "InsertionSort"-Verfahren funktioniert nach dem Prinzip des "Vergleichens und Einfügens". Wie das "SelectionSort"-Verfahren und das "BubbleSort"-Verfahren ist es auf alle linearen Datenstrukturen anwendbar, die eine sequentielle Zugriffsmöglichkeit auf die Elemente bieten. Um eine solche Datenstruktur zu sortieren, werden folgende Schritte durchgeführt[4]:

- Das erste Element wird als sortiert angesehen.
- Das zweite Element wird mit dem ersten Element verglichen.
- Das zweite Element wird an die richtige Stelle eingefügt.
- Das dritte Element wird mit dem zweiten Element verglichen.
- Das dritte Element wird an die richtige Stelle eingefügt.
- Dieser Vorgang wird so lange wiederholt, bis die Elemente in der gewünschten Reihenfolge angeordnet sind.

Beispiel

1. Durchlauf	3	1	2	4
2. Durchlauf	1	3	2	4
3. Durchlauf	1	2	3	4

Tabelle 2.3: Beispiel für das "InsertionSort"-Verfahren

2.5 QuickSort

Das "QuickSort"-Verfahren ist im Vergleich den anderen Verfahren, ein sehr effizienter, aber auch komplexerer Sortieralgorithmus. Es basiert auf dem Prinzip "Teile und Herrsche". Dabei wird zunächst ein Element aus

[4] **insertion-sort**: **insertion-sort** (Stand: **insertion-sort**) **insertion-sort**

der zu sortierenden Datenmenge ausgewählt, welches als "Pivotelement" bezeichnet wird. Anschließend werden alle Elemente der Datenmenge mit dem Pivotelement verglichen. Die Elemente, die kleiner als das Pivotelement sind, werden in eine Teilliste einsortiert. Die Elemente, die größer als das Pivotelement sind, werden in eine andere Teilliste einsortiert. Die beiden Teillisten werden anschließend rekursiv sortiert. [5]

Beispiel

1. Durchlauf	3	1	2	5	4
2. Durchlauf	1	2	3	5	4
3. Durchlauf	1	2	3	4	5
4. Durchlauf	1	2	3	4	5

Tabelle 2.4: Beispiel für das "QuickSort"-Verfahren

[5] **quick-sort**: **quick-sort** (Stand: **quick-sort**) **quick-sort**

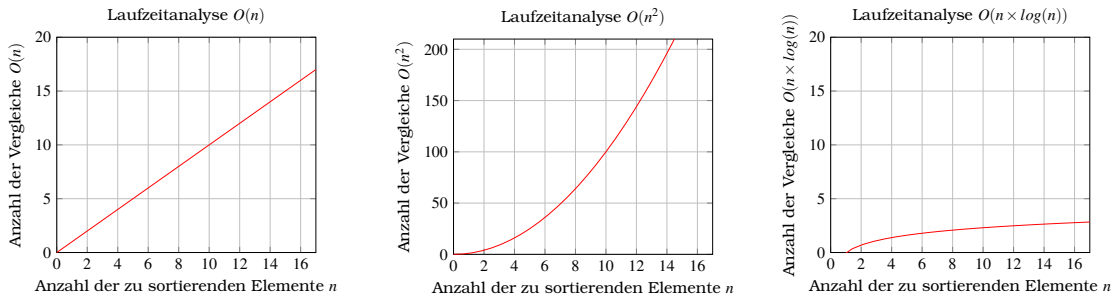
2.6 Aufwandsanalyse

In der Informatik wird die Laufzeit eines Algorithmus in der Regel in der sogenannten "Big-O-Notation" angegeben. Im Beispiel $O(n)$ geht man von einer linearen Laufzeit aus, das heißt, dass die Anzahl der Vergleiche proportional zur Anzahl der Elemente ist. "n" steht hierbei für die Anzahl der Elemente in der zu sortierenden Datenmenge. In der Tabelle 2.5 sind die Laufzeiten der benannten Sortieralgorithmen aufgelistet.[6]

Algorithmus	**bester Fall**	**schlechtester Fall**	**Durchschnitt**
BubbleSort[7]	$O(n)$	$O(n^2)$	$O(n^2)$
SelectionSort[8]	$O(n)$	$O(n^2)$	$O(n^2)$
InsertionSort[9]	$O(n)$	$O(n^2)$	$O(n^2)$
QuickSort[10]	$O(n \times log(n))$	$O(n^2)$	$O(n \times log(n))$

Tabelle 2.5: Laufzeit-/Aufwandsanalyse der Sortieralgorithmen

Visualisierung Die "Big-O-Notation" kann auch grafisch dargestellt werden. Hierbei wird die Anzahl der zu sortierenden Elemente auf der x-Achse und die Anzahl der Vergleiche auf der y-Achse aufgetragen:



Auswertung Man kann erkennen, dass sowohl der BubbleSort-Algorithmus

[6] **big-o-notation**: **big-o-notation** (Stand: **big-o-notation**) **big-o-notation**

[7] **bubble-sort-aufwand**: **bubble-sort-aufwand** (Stand: **bubble-sort-aufwand**) **bubble-sort-aufwand**

[8] **selection-sort-complexity**: **selection-sort-complexity** (Stand: **selection-sort-complexity**) **selection-sort-complexity**

[9] **insertion-sort**: **insertion-sort** (Stand: **insertion-sort**) **insertion-sort**

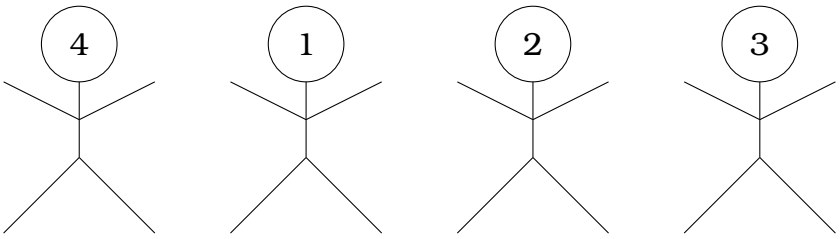
[10] **quick-sort**: **quick-sort** (Stand: **quick-sort**) **quick-sort**

als auch der SelectionSort-Algorithmus und der InsertionSort-Algorithmus im Durchschnitt eine quadratische Laufzeit haben, was bei allen drei Algorithmen dem schlechtesten Fall entspricht. Vergleicht man nun diese Algorithmen mit dem QuickSort-Algorithmus, so fällt auf, dass dieser im Durchschnitt eine deutlich geringere Laufzeit hat und somit effizienter arbeitet. Dies liegt daran, dass der QuickSort-Algorithmus die Datenmenge in jedem Schritt halbiert, wodurch die Anzahl der Vergleiche deutlich geringer ist. Der QuickSort-Algorithmus hat allerdings auch einen schlechtesten Fall, in dem er eine quadratische Laufzeit hat. Dieser Fall tritt ein, wenn die Datenmenge bereits sortiert ist und das Pivot-Element immer das kleinste oder größte Element ist.

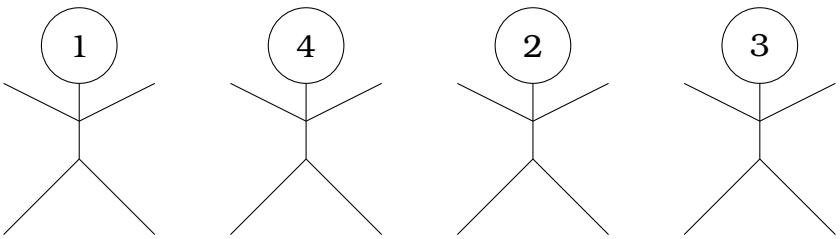
3. Integration in den Unterricht

Im Unterricht können Sortieralgorithmen auf verschiedene Arten den Schülern näher gebracht werden. Die einfachste Möglichkeit ist die visuelle Darstellung der Algorithmen. Hierbei bekommt jeder Schüler ein Blatt Papier, auf dem eine Zahl abgebildet ist. Jede Zahl sollte, wenn möglich, nur einmal verwendet werden, damit eine eindeutige Sortierung erkennbar ist. Die Schüler stellen sich nun in einer Reihe auf, die zu Anfang unsortiert ist. Anschließend wird der gewählte Algorithmus schrittweise durchgeführt. Es ist zu empfehlen, mit einem einfachen Algorithmus, wie dem "Bubble-Sort"-Verfahren zu beginnen, damit das Prinzip verstanden werden kann.

Beispiel Es wurden die Zahlen eins bis vier unter den Schülern aufgeteilt, welche sich in eine unsortierte Reihefolge aufgestellt haben und anschließend das "Bubble-Sort"-Verfahren durchführen. Folgende Abbildung zeigt die Ausgangssituation.

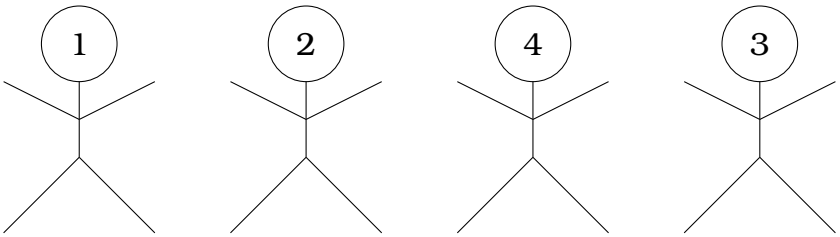


Nun wird der Algorithmus schrittweise durchgeführt. In der folgenden Grafik ist der erste Schritt dargestellt. Die Schüler vergleichen die erste und zweite Zahl. Da die erste Zahl größer ist, werden die beiden Zahlen getauscht.

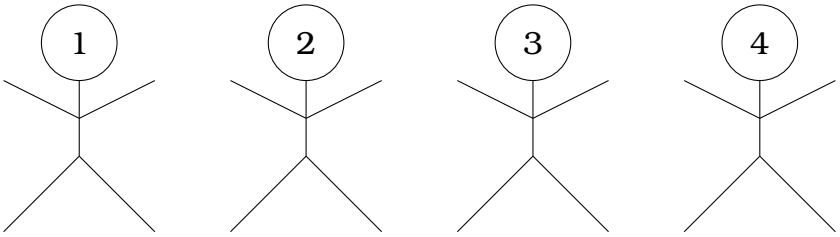


Anschließend werden die zweite und dritte Zahl verglichen. Da die

zweite Zahl größer ist, werden die beiden Zahlen getauscht.



Abschließend werden die dritte und vierte Zahl verglichen. Da die dritte Zahl größer ist, werden die beiden Zahlen getauscht.



Jetzt ist die Reihefolge sortiert und die Prozedur kann mit einer neuen Reihenfolge oder einem anderen Algorithmus wiederholt werden.

Erfahrungen der MINT-Nacht:

Bei der praktischen Umsetzung dieses Vorgehens bei der MINT-Nach am 02.02.24 ist aufgefallen, dass die Größe der Gruppe sowohl für den Lernprozess, als auch für den Spaß der Schüler sehr wichtig ist.

Es ging hervor, dass die optimale Gruppengröße zwischen 6 und 8 Schülern liegt. Durch diese Größe können in kurzer Zeit mehrere Schüler die verschiedenen Sortieralgorithmen durchführen. Hierdurch festigen sich die Funktionsweisen der Algorithmen besser bei den Schülern und es wird eine gewisse Dynamik ins Spiel gebracht. Damit das Spiel besser funktioniert, ist es wichtig, dass die Schüler bei den ersten Durchläufen unterstützt werden. Diese Unterstützung kann bei jedem Durchlauf geringer werden, da besonders beim BubbleSort-Algorithmus das Verfahren für die Schüler leicht verständlich war und schnell selbst durchgeführt werden konnte.

4. Literaturverzeichnis

5. Anhang

5.1 Fachspezifische Begriffe und Konzepte

Lineare Datenstrukturen Lineare Datenstrukturen sind Datenstrukturen, die die Elemente in einer bestimmten Reihenfolge speichern. Die Reihenfolge wird durch die Reihenfolge der Elemente bestimmt. Die Elemente werden nacheinander gespeichert und können nur über die Position im Speicher angesprochen werden. Die bekannteste linearen Datenstrukture ist das Array. Weitere lineare Datenstrukturen sind Listen, Schlangen und Stapel. Einer der größten Unterschiede der verschiedenen linearen Datenstrukturen ist die Zugriffsmöglichkeit auf die Elemente. Bei Arrays kann auf die Elemente über den Index zugegriffen werden. Bei Listen kann auf die Elemente über einen Zeiger zugegriffen werden. Bei Schlangen kann nur auf das erste Element und bei Stapeln nur auf das oberste Element zugegriffen werden. [1]

Big-O Notation Die Big-O-Notation ist eine mathematische Notation, die verwendet wird, um das asymptotische Verhalten von Funktionen zu beschreiben, das heißt das Grenzwertverhalten der Funktion zu klassifizieren[2]. Die Big-O-Notation wird verwendet, um die Laufzeit eines Algorithmus in Abhängigkeit von der Anzahl der zu sortierenden Elemente zu beschreiben. Dabei wird die Anzahl der zu sortierenden Elemente mit n bezeichnet. Die Laufzeit wird in Abhängigkeit von n angegeben. Die Laufzeit wird mit $O(n)$ angegeben. Dabei ist $O(n)$ die obere Schranke der Laufzeit. [3]

[1] **linear-datastructures**: **linear-datastructures** (Stand: **linear-datastructures**) **linear-datastructures**

[2] **asymptomic-analysis**: **asymptomic-analysis** (Stand: **asymptomic-analysis**) **asymptomic-analysis**

[3] **big-o-notation**: **big-o-notation** (Stand: **big-o-notation**) **big-o-notation**

Rekursion Rekursion ist ein Prinzip, das in der Informatik verwendet wird, um ein Problem in kleinere Teilprobleme zu zerlegen. Dabei wird das Problem so lange in kleinere Teilprobleme zerlegt, bis diese einfach zu lösen sind. Anschließend werden die Teilprobleme gelöst und die Lösungen werden zu einer Lösung des ursprünglichen Problems zusammengefügt. [4]

5.2 Java-Quellcode der Sortieralgorithmen

BubbleSort

```
public static int[] bubbleSort(int[] unsorted) {
    int[] sorted = unsorted.clone();
    int temp = 0;
    for (int i = 0; i < sorted.length; i++) {
        for (int j = 1; j < (sorted.length - i); j++) {
            if (sorted[j - 1] > sorted[j]) {
                temp = sorted[j - 1];
                sorted[j - 1] = sorted[j];
                sorted[j] = temp;
            }
        }
    }
    return sorted;
}
```

SelectionSort

```
public static int[] selectionSort(int[] unsorted) {
    int[] sorted = unsorted.clone();
    int temp = 0;
```

[4] **recursion**: **recursion** (Stand: **recursion**) **recursion**

```
        for (int i = 0; i < sorted.length - 1; i++) {
            int min = i;
            for (int j = i + 1; j < sorted.length; j++) {
                if (sorted[j] < sorted[min]) {
                    min = j;
                }
            }
            temp = sorted[min];
            sorted[min] = sorted[i];
            sorted[i] = temp;
        }
        return sorted;
    }
```

InsertionSort

```
    public static int[] insertionSort(int[] unsorted) {
        int[] sorted = unsorted.clone();
        int temp = 0;
        for (int i = 1; i < sorted.length; i++) {
            for (int j = i; j > 0; j--) {
                if (sorted[j] < sorted[j - 1]) {
                    temp = sorted[j];
                    sorted[j] = sorted[j - 1];
                    sorted[j - 1] = temp;
                }
            }
        }
        return sorted;
    }
```

6. Erklärung

Ich erkläre, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Hattingen, 15. Februar 2024 Ben Siebert

Ort, Datum **Unterschrift**