

PD Dr. Mathias J. Krause
M.Sc. Stefan Karch
M.Sc. Mariia Sukhova

09.12.2022

Einstieg in die Informatik und Algorithmische Mathematik

Aufgabenblatt 9

Bearbeitungszeitraum: 9.01.2023 – 20.01.2023

Aufgabe 1 *Sortieren durch Zerlegen (Quicksort)*

Das nachfolgend beschriebene Sortierverfahren ist für kleinere und mittlere Datenbestände im Durchschnitt eines der schnellsten. Wir betrachten im Folgenden Mengen von ganzen Zahlen, welche sortiert werden sollen, bemerken aber, dass dieses Verfahren für jede Art von Grundmenge verwendbar ist, sofern für diese Grundmenge eine Ordnungsrelation definiert wurde. Wir betrachten also eine n -elementige Menge ganzer Zahlen $\{a_1, \dots, a_n\}$. Das Grundprinzip des Verfahrens ist es, aus der gegebenen Menge ein geeignetes Element m (den sog. *Pivot*) auszuwählen und anschließend mittels m die vorgegebene Menge in zwei Teilmengen M_1 und M_2 zu zerlegen, so dass

$$x \leq m \text{ für alle } x \in M_1 \text{ und } m \leq y \text{ für alle } y \in M_2 \text{ gilt.}$$

Nach dieser „Vorsortierung“ läßt sich das ursprüngliche Problem darauf reduzieren, dass die Mengen M_1 und M_2 getrennt voneinander zu sortieren sind und das Zusammenfügen der resultierenden sortierten Tupel die ursprüngliche Menge, jedoch vollständig sortiert, ergibt. Da die beiden Teilmengen jedoch wieder auf dieselbe Art und Weise sortiert werden können, ergibt sich ein rekursives Sortierverfahren.

Die Umsetzung als Algorithmus basiert auf der Idee, dass alle Elemente, die kleiner als der Pivot sind, auch vor dem Pivot im Feld zu finden sind. Wohingegen alle größeren Elemente auch größere Indizes besitzen. Dafür wird das unsortierte Feld jeweils gleichzeitig von links und rechts durchkämmt, um Elemente die dieses Kriterium nicht erfüllen, zu finden und anschließend zu vertauschen. Ist also eine unsortierte Folge $a_{links}, \dots, a_{rechts}$ gegeben, so beschreibt folgendes Verfahren den Quicksort-Algorithmus mit deterministischem Pivot:

Falls $links < rechts$

{

 Setze $m := a_{links}$

 Setze $i := links$ und $j := rechts$

 Führe die folgenden Schritte durch, solange $i \leq j$ gilt

- Solange $a_i < m$ ist, erhöhe i um 1
- Solange $a_j > m$ ist, erniedrige j um 1
- Falls $i \leq j$ gilt, vertausche a_i und a_j , erhöhe i und erniedrige j um 1

Sortiere a_{links}, \dots, a_j

Sortiere a_i, \dots, a_{rechts}

}

Schreiben Sie ein Java-Programm welches die hier vorgestellte Form des quicksort-Algorithmus implementiert. Gehen Sie dazu folgendermaßen vor:

- Erstellen Sie eine öffentliche Klasse `Quicksort` welche zunächst eine Methode `sortieren` ohne Rückgabewert besitzen soll. Ihr sollen als Argumente das zu sortierende Feld `a` vom Typ `int` und zwei Indizes `links` und `rechts` übergeben werden. Diese Methode soll nach obigem Schema die Feldelemente `a[links], ..., a[rechts]` durch Vertauschungen der Größe nach sortieren. Dabei wird das Sortieren der beiden neuen Feldabschnitte wieder durch einen rekursiven Aufruf von `sortieren` erreicht - sie müssen dafür keine neuen Felder erzeugen, sondern lediglich die Indizes beim Aufruf anpassen.

Hinweis: In Java können die Elemente zusammengesetzter Datentypen (wie Felder oder Klassen) innerhalb einer Methode dauerhaft verändert werden, wenn sie als Argument übergeben wurden - bei einfachen Datentypen wie `int` oder `double` ist dies nicht der Fall!

- Erstellen Sie die `main`-Methode der Klasse. Hier soll (mit passendem Text) zunächst die Anzahl der zu sortierenden Elemente `n` abgefragt und danach ein Feld `a` vom Typ `int` mit entsprechender Länge vom Benutzer nach vorheriger Aufforderung über die Konsole gefüllt werden. Dann soll das Feld mittels `sortieren` sortiert und zum Schluss das sortierte Feld wieder auf dem Bildschirm ausgegeben werden.

Hinweis:

- Führen Sie den Algorithmus auch einmal von Hand durch. Nehmen sie zum Beispiel die Zahlen von 1 bis 10 und schreiben Sie sie in unsortierter Reihenfolge auf. Sortieren Sie die Zahlen dann gemäß dem Quicksort-Algorithmus. Auf diese Art und Weise lernen Sie, das Sortierprinzip besser zu verstehen.
- Wir haben hier der Einfachheit halber immer das erste Element der Liste als Pivot gewählt. Welche Folgen hat das beispielsweise für die Laufzeit, wenn die übergebene Liste bereits in umgekehrter Reihenfolge sortiert ist?
Um solche *worst-case*-Laufzeiten zu vermeiden, wird für den Pivot oft ein zufälliges Element der Liste ausgewählt. Damit ist der Algorithmus zumindest im Durchschnitt effizient, ganz egal wie die Eingabe permutiert ist.

Literatur: T. Ottmann, P. Widmayer: *Algorithmen und Datenstrukturen*. BI-Verlag, 1993.

Aufgabe 2 (Pflichtaufgabe) *Ein Gartenzwerg sortiert Blumentöpfe*

Ein Gartenzwerg steht vor einer Reihe von N Blumentöpfen, die sich in ihrer Größe teilweise unterscheiden. Da der Gartenzwerg ein ordnungsliebender Zeitgenosse ist, verspürt er ein inneres Bedürfnis, die Blumentöpfe der Größe nach von links nach rechts aufsteigend anzuordnen. Er tut dies in folgender Weise zu tun:

Der Gartenzwerg stellt sich zu Anfang vor den am weitesten links stehenden Blumentopf. Als nächstes vergleicht er den Blumentopf, vor dem er steht, mit dem Blumentopf, der rechts neben ihm steht. Stellt er fest, dass diese beiden Blumentöpfe richtig angeordnet sind, so bewegt er sich einen Blumentopf weiter nach rechts und wiederholt den Vorgang. Stellt er hingegen fest, dass die zwei Blumentöpfe falsch angeordnet sind, so vertauscht er beide und bewegt sich – sofern dies möglich ist – einen Blumentopf weiter nach links. Dort wiederholt er den Vorgang. Auf diese Weise läuft der Zwerg von Blumentopf zu Blumentopf. Dies tut er so lange, bis er an dem Blumentopf ankommt, der am weitesten rechts steht.

Ohne es zu ahnen, hat der Gartenzwerg in unserer kleinen Geschichte einen Sortieralgorithmus mit dem Namen *GnomeSort* (Zwergensortierung) angewendet. Ihre Aufgabe ist es, diesen Algorithmus in einem Java-Programm für eine Liste von ganzen Zahlen umzusetzen. Gehen Sie dabei wie folgt vor:

- (a) Erstellen Sie eine `main`-Methode in einer öffentlichen Klasse namens `Sortieren`. Lesen Sie in der `main`-Methode zunächst die Anzahl der zu sortierenden ganzen Zahlen von der Konsole ein, und speichern Sie diese Anzahl in einer Variable vom Typ `int` ab.
- (b) Erzeugen Sie ein Feld vom Typ `int`, dessen Länge der Anzahl zu sortierender Zahlen entspricht. Lesen Sie die zu sortierenden Zahlen nacheinander von der Konsole ein und speichern Sie diese im Feld ab. Verwenden Sie eine dazu eine `for`-Schleife
- (c) Sortieren Sie das Feld mit dem *GnomeSort*-Algorithmus. Speichern Sie die Position des Gartenzwergs in einer Variable vom Typ `int` ab. Lassen Sie den Gartenzwerg bei der Position 0, d.h. bei der ersten Feldkomponente beginnen. Verändern Sie anschließend innerhalb einer `while`-Schleife fortlaufend die Position des Gartenzwergs, und vertauschen Sie ggf. die Werte zweier benachbarter Feldkomponenten wie oben beschrieben. Die `while`-Schleife soll so lange durchlaufen werden, bis der Gartenzwerg bei der letzten Feldkomponente angekommen ist.

Hinweis:

- Zum Vertauschen zweier Feldkomponenten benötigen Sie eine zusätzliche Variable vom Typ `int` als Zwischenspeicher.
 - Den Index der letzten Feldkomponente eines Feldes `feld` erhält man durch den Ausdruck `feld.length-1`.
- (d) Geben Sie abschließend die Feldkomponenten des sortierten Feldes nacheinander auf der Konsole aus. Verwenden Sie dazu eine `for`-Schleife. Testen Sie anschließend Ihr Programm.

Fragen 2 Ein Gartenzwerg sortiert Blumentöpfe

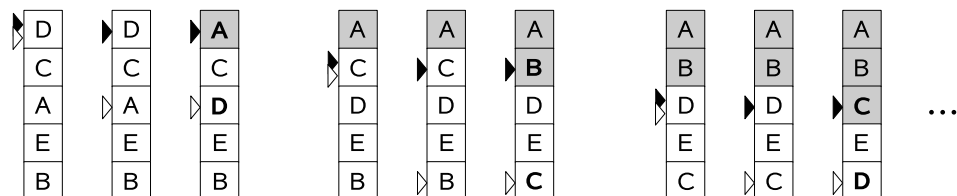
- Nennen Sie jeweils zwei iterative und rekursive Sortieralgorithmen.
- Erklären Sie wann man Sortieralgorithmen als stabil bzw. als in-place bezeichnet.

Aufgabe 3 Sortieren

In der elektronischen Datenverarbeitung taucht häufig das Problem auf, eine Folge von Daten bezüglich einer bestimmten Ordnungsrelation zu sortieren. Hierbei kommen sogenannte *Sortieralgorithmen* zum Einsatz. Ein relativ einfacher Sortieralgorithmus ist der *MinSort*-Algorithmus. Er funktioniert folgendermaßen:

- (1) Gegeben sei ein Feld der Länge n . Der Index der ersten Feldkomponente sei 0.
- (2) Setze $i = 0$.
- (3) Solange $i < n - 1$ gilt, bestimme den Index k der kleinsten Feldkomponente mit $i \leq k \leq n - 1$, vertausche die Feldkomponenten mit den Indizes i und k und erhöhe i anschließend um Eins.

In der folgenden Abbildung sind die ersten drei Schritte des *MinSort*-Algorithmus schematisch dargestellt. Das schwarze Dreieck bezeichnet dabei die Feldkomponente mit dem Index i , das weiße Dreieck die mit dem Index k . Die grau unterlegten Feldkomponenten stellen den *sortierten* Teil des Feldes dar.



Erstellen Sie ein Java-Programm mit dem Namen `MinSort`, welches eine Liste ganzer Zahlen einliest, und diese mit dem *MinSort*-Algorithmus sortiert. Gehen Sie dabei wie folgt vor:

- Erstellen Sie eine `main`-Methode, in der Sie eine ganze Zahl n von der Konsole einlesen und diese in einer Variable vom Typ `int` speichern. Erzeugen Sie anschließend ein Feld namens `liste` der Länge n vom Typ `int`. Lesen Sie mit einer `for`-Schleife n ganze Zahlen von der Konsole ein, und speichern Sie diese in den Feldkomponenten des Feldes `liste`. Geben Sie anschließend die Werte der Feldkomponenten der Reihe nach auf der Konsole aus.
- Sortieren Sie das Feld `liste` mit dem *MinSort*-Algorithmus. Durchlaufen Sie dazu das Feld mit einer `for`-Schleife. Bestimmen Sie für jeden Index i mit einer inneren `for`-Schleife den Index $k \geq i$ der kleinsten Feldkomponente im unsortierten Teil des Feldes. Vertauschen Sie anschließend den Wert der i -ten Feldkomponente mit dem der k -ten.

- (c) Geben Sie die Feldkomponenten des sortierten Feldes auf der Konsole aus. Testen Sie ihr Programm.