Institut für Angewandte und
Numerische Mathematik

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

12.12.2023

Einstieg in die Informatik und Algorithmische Mathematik

Aufgabenblatt 8

Bearbeitungszeitraum: 08.01.2023 – 19.01.2023

Aufgabe 1 (Pflichtaufgabe) *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 $\{a_1, \ldots, a_n\}$. Das Grundprinzip des Verfahren 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 $a_{links}, ..., a_{rechts}$ 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 $a_{links}, \ldots, a_j$

Sortiere $a_i, \ldots, a_{rechts}$

}

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.

Fragen 1 *Sortieren durch Zerlegen (Quicksort)*

- Nennen Sie jeweils zwei iterative und rekursive Sortieralgorithmen.
- Worin besteht der Unterschied zwischen direkter und indirekter Rekursion?

Aufgabe 2 *Reichweitenbestimmung*

Ziel dieser Aufgabe ist es, herauszufinden, ob ein Fahrzeug mit einer begrenzten Menge an Treibstoff eine Wüste durchqueren kann. Das Fahrzeug habe eine Tankkapazität von c Litern Treibstoff und verbrauche v Liter Treibstoff pro 100 km. Am Ausgangspunkt stehen k Tankfüllungen Treibstoff zur Verfügung.
Das Fahrzeug kann nur in seinem Tank Treibstoff mit sich führen. Auf dem Weg durch die Wüste müssen daher Treibstoffzwischenlager angelegt werden. Dies wird in folgender Weise realisiert: Das Fahrzeug verlässt den Ausgangspunkt mit vollem Tank und lädt am ersten Zwischenlager soviel Treibstoff ab, dass der übrigbleibende Treibstoff gerade noch reicht, um zum Ausgangspunkt zurückzufahren. Dies wird so oft wiederholt, bis sämtlicher Treibstoff zum ersten Zwischenlager transportiert wurde. Das Fahrzeug soll dabei insgesamt eine Tankfüllung verbrauchen, so dass am ersten Zwischenlager $k-1$ Tankfüllungen zur Verfügung stehen.
Die Distanz zwischen Ausgangspunkt und erstem Zwischenlager $d(k,c,v)$, sowie die Treibstoffmenge $a(k,c,v)$, die jedes mal am ersten Zwischenlager ausgeladen werden muss, kann man folgendermaßen berechnen

$$d(k,c,v) = \frac{100c}{(2k-1)v}, \qquad a(k,c,v) = c - \frac{2vd(k,c,v)}{100}.$$

Das zweite Zwischenlager wird nach derselben Arbeitsweise errichtet. Die Distanz zwischen erstem und zweitem Zwischenlager ist entsprechend durch $d(k-1,c,v)$, die im zweiten Zwischenlager zu entladende Treibstoffmenge durch $a(k-1,c,v)$ gegeben. Die maximale Reichweite $R(k,c,v)$ des Fahrzeugs kann man folgendermaßen rekursiv bestimmen:

$$R(k,c,v) = \begin{cases} \dfrac{100c}{v} & \text{falls } k = 1, \\ d(k,c,v) + R(k-1,c,v) & \text{falls } k > 1. \end{cases}$$

Schreiben Sie ein Java-Programm, welches die Zwischenlager plant und die maximale Reichweite des Fahrzeugs bestimmt. Gehen Sie dabei folgendermaßen vor:

- Erstellen Sie eine öffentliche Klasse mit dem Namen `Reichweite`. Definieren Sie für die Klasse eine Klassenmethode namens `distanz` mit drei formalen Parametern für die ganzzahlige Anzahl der vorhandenen Tankfüllungen k, der Tankkapazität des Fahrzeugs c und dem Verbrauch pro 100 km v je in Gleitkommazahlen. Die Methode soll die Distanz zum nächsten Zwischenlager $d(k,c,v)$ als Gleitkommazahl zurückgeben.
- Definieren Sie eine Klassenmethode namens `ausladen` mit einem Rückgabewert vom Typ **`double`** und drei formalen Parametern vom Typ **`int`**, **`double`** und **`double`**. Die Methode soll die Menge des Treibstoffs $a(k,c,v)$, der am nächsten Zwischenlager ausgeladen werden muss, als Wert vom Typ **`double`** zurückgeben.

- Definieren Sie eine Klassenmethode namens `reichweite` mit einer Gleitkommazahl als Rückgabewert und drei formalen Parametern vom Typ **`int`**, **`double`** und **`double`**. In der Methode soll die maximale Reichweite $R(k, c, v)$ für k Tankfüllungen *rekursiv* berechnet und als Gleitkommazahl zurückgegeben werden.
- Schreiben Sie nun die `main`–Methode der Klasse. Lesen Sie zunächst die Anzahl der vorhandenen Tankfüllungen k, die Tankkapazität des Fahrzeugs c und dessen Verbrauch pro 100 km v von der Konsole ein. Berechnen Sie dann die Distanzen zwischen den k Zwischenlagern, sowie die jeweils zu entladenden Treibstoffmengen. Geben Sie jeweils beide Größen auf der Konsole aus. Bestimmen Sie zuletzt die maximal Reichweite des Fahrzeugs und geben Sie diese ebenfalls auf der Konsole aus.
- Testen Sie Ihr Programm mit folgenden Daten: Für $k = 3$, $c = 20$ und $v = 10$ erhält man die Streckenplanung

Teilstrecke	Distanz	Ausladen
1	$40{,}00\,\mathrm{km}$	$12{,}00\,\ell$
2	$66{,}67\,\mathrm{km}$	$6{,}67\,\ell$
3	$200{,}00\,\mathrm{km}$	–

  Die maximale Reichweite beträgt für diesen Fall also $306{,}67\,\mathrm{km}$.

Aufgabe 3 *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 Art und Weise:

> 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 Blumentopfe 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:

- 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 ganzzahligen Variable ab.

- Erzeugen Sie ein Feld, welches ganzzahlige Zahlen speichert und 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
- Sortieren Sie das Feld mit dem *GnomeSort*–Algorithmus. Speichern Sie die Position des Gartenzwergs in einer ganzzahligen Variable 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 Hilfsvariable als Zwischenspeicher.
  - Den Index der letzten Feldkomponente eines Feldes `feld` erhält man durch den Ausdruck `feld.length-1`.
- Geben Sie abschließend die Feldkomponenten des sortierten Feldes nacheinander auf der Konsole aus. Verwenden Sie dazu eine **`for`**–Schleife. Testen Sie anschließend Ihr Programm.