

DAP2 Praktikum – Blatt 2

Abgabe: ab 18. April

Studienleistung

- Zum Bestehen des Praktikums muss jeder Teilnehmer die folgenden Leistungen erbringen:
 - Es müssen mindestens 50 Prozent der Punkte in den Kurzaufgaben erreicht werden.
 - Es müssen mindestens 50 Prozent der Punkte in den Langaufgaben erreicht werden.
- Im Krankheitsfall kann ein Testat bei Vorlage eines Attests in der folgenden Woche nachgeholt werden.
- Wenn ein Praktikumstermin auf einen Feiertag fällt, müssen Sie sich an einem beliebigen anderen Praktikumstermin in der gleichen Woche testieren lassen.
- **Hinweis:** Notieren Sie sich Ihre Punkte nach jedem Testat! Dies dient der eigenen Kontrolle. (Ihr Punktestand kann Ihnen während des Semesters nicht genannt werden kann.)

Wichtige Information (im Moodle verfügbar)

- Beachten Sie die Erklärung des **Ablaufs (Blatt A)**.
- Beachten Sie die **Regeln und Hinweise (Blatt R)** in der aktuellsten Version!
- Beachten Sie die **Hilfestellungen (Blatt H)** in der aktuellsten Version!

Languaufgabe 2.1: Quicksort

(3 Punkte)

Auf diesem Blatt dreht sich alles um den vergleichsbasierten Sortieralgorithmus Quicksort. Der Algorithmus soll ein Subarray $A[l, r]$ von vergleichbaren Werten (möglicherweise das gesamte Array $A = A[0, n - 1]$) als Eingabe bekommen, und es in *absteigende Reihenfolge* bringen. Beispiel für ein (Sub-)Array der Länge 8:

[5, 8, 1, 4, 4, 9, 2, 3] \rightarrow [9, 8, 5, 4, 4, 3, 2, 1]

Dabei geht der Algorithmus rekursiv vor. Wenn $l \geq r$, dann enthält das Subarray höchstens ein Element und ist bereits korrekt sortiert. Andernfalls wird $A[l, r]$ zunächst partitioniert. Sei $p = A[l]$ das erste Element des Subarrays *vor der Partitionierung* (das sogenannte Pivot-Element). Die Elemente des Subarrays werden nun umgeordnet, sodass für ein $m \in \{l, \dots, r\}$ gilt:

- $A[m] = p$ (i.e., m ist die neue Position von p), und
- $\forall i \in \{l, \dots, m\} : A[i] \geq p$ (i.e., Elemente links von p sind nicht kleiner als p), und
- $\forall i \in \{m, \dots, r\} : A[i] \leq p$ (i.e., Elemente rechts von p sind nicht größer als p).

Jetzt müssen nur noch die Intervalle $A[l, m - 1]$ und $A[m + 1, r]$ rekursiv sortiert werden. Der erste Partitionierungsschritt für das obige Beispiel sieht wie folgt aus:

[5, 8, 1, 4, 4, 9, 2, 3] \rightarrow $\underbrace{[8, 9]}_{\text{rekursiv sortieren}}, \overset{\text{Pivot-Element}}{5}, \underbrace{[1, 4, 4, 2, 3]}_{\text{rekursiv sortieren}}$

Legen Sie eine Klasse `Quicksort` an, in welcher Sie folgende Methoden implementieren:

- Die Methode `public static int partition(int[] data, int l, int r)` partitioniert ein Subarray `data[l, r]` wie oben beschrieben. Rückgabewert ist der dabei gefundene Wert m . Sie dürfen in der Methode `partition` keine Hilfsarrays verwenden (die Partitionierung soll in-place erfolgen).
- Die Methode `public static void qsort(int[] data, int l, int r)` sortiert ein Subarray `data[l, r]` mit der oben beschriebenen rekursiven Strategie. Dazu muss die Methode `partition` aus Aufgabe 2.1(a) verwendet werden.
- Die Methode `public static void qsort(int[] data)` sortiert das gesamte Array.
- Die Methode `public static void main(String[] args)` enthält (wie immer) das ausführbare Programm ihrer Implementierung. Hier sollten Sie ein Array vom Typ `int []` von Standard-In einlesen (siehe Blatt 01, `Scanner` und `ArrayList` erlaubt) und mit `qsort` sortieren. Falls das Array weniger als 20 Elemente enthält, geben Sie es vor und nach dem Sortieren aus. Dazu dürfen Sie `Arrays.toString()` aus der Bibliothek `java.util.Arrays` verwenden. Geben Sie immer das Minimum, das Maximum, und den Median aus. Beispiel:

```
seq 5 2 15 | shuf | java Quicksort
```

```
[11, 7, 15, 16, 5, 9]
```

```
[15, 13, 11, 9, 7, 5]
```

```
Min: 5, Med: 10.0, Max: 15
```

```
seq 1337 -5 42 | java Quicksort
```

```
Min: 42, Med: 689.5, Max: 1337
```

Languaufgabe 2.2: Dual-Pivot Quicksort

(3 Punkte)

Eine Variante von Quicksort erzielt in der Praxis oftmals schnellere Laufzeiten, indem sie zwei Pivot-Elemente verwendet. Die Partitionierung von $A[l, r]$ funktioniert dann wie folgt: Seien $p_1 = A[l]$ und $p_2 = A[r]$ das erste und letzte Element des Subarrays *vor der Partitionierung* (die beiden Pivot-Elemente). Die Elemente des Subarrays werden nun umgeordnet, sodass für zwei Positionen $m_1, m_2 \in \{l, \dots, r\}$ gilt:

- $A[m_1] = \max(p_1, p_2)$ und $A[m_2] = \min(p_1, p_2)$, und
- $\forall i \in \{l, \dots, m_1\} : A[i] \geq \max(p_1, p_2)$, und
- $\forall i \in \{m_2, \dots, r\} : A[i] \leq \min(p_1, p_2)$, und
- $\forall i \in \{m_1, \dots, m_2\} : \max(p_1, p_2) \geq A[i] \geq \min(p_1, p_2)$.

Jetzt müssen nur noch die Intervalle $A[l, m_1 - 1]$, $A[m_1 + 1, m_2 - 1]$ und $A[m_2 + 1, r]$ rekursiv sortiert werden. Der erste Partitionierungsschritt für das obige Beispiel sieht wie folgt aus:

$[5, 8, 1, 4, 4, 9, 2, 3] \rightarrow \underbrace{[8, 9]}_{\text{rekursiv sortieren}}, \overset{\text{Pivot}}{5}, \underbrace{[4, 4]}_{\text{rekursiv sortieren}}, \overset{\text{Pivot}}{3}, \underbrace{[1, 2]}_{\text{rekursiv sortieren}}$

Implementieren Sie in einer Klasse `Quicksort2` die Dual-Pivot-Variante von Quicksort. Als Startpunkt können Sie einfach Ihre Lösung von Aufgabe 2.1 kopieren. Die Methode `partition` muss in dieser Variante natürlich zwei Werte (m_1 und m_2) zurückgeben. Dazu sollten Sie ein Array vom Typ `int []` der Länge 2 verwenden.

Languaufgabe 2.3: Assertions

(1 Punkt)

Implementieren Sie eine Methode `public static boolean isSorted(int[] data)`, die überprüft, ob das gegebene Array absteigend sortiert ist. Verwenden Sie `isSorted` im Zusammenspiel mit dem Schlüsselwort `assert` in der `main`-Methode, um die Korrektheit von `qsort` nach der Ausführung zu verifizieren.

Hinweis: Sie müssen den Code mit der Flag `-ea` ausführen: `java -ea Quicksort`

Languaufgabe 2.4: Laufzeitmessung

(1 Punkt)

Messen Sie die Laufzeit Ihrer Implementierungen mithilfe der Bibliotheken `java.time.Instant` und `java.time.Duration` (Beispiel unten). Sie sollten sicherstellen, dass *nur die zum Sortieren benötigte Zeit* gemessen wird, und nicht die Zeit zum Lesen der Eingabe!

Wie lange brauchen Sie, um eine zufällige Permutation von $\{1, \dots, 2^{22}\}$ zu sortieren? Ist Ihre Dual-Pivot-Variante schneller als Ihre Single-Pivot-Variante?

Um aussagekräftige Messwerte zu erhalten, sollten Sie die Messung mehrfach ausführen. Der Median von fünf gemessenen Zeiten ist ein guter Richtwert.

```
Instant start = Instant.now();
// Code, dessen Laufzeit Sie messen wollen
// ...
Instant finish = Instant.now();
long time = Duration.between(start, finish).toMillis();
System.out.println("Time: " + time);
```
