technische universität
dortmund

Fakultät für Informatik
Lehrstühle 2, 8, 12

Prof. apl Dr. Beate Bollig
K. Brandenbusch, P. Oberdiek, E. Thordsen

Sommersemester 2020
30. April

DAP2 Praktikum – Blatt 3

Abgabe: 11. Mai

Kurzaufgabe 3.1: BubbleSort (4 Punkte)

Legen Sie eine Klasse `BubbleSort` an und implementieren Sie den BubbleSort-Algorithmus (siehe Hinweis) in einer Methode namens `public static void bubbleSort(int[] array)`, welcher ein Feld aufsteigend sortiert. Schreiben Sie die `main`-Methode, die ein Feld mit 50000 Elementen in absteigender Reihenfolge initialisiert und die Laufzeit (in Sekunden, siehe Hinweis) ausgibt. Das Befüllen des Feldes soll in eine zusätzliche Methode ausgelagert werden.
Beispiel für einen Aufruf:

```
Input:  java BubbleSort
Output: 2.096s
```

Kurzaufgabe 3.2: Geometrische Suche / Tertiäre Suche (4 Punkte)

In dieser Aufgabe soll die Suchstrategie „Geometrische Suche“ erprobt werden. Gesucht wird die Länge eines Arrays, so dass das Sortieren des Arrays mittels BubbleSort in guter Näherung eine vorgegebene Zeit in Sekunden dauert. Dies soll vollständig automatisiert erfolgen, d.h. das zu entwickelnde Programm führt alle unten beschriebenen Schritte selbstständig aus. Die Felder sind hier der Einfachheit halber alle absteigend zu initialisieren. Gehen Sie wie folgt vor:

- Legen Sie eine weitere Klasse `BubbleSortSuche` an.
- Übernehmen Sie ihre `main`-Methode von Aufgabe 3.1 und passen Sie diese so an, dass ein `float` als Parameter auf der Kommandozeile übergeben werden kann. Dieser entspricht der vorgegebenen Zeit in Sekunden. Behandeln Sie auch Fehleingaben indem Sie einen Text ausgeben der mit `"FEHLER: "` beginnt und eine sinnvolle Beschreibung des Fehlers und der erlaubten Eingaben enthält.
- Schreiben Sie eine Methode `public static double bubbleSort(int[] array)`, welche ein Feld sortiert und die benötigte Zeit in Sekunden zurückgibt.
- Beginnen Sie die Suche mit einer initialen Feldgröße von 1000 Elementen.
- Wenn BubbleSort auf diesem Feld schneller als die eingegebene Zeit ist, wird die Feldgröße verdoppelt. Dies wird so oft wiederholt, bis BubbleSort länger als die eingegebene Zeit benötigt. Geben Sie bei jedem dieser Schritte die aktuelle Feldgröße sowie die benötigte Laufzeit für dieses Feld in der Form `Size: 2000, Time: 0.002s` aus.

- Zwischen den letzten beiden Feldgrößen (eine mit kürzerer und eine mit längerer Laufzeit) wird eine *tertiäre Suche* gestartet. Das Konzept der *binären Suche* ist bereits aus der Vorlesung bekannt. Die *tertiäre Suche* unterscheidet sich von der *binären Suche* in der Anzahl der Aufteilungen des Suchbereiches. Anstatt den Suchbereich in zwei gleichgroße Teile zu teilen (*binär*) wird der Suchbereich in drei gleichgroße Teile geteilt (*tertiär*).
- Geben Sie während der tertiären Suche alle vier Feldgrößen (untere und obere Intervall Grenzen sowie zwei Werte die das Intervall in drei gleichgroße Teile teilen) mit der zugehörigen Laufzeit (in Sekunden, siehe Hinweis) an. Verwenden Sie dabei das Ausgabeformat

  ```
  Sizes: 64000, 85333, 106666, 128000
  Times: 1.088s, 1.928s, 3.022s, 4.404s
  ```

- Wenn die Laufzeit weniger als $0,1$ Sekunden von der eingegebenen Laufzeit abweicht, wird die Suche terminiert und die Feldgröße mit der zugehörigen Laufzeit ausgegeben. Dies soll wieder im Format `Size: 71111, Time: 1.344s` erfolgen.

Beispiel für einen Programmaufruf:

```
Input:  java BubbleSortSuche 1.3
Output: Size: 1000, Time: 0.003s
        Size: 2000, Time: 0.002s
        Size: 4000, Time: 0.005s
        Size: 8000, Time: 0.018s
        Size: 16000, Time: 0.069s
        Size: 32000, Time: 0.27s
        Size: 64000, Time: 1.088s
        Size: 128000, Time: 4.404s
        Sizes: 64000, 85333, 106666, 128000
        Times: 1.088s, 1.928s, 3.022s, 4.404s
        Sizes: 64000, 71111, 78222, 85333
        Times: 1.088s, 1.339s, 1.619s, 1.928s
        Size: 71111, Time: 1.339s
```

Beachten Sie die Hinweise und Tipps auf den folgenden Seiten.

Hinweise und Tipps

BubbleSort-Pseudocode

Procedure BUBBLESORT(Array A)

```
n ← LENGTH[A]
for i ← 1 to n do
    for j ← n downto i + 1 do
        if A[j − 1] > A[j] then
            tmp ← A[j]
            A[j] ← A[j − 1]
            A[j − 1] ← tmp
```

Messung von Laufzeiten, Laufzeitschwankungen

Um Laufzeiten in Programmen zu messen, kann man die Methode `System.currentTimeMillis()` benutzen. Diese gibt die Systemzeit in Millisekunden zurück und hat den Rückgabetyp `long`. Ein Beispiel zur Laufzeitmessung sieht dann wie folgt aus:

```
...
long tStart, tEnd;
double secs;
...
// Beginn der Messung
tStart = System.currentTimeMillis();

// Hier wird der Code ausgeführt, dessen Laufzeit gemessen werden soll

// Ende der Messung
tEnd = System.currentTimeMillis();

// Die vergangene Zeit ist die Differenz von tStart und tEnd
secs = (tEnd - tStart) / 1000.0;
...
```

Zu Aufgabe 3.2: Diverse Effekte (wechselnde CPU-Auslastung der Poolrechner, wechselnde CPU-Taktungen bei Laptops und anderen Rechnern) können zu Schwankungen in der Laufzeit führen. Falls dies bei Ihnen auftreten sollte, können Sie bei Aufgabe 3.2 auch eine höhere Toleranzgrenze als 0,1 Sekunden wählen.

Best Practice (empfehlenswert, aber nicht Gegenstand der Punktwertung): Neben den oben genannten Effekten kann auch der Garbage-Collector einen Einfluss auf die Laufzeit haben. Bei einzelnen Messungen kann dies zu großen Laufzeitunterschieden führen. Daher ist es empfehlenswert, jede Messung mehrmals zu wiederholen und den Mittelwert der Laufzeiten zu bestimmen. Dem Garbage-Collector kann *zwischen* den Messungen mittels `System.gc()` empfohlen werden, den Speicher aufzuräumen. Vorher müssen die betreffenden Referenzen auf `null` gesetzt sein, damit der ehemals referenzierte Speicher freigegeben werden kann.