HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Algorithmen und Datenstrukturen (IB) Praktikum
Aufgabenblatt 4 – Algorithmenanalyse Korrektur

Ausgabe: 12. April 2018
Abgabe: 17. April 2018, 18:00 Uhr

1 O-Notation (Pflichtaufgabe

1.1 Nachweis für O-Notation

Geben Sie für die folgenden Funktionenpaare f, g jeweils ein n_0 und eine möglichst scharfe Konstante c an, die belegen, dass $f(n) \in O(g(n))$! Beachten Sie: $\log n$ ist eine Kurzschreibweise für $\log(n)$; die Anwendung des Logarithmus hat dann Vorrang vor allen anderen Operatoren.

- $f(n) = 3n^2 + 18n + 5$, $g(n) = n^3$ — n0 > 0; c=26
- $f(n) = 1000$, $g(n) = 1$ — n0 beliebig; c=1001
- $f(n) = n \log n + n$, $g(n) = n \log n$ — n0 > 10; c=2

1.2 Vereinfachung

Vereinfachen Sie die folgenden Funktionsmengen im O-Kalkül so weit wie möglich!

- $O(n \cdot (4n^2 + n \log n))$ — = O(4n³ + n² log n) = O(4n³) = O(n³)
- $O(n^2 \log n + 4n^3)$ — = O(4n³) = O(n³)
- $O(2^n + 3^n \log n)$ — = O(3^n log n)

1.3 Zuordnung

Ordnen Sie Sie die folgenden Funktionsmengen im O-Kalkül nach ihrer Inklusion in der Form $A \subset B = C \subset D \subset E$ (Anordnung von echter Inklusion und Äquivalenz kann abweichen)!

- $O(n^2)$
- $O(n^4 + 3n^3 + 4)$
- $O(12)$
- $O(n \log n)$
- $O(n \cdot (n + \log n))$
- $O(2^n)$

$$O(12) \subset O(n \log n) \subset O(n^2) = O(n \cdot (n + \log n)) \subset O(n^4 + 3n^2 + 4) \subset O(2^n)$$

2 Schnittmengen (nur Teilaufgabe 2.1 Pflicht, Rest optional)

In den folgenden Teilaufgaben geht es darum, Methoden zu implementieren, die zu zwei gegebenen Arrays mit Ganzzahlen ermitteln, wie viele Zahlen in beiden Arrays vorkommen. Sie können davon ausgehen, dass die beiden Arrays jeweils keine Zahl doppelt enthalten.

Im Moodle finden Sie ein Maven-Projekt mit einer Klasse IntArrayUtils, in der die zu implementierenden Methoden bereits angelegt sind. Ebenfalls sind einige wenige Tests hinterlegt. Für die Implementierung verwenden Sie bitte keine Bibliotheken oder das Collections-Framework. Die Sprachmittel, mit denen die Beispiele aus der Vorlesung implementiert waren, reichen zur Bearbeitung aus.

Zur Abgabe laden Sie bitte die von Ihnen veränderte Klasse `IntArraUtils` in Moodle hoch!

2.1 Schnittmenge zweier unsortierter Arrays (Pflicht)

Ergänzen Sie die Methode

```
public static int countIntersectionUnsorted(int[] unsorted, int[] unsorted2)
```

die in einer Laufzeit in $O(n^2)$ die Anzahl der Zahlen ermittelt, die in beiden Eingabe-Parametern vorkommen, wobei n die Größe der beiden Arrays (zusammen) bezeichnet. Gehen Sie davon aus, dass beide Arrays für sich genommen keine doppelten Einträge enthalten und nicht sortiert sind. Tipp: Wenn Sie sich unsicher sind, ob Ihr Algorithmus die gewünschte Laufzeit einhält, dann stellen Sie sich vor, beide Arrays hätten die Größe n. Wenn der Algorithmus dann eine Laufzeit in $O(n^2)$ hat, hat er es auch, wenn die Arrays zusammen n Zahlen enthalten. Beispiel:

```
countIntersectionUnsorted({2,9,5,6,17,11}, {17,1,100,5}) = 2
```

2.2 Schnittmenge eines sortierten und eines unsortierten Arrays (optional)

Ergänzen Sie die Methode

```
public static int countIntersectionOneInputOrdered(int[] unsorted, int[] sorted)
```

die in einer Laufzeit in $O(nlog(n))$ die Anzahl der Zahlen ermittelt, die in beiden Eingabe-Parametern vorkommen, wobei n die Größe der beiden Arrays (zusammen) bezeichnet. Gehen Sie davon aus, dass beide Arrays für sich genommen keine doppelten Einträge enthalten und das Array `unsorted` nicht sortiert ist, wohingegen `sorted` aufsteigend sortiert ist. Der Tipp aus Teilaufgabe 1 gilt analog. Tipp: Sie können die Implementierung der Binären Suche aus der Vorlesung verwenden, die in der Klasse bereits enthalten ist! Beispiel:

```
countIntersectionOneInputOrdered({2,9,6,5,1,11},{1,5,6,12,13} ) = 3
```

2.3 Schnittmenge zweier sortierter Arrays (optional)

Ergänzen Sie die Methode

```
public static int countIntersectionOrdered(int[] sorted, int[] sorted)
```

die in einer Laufzeit in $O(n)$ die Anzahl der Zahlen ermittelt, die in beiden Eingabe-Parametern vorkommen, wobei n die Größe der beiden Arrays (zusammen) bezeichnet. Gehen Sie davon aus, dass beide Arrays für sich genommen keine doppelten Einträge enthalten, aber aufsteigend sortiert sind. Der Tipp aus Teilaufgabe 1 gilt analog. Beispiel:

```
countIntersectionOrdered({1,2,5,6,9,11},{1,5,6,12,13} ) = 3
```

2.4 Laufzeitmessung

Die Main-Methode der Klasse `IntArrayUtil` enthält Code, mit dem sich grobe Zeitmessungen vornehmen lassen. Ermitteln Sie die Laufzeiten der drei implementierten Methoden für Eingaben der Größe 10.000, 100.000, 1.000.000 und 10.000.000 Elementen je Eingabearray. Sofern die Laufzeit für den unsortierten Fall 5 Minuten übersteigt, können Sie diese Werte weglassen.