Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
Arbeitsgruppe Datenbanksysteme

Sommersemester 2023
30. 04. 2023

Prof. Dr. Bernhard Seeger
Jana Holznigenkemper M.Sc.
Marius Kuhrt M.Sc.

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Abgabe: 07. 05. 2023,
bis **spätestens** 23:59 Uhr
über die ILIAS Plattform

Übung 3

Aufgabe 3.1: Analyse durch Rekursionsgleichung (3 Punkte)

Lösen Sie die folgende Rekursionsgleichung durch iteratives Einsetzen. Verwenden Sie zum Lösen der Gleichung keine Abschätzung, d.h. lösen Sie die Gleichung vollständig auf. Geben Sie anschließend eine Abschätzung durch die $\mathcal{O}$-Notation an.

$$\begin{aligned} T(0) &= 0 \\ T(n) &= T(n-1) + n^2 \end{aligned}$$

Folgende Formel dürfen Sie ohne Beweis verwenden:

$$\sum_{j=0}^{n} j^2 = \frac{1}{6} n(n+1)(2n+1)$$

Aufgabe 3.2: Amortisierte Analyse (3 Punkte)

In dieser Aufgabe geht es darum n Elemente in ein Array einzufügen. Dieses Array ist zu Beginn 1 Element groß und soll dynamisch wachsen. Die naive Strategie hierfür ist, bei jedem Einfügen ein neues Array zu erzeugen, das um ein Element größer ist. Der Inhalt des alten Arrays wird dann in das neue kopiert.

Bei der Doubling-Strategie wird ein doppelt so großes Array erzeugt, wenn es keinen Platz für ein neues Element gibt. Zeigen Sie, dass die Doubling-Strategie effizienter ist, indem Sie die Gesamtlaufzeit der naiven und der Doubling-Strategie angeben und begründen. Dabei soll nur das Kopieren eines Datensatzes als Elementaroperation für die Kosten betrachtet werden.

Geben Sie außerdem die amortisierten und Best/Worst-Case Kosten für einen einzelnen Einfügevorgang an. Begründen Sie die jeweiligen Kosten.

Aufgabe 3.3: Implementierung von COLA (4+2+2) (8 Punkte)

In der Vorlesung wurde das *Cache-Oblivious Lookahead Array* (COLA), eine Array-basierte Datenstruktur für schnelles Einfügen und Suchen, präsentiert (Folie 121ff.). In dieser Aufgabe wird diese Datenstruktur implementiert und auf Ihre Performance in der Praxis untersucht.

a) Vervollständigen Sie die Klasse `COLAImpl`, welche das zur Verfügung gestellte Interface

`Insert` implementiert. Realisieren Sie mit Ihrer Implementierung die Einfügelogik der COLA Datenstruktur. Die Daten für die jeweiligen Arrays sollen entsprechend in generischen Arrays verwaltet werden.

b) Erweitern Sie Ihre Klasse `COLAImpl`, sodass dieses auch das zur Verfügung gestellte Interface `Query` erfüllt. Durchsuchen Sie in der Methode `searchElement` die Arrays in der Reihenfolge vom kleinsten zum größten.

c) Erstellen Sie eine Klasse `COLAPerformanceTest`. Erzeugen Sie in dieser ein Objekt der Klasse `COLAImpl` und ein `Integer`-Array. Fügen Sie jeweils eine große Anzahl zufälliger `Integer`-Werte in die Datenstrukturen ein (z.B. 5 Millionen). Messen Sie für jede Datenstruktur individuell die Zeit, welche für das Einfügen aller Elemente benötigt wird. Messen sie für das `Integer`-Array zusätzlich die Zeit, die zum Sortieren benötigt wird (z.B. mittels `Arrays.sort`).

Hinweis: Sie dürfen die Methode `Arrays.binarySearch` verwenden, um ein Array mit Hilfe von binärer Suche zu durchsuchen. Beachten Sie dabei die Dokumentation von `binarySearch`.

Aufgabe 3.4: MergeSort für Vertauschungen* (1+5) **(6 Punkte)**

In dieser Aufgabe wollen wir die Anzahl von Vertauschungen in einem Array finden. Dabei sei eine Vertauschung wie folgt definiert. Für ein Array A mit n einzigartigen Elementen ist ein Index-Paar (i, j) mit $0 \leq i, j \leq n$ eine Vertauschung falls $i < j$ und $A[i] > A[j]$. Diese Definition lässt sich direkt auch auf ein Array mit Duplikaten erweitern.

a) Geben Sie für das Array $[2, 5, 9, 4, 1, 13]$ die 6 existierenden Vertauschungen an.

b) Vervollständigen Sie in der Klasse `MergeSort` die Methode `sortAndCount` um eine Adapation des MergeSort Algorithmus, welche die Anzahl der Vertauschungen in einem Array mit einer Worst-Case Laufzeit von $\Theta(n \log(n))$ ermittelt. Hierbei ist n die Anzahl der Elemente im Array. Testen Sie Ihre Lösung mit einem geeigneten Beispiel.

*** Aufgabe 3.4 ist für Lehramtsstudierende optional.**