Datenstrukturen und effiziente Algorithmen I

Sommersemester 2023
Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
Lehrstuhl für Datenstrukturen und effiziente Algorithmen

Übungsblatt 4

Abgabe: 09. Mai 2023

Aufgabe 1: MERGE-SORT: Rekursionsbaum (3 Punkte)

Wenden Sie MERGESORT (siehe Folie 89) auf folgendes Beispiel an und stellen Sie den Ablauf des Algorithmus nachvollziehbar dar:

$$A = [5, 4, 3, 1, 6, 8, 2, 7, 9], n = 9$$

Machen Sie dabei deutlich, welche Teilfelder in jedem Aufruf von `MergeSort` und `Merge` betrachtet werden.

Aufgabe 2: MERGE-SORT: Implementierung (2+1+4 Punkte)

(a) Implementieren Sie den Sortieralgorithmus MERGE-SORT (siehe Folien 89, 116) zum Sortieren ganzer Zahlen. Schreiben Sie dazu eine Klasse `MergeSort` mit der Methode

```
public class MergeSort {
   public static void sort(int[] A) { ... }
}
```

(b) Erweitern Sie Ihre Klasse `MergeSort` um eine `main`-Methode, welche eine Datei mit ganzen Zahlen einliest, diese mittels Ihrer `sort`-Methode sortiert, und in sortierter Reihenfolge wieder zeilenweise auf der Konsole ausgibt. Ihr Programm soll von der Kommandozeile mit der folgenden Syntax aufgerufen werden:

```
java MergeSort <filename>
```

wobei `<filename>` den Pfad zu einer Textdatei bezeichnet, in welcher zeilenweise ganze Zahlen gespeichert sind.

Testen Sie Ihr Programm an den im StudIP verfügbaren Testinstanzen.

(c) Messen Sie die Laufzeit Ihrer Sortierfunktion für jede der im StudIP vorhandenen Testinstanzen. Wiederholen Sie diese Messungen drei Mal und bilden Sie für jede Instanz eine durchschnittliche Laufzeit. Vergleichen Sie Ihre Messungen mit den Laufzeiten von `InsertionSort`. Stellen Sie dazu die durchschnittlichen Laufzeiten beider Algorithmen in Abhängigkeit von der Anzahl n der zu sortierenden Zahlen grafisch dar. Welches Sortierverfahren schlägt sich besser?

Aufgabe 3: MERGE-INSERTION-SORT: Implementierung (2+3 Punkte)

Modifizieren Sie den Algorithmus MERGE-SORT (siehe Folien 89, 116), sodass die Rekursion nicht länger bei einem Teilfeld der Restgröße 1, sondern bei k oder weniger Elementen abbricht. Das verbleibende Restfeld soll dann mit INSERTION-SORT fertig sortiert werden.

(a) Implementieren Sie die oben beschriebene Idee. Schreiben Sie dazu eine Klasse

```
class MergeInsertionSort {
public static void sort(int [] A, int k) { ... }
}
```

Erweitern Sie Ihre Klasse `MergeInsertionSort` um eine `main`-Methode, welche eine Datei mit ganzen Zahlen einliest, diese mittels Ihrer `sort`-Methode sortiert, und in sortierter Reihenfolge wieder zeilenweise auf der Konsole ausgibt. Ihr Programm soll von der Kommandozeile mit der folgenden Syntax aufgerufen werden:

```
java MergeInsertionSort <filename> <k>
```

wobei `<filename>` den Pfad zu einer Textdatei bezeichnet, in welcher zeilenweise ganze Zahlen gespeichert sind und `<k>` für die Größe des Restfeldes bei Rekursionsabbruch steht.

Testen Sie Ihr Programm für verschiedene Werte von k.

(b) Bestimmen Sie experimentell einen möglichst optimalen Wert des Parameters k. Messen Sie dazu wiederholt die Laufzeit Ihrer Methode für konstantes n und verschiedene Werte von k. Welche Werte von k stellen sich als optimal heraus? Beschreiben Sie Ihre Ergebnisse übersichtlich und nachvollziehbar.

Aufgabe 4: MERGE-INSERTION-SORT: Analyse (3+2 Punkte)

Betrachten Sie folgendes einfaches Modell für die Laufzeit des Algorithmus aus Aufgabe 3:

$$T(n,k) = \begin{cases} 2 \cdot T(n/2,k) + c_1 \cdot n, & n > k \\ c_2 \cdot n^2, & n \leq k. \end{cases}$$

Die Konstanten c_1 und c_2 sollen dabei die (uns unbekannten) Konstanten beschreiben, die sich in der asymptotischen Laufzeitabschätzung der Funktionen MERGE bzw. INSERTION-SORT verbergen.

(a) Leiten Sie mittels eines Rekursionsbaums eine nicht-rekursive Darstellung der Laufzeitfunktion $T(n,k)$ her. Bedenken Sie, dass die Kosten in den Blättern des Baumes den Aufwand von INSERTION-SORT ($T(n) = c_2 \cdot n^2$) darstellen.

Tipp: Sie können für Ihre Analyse annehmen, dass $n = k \cdot 2^h$ für $h \in \mathbb{N}$ ist.

(b) Bestimmen Sie analytisch, bei welchem Wert von k die Rekursion optimalerweise abgebrochen werden sollte. Leiten Sie dazu die Laufzeitfunktion $T(n, k)$ nach k ab, um ein lokales Minimum der Laufzeitfunktion zu bestimmen.

(c) Bonus (+3): Bestimmen Sie experimentell die Konstanten c_1 und c_2. Messen Sie dazu die Laufzeit der Methode `Merge` für ein festes, geeignet großes n und teilen diese durch n. Für c_2 messen Sie die Laufzeit von `InsertionSort` und teilen diese durch n^2.

(d) Bonus (+2): Berechnen Sie anhand der experimentell bestimmten Werte für c_1 und c_2 den Wert des optimalen k. Stimmt dieser mit der experimentellen Beobachtung aus Aufgabe 3 überein?