Universität Stuttgart

Übungsblatt 1

Datenstrukturen und Algorithmen (SS 2022)

Abgabe: Montag, 25.04.2022, 15:30 Uhr — Besprechung: ab Montag, 02.05.2022

Abgabevorschriften: Die Aufgaben auf diesem Blatt sind unter Einhaltung der Abgabevorschriften[1] zu lösen und abzugeben.

Lernziele: Nach dem Tutorium zu diesem Blatt sollten Sie folgende Lernziele erreicht haben. Wenn nicht, zögern Sie nicht, ihre:n Tutor:in anzusprechen um die Lücken zu füllen, Unklarheiten zu klären oder Fragen zu beantworten.

- Sie verstehen, wie die binäre und sequentielle Suche funktioniert und können die Algorithmen anwenden.
- Sie kennen die Unterschiede und Grenzen dieser Suchalgorithmen.
- Sie verstehen, wie SelectionSort, InsertionSort, BubbleSort, MergeSort und QuickSort funktionieren und können diese Algorithmen anwenden.
- Sie können diese Sortieralgorithmen implementieren.

Punkte: Dieses Übungsblatt beinhaltet 5 Aufgaben mit einer Gesamtzahl von 30 Punkten. Zum Bestehen werden also 15 Punkte benötigt.

Aufgabe 1 Verständnisfragen zu Suchalgorithmen *[Punkte: 4]*

Beantworten Sie folgende Verständnisfragen zur binären und sequentiellen Suche.

(a) *(1 Punkt)* Welches der behandelten Suchverfahren ist im Schnitt das schnellste Suchverfahren auf einer sortierten Liste? Begründen Sie Ihre Antwort. Eine Antwort ohne Begründung wird mit null Punkten bewertet.

(b) *(1 Punkt)* Nennen Sie eine Anwendung, bei der die *binäre* Suche nicht möglich ist, aber die *sequenzielle*. Begründen Sie Ihre Antwort. Eine Antwort ohne Begründung wird mit null Punkten bewertet.

(c) *(1 Punkt)* Konstruieren Sie eine Beispielliste, auf der die *sequentielle* Suche einen von Ihnen gewählten Wert schneller findet, als die *binäre*. Begründen Sie Ihre Antwort. Eine Antwort ohne Begründung wird mit null Punkten bewertet.

(d) *(1 Punkt)* Nennen Sie die zwei ungünstigsten Positionen in einer Liste mit neun Elementen für die binäre Suche. Anders formuliert: Wo muss das gesuchte Element stehen, damit die binäre Suche möglichst viele Schritte benötigt. Begründen Sie Ihre Antwort. Eine Antwort ohne Begründung wird mit null Punkten bewertet.

Aufgabe 2 Anwendung von Suchverfahren *[Punkte: 4]*

Gegeben sind die folgenden zwei Listen, auf die Sie in dieser Aufgabe die in der Vorlesung vorgestellten Suchverfahren selbst anwenden werden:

Gesuchtes Element: **weiß**

gelb	lila	blau	rot	grün	grau	weiß	schwarz

Gesuchtes Element: **H**

A	C	D	E	G	H	I	Z

[1] https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_file_2904210.html

(a) *(2 Punkte)* Wenden Sie (wenn möglich) den in der Vorlesung vorgestellten Algorithmus zur *sequenziellen* Suche auf die zwei gegebenen Listen an. Falls die *sequenzielle* Suche möglich ist, führen Sie dabei die einzelnen Schritte auf, damit deutlich wird, wie diese schrittweise abläuft. Nutzen Sie hierfür die in den Vorlesungsfolien vorgestellte Darstellung. Falls die *sequenzielle* Suche nicht möglich ist, begründen Sie, warum nicht.

(b) *(2 Punkte)* Wenden Sie (wenn möglich) den in der Vorlesung vorgestellten Algorithmus zur *binären* Suche auf die zwei gegebenen Listen an. Falls die *binäre* Suche möglich ist, führen Sie dabei die einzelnen Schritte auf, damit deutlich wird, wie diese schrittweise abläuft. Nutzen Sie hierfür die in den Vorlesungsfolien vorgestellte Darstellung. Falls die *binäre* Suche nicht möglich ist, begründen Sie, warum nicht.

Aufgabe 3 Verständnisfragen zu Sortieralgorithmen *[Punkte: 4]*

Beantworten Sie folgende Verständnisfragen zur Sortieralgorithmen aus der Vorlesung.

(a) *(1 Punkt)* Vergleichen Sie die beiden Sortierverfahren *SelectionSort* und *InsertionSort* miteinander und nennen Sie zwei wesentliche Eigenschaften, in denen sie sich unterscheiden.

(b) *(1 Punkt)* Wenn nur *SelectionSort* und *InsertionSort* zur Auswahl stehen, welches der beiden Verfahren ist in der Regel vorzuziehen? Begründen Sie Ihre Antwort.

(c) *(1 Punkt)* Auch Daten aus Datenbanken müssen häufig sortiert werden, bevor sie ausgegeben werden können. Allerdings ist der Zugriff auf den Festplattenspeicher, in dem die Daten eigentlich liegen ziemlich teuer. Arbeiten im Arbeitsspeicher sind hingegen um ein Vielfaches günstiger. Deshalb ist es häufig effizienter, die Daten zunächst in den Arbeitsspeicher zu laden, um sie dort zu sortieren und sie anschließend auszugeben. Selbst, wenn die Daten nicht in den Arbeitsspeicher passen und sie deswegen portionsweise sortiert und zwischendurch auf einer Festplatte zwischengespeichert werden müssen, ist diese Strategie trotzdem generell dem Sortieren auf der Festplatte überlegen. even if

Welche zusätzliche (in der Vorlesung behandelte) Eigenschaft von Sortieralgorithmen ist in solchen Situationen kein Nachteil mehr? Welche Sortieralgorithmen erfüllen diese Eigenschaft?

(d) *(1 Punkt)* Beschreiben Sie eine Situation, in der (im Gegensatz zum vorherigen Beispiel) der verwendete Speicherplatz optimiert werden muss und deswegen diese eben erwähnten Algorithmen wahrscheinlich nicht ausgewählt werden würden. Begründen Sie Ihre Antwort.

Aufgabe 4 Anwendung von Sortierverfahren *[Punkte: 6]*

Gegeben ist die folgende Liste, auf die Sie in dieser Aufgabe Sortierverfahren selbst anwenden werden:

15	12	1	4	17	26

(a) *(3 Punkte) MergeSort.* Wenden Sie (wenn möglich) den in der Vorlesung vorgestellten Algorithmus zu *MergeSort* auf die gegebene Liste an, um sie **aufsteigend** zu sortieren. Falls die Anwendung möglich ist, führen Sie dabei die einzelnen Schritte auf, damit deutlich wird, wie diese schrittweise abläuft. Nutzen Sie hierfür die in den Vorlesungsfolien vorgestellte Darstellung. Geben Sie **zusätzlich** im letzten Schritt an, welche **Zahlenpaare** bei der Vereinigung verglichen werden, um die finale, sortierte Liste zu erhalten. Falls die Anwendung nicht möglich ist, begründen Sie, warum nicht.

(b) *(3 Punkte) QuickSort.* Wenden Sie (wenn möglich) den in der Vorlesung vorgestellten Algorithmus zu *QuickSort* auf die gegebene Liste an, um sie **aufsteigend** zu sortieren. Falls die Anwendung möglich ist, führen Sie dabei die einzelnen Schritte auf, damit deutlich wird, wie diese schrittweise abläuft. Nutzen Sie hierfür die in den Vorlesungsfolien vorgestellte Darstellung. Wählen Sie als Pivot-Element immer das mittlere Element, bzw. bei einer geraden Anzahl das Element links der Mitte. Geben Sie **zusätzlich** für jede Iteration die **Menge der Pivot-Elemente** an und **markieren Sie** das Pivot-Element und die bereits fixierten Elemente. Falls die Anwendung nicht möglich ist, begründen Sie, warum nicht.

Aufgabe 5 Impl Implementierung von Sortierverfahren *[Punkte: 12]*

In dieser Aufgabe sollen Sie einige Sortieralgorithmen selbst implementieren. Im Eclipse-Projekt gegeben sind hierfür:

- `ISimpleList`: ein Interface für Listen, deren Elemente das Interface `Comparable` implementieren.
- `SimpleList`: eine Klasse, die das Interface `ISimpleList` implementiert und im Rahmen dieser Aufgabe **nicht** zu verändern ist.
- `Sorter`: eine Klasse, die um die Implementierungen der Sortieralgorithmen ergänzt werden soll.

Implementieren Sie die folgenden drei Sortierverfahren jeweils als statische Methode der Klasse `Sorter`, die jeweils eine Liste `ISimpleList` als Eingabeparameter erwarten, welche durch die Methode sortiert wird.

(a) *(4 Punkte) SelectionSort.* Dieses Verfahren soll eine Liste in **absteigender** Reihenfolge sortieren, indem es das jeweils *größte* Element im unsortierten Teil der Liste sucht und es mit dem Anfang des unsortierten Teils der Liste vertauscht.

Achten Sie hierbei auch auf die zugehörigen Kommentare in der Klasse `Sorter`.

(b) *(4 Punkte) InsertionSort.* Dieses Verfahren soll eine Liste in **absteigender** Reihenfolge sortieren, indem es zunächst nur das erste Element der Liste als sortierte Teilliste betrachtet und dann nach und nach die weiteren Elemente der noch unsortierten Teilliste hinzunimmt um sie an der richtigen Stelle in der bereits sortierten Teilliste einzufügen. Dies geschieht indem das nächste Element so weit nach vorne rückt bis die Elemente links davon größer sind und Elemente mit kleineren Werten nach hinten rücken.

Achten Sie hierbei auch auf die zugehörigen Kommentare in der Klasse `Sorter`.

(c) *(4 Punkte) BubbleSort.* Dieses Verfahren soll eine Liste in **absteigender** Reihenfolge sortieren, indem es beim Durchlaufen der unsortierten Liste jeweils das aktuelle Element mit seinem rechten Nachbarn vergleicht und diese vertauscht, falls das aktuelle Element kleiner als sein rechter Nachbar ist. Die Liste wird so oft durchlaufen, bis es keine Änderungen mehr gibt.

Achten Sie hierbei auch auf die zugehörigen Kommentare in der Klasse `Sorter`.