

11. Übung

Abgabe bis 17.07.2017, 10:00 Uhr

Einzelaufgabe 11.1: Sortiervverfahren – Theorie

39 EP

Betrachten Sie in den folgenden Teilaufgaben die Buchstaben und Sonderzeichen der Zeichenkette „JohnDoe@(AuD&PFP)“ symbolweise und in Leserichtung mit Ordnung gemäß [ASCII-Tabelle](#) (also z.B. < A < D < a < u). Verwenden Sie jeweils die in den Beispielen gezeigte Darstellung! **Achtung:** Verwenden Sie exakt die in der Vorlesung beschriebenen Varianten der Sortiervverfahren!

- a) Sortieren Sie die Zeichen zuerst absteigend und stabil mittels *Sortieren durch Auswählen*. Löschen Sie dazu jeweils das Maximum aus der noch zu sortierenden Buchstabenliste l und fügen Sie es hinten an die anfangs leere Ergebnisliste l_s an. *Beispiel:*

l_s	l
	B a r
r	B a
r a	B
r a B	

- b) Sortieren Sie die Zeichen jetzt aufsteigend mittels *Sortieren durch Einfügen*. Stellen Sie den Ablauf wie vorhin tabellarisch dar.
- c) Sortieren Sie die Zeichen nun wieder absteigend aber *in-situ* mittels *Blasensortierung*. Stellen Sie den Ablauf ebenfalls in einer Tabelle (diesmal *ohne* Spalte l_s , da *in-situ*!) dar.
- d) Sortieren Sie die Zeichen jetzt wieder aufsteigend, diesmal aber mittels *Haldensortierung*. Verwenden Sie dazu die Array-Einbettung einer *Min-Halde* (d.h. *kleinstes* Zeichen „oben“, also bei $l[0]$). Stellen Sie den Ablauf in *zwei Phasen* dar: Die erste zeigt den initialen Aufbau der *Min-Halde* in l (dabei bleibt l_s leer) und die zweite den eigentlichen Sortiervorgang, bei dem das Halden-Minimum jeweils ans Ende von l_s verschoben und anschließend die Min-Halden-Eigenschaft wiederhergestellt wird. Pro Zeile der ersten Phase wird jeweils ein Element „versickert“. *Beispiel:*

l_s	l
	F A I L
	A F I L
A	F L I
A F	I L
A F I	L
A F I L	

- e) Sortieren Sie die Zeichen absteigend durch *Verschmelzen*. Führen Sie den rekursiven Abstieg zuerst in der **linken** und danach in der **rechten** Hälfte des jeweils betrachteten Intervalls. Geben Sie jeweils an, ob Sie ein Intervall gerade zerlegen (S) oder **verschmelzen** (M). *Beispiel:*

Aktion	l
	F a i l
S	F a i l
S	F a i l
M	a F i l
S	a F i l
M	a F i l
M	l i a F

- f) Sortieren Sie die Zeichen *aufsteigend durch Zerlegen*. Nach dem Partitionieren führen Sie bitte den *rekursiven Abstieg* zuerst im Intervall *vor* (L) und dann *nach* (R) dem *Pivot* und geben Sie auch das *Ergebnis nach der Rückkehr* aus beiden rekursiven Aufrufen an (Q). Wählen Sie als Pivot stets das letzte Element im Intervall (wie in der Vorlesung). *Beispiel:*

Aktion	l				
	F	a	u	l	
L	F	a		l	u
L	F		a	l	u
R	F	a		l	u
Q	F	a		l	u
R	F	a	l		u
Q	F	a	l	u	

- g) Sortieren Sie die Zeichen *absteigend durch einfaches Fachverteilen*. Legen Sie für jedes Zeichen ein eigenes Fach an, geben Sie in der Tabelle jedoch nur die Fächer mit mindestens einem Zeichen an und stellen Sie ausgelassene Fächer als graue Spalten dar. *Beispiel:*

l (Eingabe)							
F, a, l, l, e							
Buckets (Sortiervorgang)							
	F		a		e		l
	F		a		e		l
l (Ergebnis)							
l, l, e, a, F							

Geben Sie Ihre Lösung als `Sortieren.pdf` über EST ab.

Gruppenaufgabe 11.2: Verallgemeinertes Sortieren durch Fachverteilen

21 GP

Die in der Aufgabe 11.1 behandelten Sortierv Verfahren benötigen i.d.R. eine *inhärente* oder *externe* Ordnungsrelation für die zu sortierenden Elemente selbst. Etwas anders verhält es sich beim *verallgemeinerten Sortieren durch Fachverteilen*: Hierbei wird die Ordnung der zu sortierenden Elemente indirekt über die Ordnung der einzelnen Segmente definiert. In dieser Aufgabe sollen Sie eine generische Implementierung dieses Verfahrens umsetzen, die für beliebige Objekte und Segment-Typen funktioniert. Objekte, die damit sortierbar sein sollen, müssen die bereitgestellte Schnittstelle *Segmentierbar* implementieren (Beschreibung siehe Kommentare!).

Erstellen Sie die Klasse `VerallgemeinertesSortierenDurchFachverteilen` mit der öffentlichen Klassenmethode `sortiere(LinkedList<Segmentierbar>)`, die die Elemente der übergebenen Liste *stabil* mittels *Sortieren durch Fachverteilen* auf Basis der Segmente *aufsteigend* sortiert. Beachten Sie dabei, dass die Anzahl der Segmente nicht notwendigerweise gleich sein muss (auch nicht innerhalb einer zu sortierenden Liste, wie z.B. im öffentlichen Test mit verschiedenen langen Namen): Bei gleichem Segment-Präfix sollen „kürzere“ Elemente vor längeren stehen!

Beispiel: „Doe, Jane“ < „Doe, John“ < „Doe, Johnny“

Achtung: Verwenden Sie in Ihrer Implementierung *ausschließlich* die bereitgestellte Schnittstelle *Segmentierbar* sowie *nur* `Comparable`, `LinkedList` und `TreeMap` aus der Java-API!

39 EP + 21 GP = 60 Punkte