

**P** 5

## 11. Übung

Abgabe bis 17.07.2017, 10:00 Uhr

## Einzelaufgabe 11.1: Sortierverfahren – 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 <u>exakt</u> die in der Vorlesung beschriebenen Varianten der Sortierverfahren!

a) Sortieren Sie die Zeichen zuerst <u>absteigend</u> 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:

$\iota_s$		$\iota$	
	В	a	r
r		В	a
r	a		В
r	a	В	

- **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 <u>absteigend</u> aber <u>in-situ</u> mittels <u>Blasensortierung</u>. Stellen Sie den Ablauf ebenfalls in einer Tabelle (diesmal *ohne* Spalte  $l_s$ , da <u>in-situ</u>!) dar.
- d) Sortieren Sie die Zeichen jetzt wieder <u>aufsteigend</u>, diesmal aber mittels <u>Haldensortierung</u>. Verwenden Sie dazu die Array-Einbettung einer <u>Min-Halde</u> (d.h. <u>kleinstes</u> Zeichen "oben", also bei l[0]). Stellen Sie den Ablauf in <u>zwei Phasen</u> dar: Die erste zeigt den initialen Aufbau der <u>Min-Halde</u> 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". <u>Beispiel</u>:

$l_s$			ļ	
	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 <u>ab</u>steigend 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:

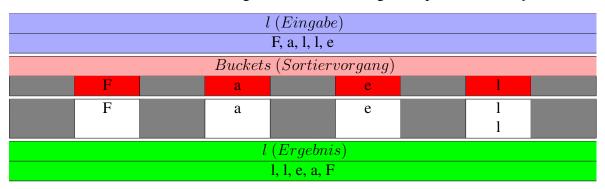
Aktion			l		
	F	a	i	1	
S	F	a		i	1
S	F		a	1	1
M	a	F		i	1
S	a	F	i		1
M	a	F	1	i	
M	1	i	a	F	

Sommersemester 2017

f) Sortieren Sie die Zeichen <u>aufsteigend durch Zerlegen</u>. 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			$\iota$		
	F	a	u	1	
L	F	a		1	
L	F		a	1	u
R	F	a		1	u
Q	F	a		1	u
R	F	a	1		u
Q	F	a	1	u	

g) Sortieren Sie die Zeichen <u>absteigend durch einfaches Fachverteilen</u>. 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:



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 Sortierverfahren 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 <u>aufsteigend</u> 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 verschieden langen Namen): Bei gleichem Segment-<u>Präfix</u> sollen "kürzere" Elemente vor längeren stehen!

Achtung: Verwenden Sie in Ihrer Implementierung <u>ausschlieβlich</u> die bereitgestellte Schnittstelle Segmentierbar sowie <u>nur</u> Comparable, LinkedList und TreeMap aus der Java-API!