DA-4

Lorenz Pfäffli 18-925-230
Lukas Ingold 20-123-998

TK

8/10

Datenstrukturen und Algorithmen
Übung 4 – Sortieren II

Abgabefrist: 25.03.2021, 16:00 h
Verspätete Abgaben werden nicht bewertet.

Theoretische Aufgaben

1. Erläutern Sie analog zu Abbildung 8.2 im Buch, wie COUNTING-SORT auf dem Feld

$$A = \langle 2,7,1,3,2,4,1,8,5,1,4 \rangle$$

arbeitet. **(1 Punkt)**

2. Gegeben sind n ganze Zahlen zwischen 0 und k. Geben Sie einen Algorithmus an, der die Eingabe vorbearbeitet und dann jede Anfrage der Art, wie viele der n Zahlen in ein Intervall $[a \ldots b]$ fallen, in Zeit $\mathcal{O}(1)$ beantwortet. Ihr Algorithmus sollte für den Vorbearbeitungsschritt mit der Zeit $\mathcal{O}(n+k)$ auskommen. **(1 Punkt)**

3. Beschreiben Sie analog zu Abbildung 8.3 im Buch die Arbeitsweise von RADIX SORT angewendet auf die folgende Liste von Wörtern: *NDB, MND, NSA, CIA, BND, MAD, BFV, FSB, KGB, BVT, DND, NIS, NSB*. **(1 Punkt)**

4. (a) Welche der folgenden Sortieralgorithmen sind (in der Variante aus der Vorlesung) stabil: INSERTIONSORT, MERGESORT, HEAPSORT, QUICKSORT? Begründen Sie kurz. **(0.4 Punkte)**
   (b) Geben Sie ein einfaches Schema an, nach dem *beliebige* vergleichende Sortieralgorithmen stabilisiert werden können. Wie viel zusätzliche Zeit und wie viel zusätzlicher Speicherplatz zieht Ihr Schema nach sich? **(0.6 Punkte)**

5. Erläutern Sie analog zu Abbildung 8.4 die Arbeitsweise von BUCKET-SORT angewendet auf das Feld

$$A = \langle 0.79,\ 0.13,\ 0.16,\ 0.64,\ 0.39,\ 0.20,\ 0.89,\ 0.53,\ 0.71,\ 0.42 \rangle$$

(1 Punkt)

6. Gegeben sei ein Feld mit ganzzahligen Werten, wobei die einzelnen Zahlen unterschiedlich viele Stellen haben können. Die Gesamtanzahl der Stellen *aller* Zahlen des Feldes hat jedoch den festen Wert n. Zeigen Sie, wie das Feld in der Zeit $\mathcal{O}(n)$ sortiert werden kann. **(1 Punkt)**

Praktische Aufgaben

In dieser Aufgabe geht es darum, Zeichenketten mit RADIX-SORT zu sortieren. Sie sollen einen verbesserten RADIX-SORT implementieren, der effizient Zeichenketten *unterschiedlicher Länge* sortiert. Wir stellen eine Implementation von RADIX-SORT auf *Ilias* zur Verfügung, auf welcher Ihre Verbesserung aufbauen soll.

1. Studieren Sie die Funktion `radixSort` in `RadixSort.java`. Was ist die Komplexität dieses Algorithmus gegeben die Anzahl Zeichenketten n und die Länge der längsten Zeichenkette d? **(1 Punkt)**

2. Entwicklen Sie einen verbesserten RADIX-SORT Algorithmus, dessen Laufzeit linear in der Summe aller Buchstaben in allen Zeichenketten ist. Die Idee ist, dass die Zeichenketten zuerst ihrer Länge nach sortiert werden. In jedem Schritt von RADIX-SORT sollen dann nur diese Zeichenketten sortiert werden, die genug lang sind. Das heisst, die Zeichenketten haben an der entsprechenden Position keinen "Leerbuchstaben". Zeigen Sie mit einem kleinen Beispiel (zehn Zeichenketten, maximale Länge fünf Buchstaben), dass Ihr Algorithmus korrekt sortiert. **(2 Punkte)**

3. Vergleichen Sie Ihren verbesserten Algorithmus mit der ursprünglichen Version. Erstellen Sie eine Grafik, welche Zeitmessungen für beide Algorithmen und verschiedene Parameter n und d zusammenfasst. Erläutern Sie, ob Ihre Messungen der Theorie entsprechen. **(1 Punkt)**

6.) n = Anz. Stellen aller Zahlen zsm.
m = Anz. Zahlen
l = Anz. Stellen der längsten Zahl

mit Radixsort:

Wc.: Eine Zahl der Länge $l = n/2$
$n/2$ Zahlen der Länge 1
↳ $m = n/2 + 1$ Zahlen $\Rightarrow \Theta(l \cdot m) \Rightarrow \Theta(n^2)$ ↯

m_i = Anzahl Zahlen mit i Stellen
die Buckets ? müssen sortiert werden: $\Theta(i \cdot m_i)$
die Summe aller Buckets $\Rightarrow \sum_{i=1}^{n} \Theta(i \cdot m_i) \rightarrow \Theta(n)$

0.5/1

3.) Unser Algorithmus läuft zwar immer noch mit worstcase O(n*d) allerdings ist er so optimiert das er sehr viel weniger zeit in Anspruch nimmt.

Da geht was schief im Alg.

Praktisch 2.) 0.5/2

(✓) 1/1

1.) A = ⟨2, 7, 1, 3, 2, 4, 1, 8, 5, 1, 4⟩

a.

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4

	0	1	2	3	4	5	6	7	8
C	0	3	2	1	2	1	0	1	1

b.

	0	1	2	3	4	5	6	7	8
C	0	3	5	6	8	9	9	10	11

c.)

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E								4			

	0	1	2	3	4	5	6	7	8
C	0	3	5	6	7	9	9	10	11

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E			1					4			

	0	1	2	3	4	5	6	7	8
C	0	2	5	6	7	9	9	10	11

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E			1					4	5		

	0	1	2	3	4	5	6	7	8
C	0	2	5	6	7	8	9	10	11

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E			1					4	5		8

	0	1	2	3	4	5	6	7	8
C	0	2	5	6	7	8	9	10	10

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E		1	1					4	5		8

	0	1	2	3	4	5	6	7	8
C	0	1	5	6	7	8	9	10	10

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E		1	1				4	4	5		8

	0	1	2	3	4	5	6	7	8
C	0	1	5	6	6	8	9	10	10

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E		1	1		2		4	4	5		8

	0	1	2	3	4	5	6	7	8
C	0	1	4	6	6	8	9	10	10

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E		1	1		2	3	4	4	5		8

	0	1	2	3	4	5	6	7	8
C	0	1	4	6	5	8	9	10	10

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E	1	1	1		2	3	4	4	5		8

	0	1	2	3	4	5	6	7	8
C	0	0	4	6	5	8	9	10	10

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E	1	1	1		2	3	4	4	5	7	8

	0	1	2	3	4	5	6	7	8
C	0	0	4	6	5	8	9	9	10

	1	2	3	4	5	6	7	8	9	10	11
A	2	7	1	3	2	4	1	8	5	1	4
E	1	1	1	2	2	3	4	4	5	7	8

	0	1	2	3	4	5	6	7	8
C	0	0	3	6	5	8	9	9	10

✓ 1/1

3.)

			⇒			⇒			⇒	SORTED		
N	D	B		N S A			M A D			B F V		
M	N	D		C I A			N D B			B N D		
N	S	A		N D B			B F V			B V T		
C	I	A		F S B			K G B			C I A		
B	N	D		K G B			C I A			D N D		
M	A	D		N S B			N I S			F S B		
B	F	V		M N D			M N D			K G B		
F	S	B		B N D			B N D			M A D		
K	G	B		M A D			D N D			M N D		
B	V	T		D N D			N S A			N D B		
D	N	D		N I S			F S B			N I S		
N	I	S		B V T			N S B			N S A		
N	S	B		B F V			B V T			N S B		

✓ 1/1

5.)

	A
1	.79
2	.13
3	.16
4	.64
5	.39
6	.20
7	.89
8	.53
9	.71
10	.42

	B
0	/
1	→ .13 → .16 /
2	→ .20 /
3	→ .39 /
4	→ .42 /
5	→ .53 /
6	→ .64 /
7	→ .79 → .71 /
8	→ .89 /
9	/

und dann..

✓ 1/1

2.) if (a == 0) {
return C[b] }
else {
return C[b] − C[a−1] } ✓

↳ for i = 0 to k ⇒ $\Theta(k)$
for j = 1 to A.length ⇒ $\Theta(n)$
for i = 1 to k ⇒ $\Theta(k)$
Auslesen ⇒ $\Theta(1)$ ✓

$\Rightarrow \Theta(k+n+k+1) = \Theta(k+n) = \Theta(n+k)$ ✓

1/1

4.) a) Insertionsort stabil ✓
Mergesort stabil ✓
Heapsort stabil bis zum Heapify-Aufruf → Also nicht stabil
Quicksort instabil wegen Austausch zwischen Pivot-Element und Element ✓

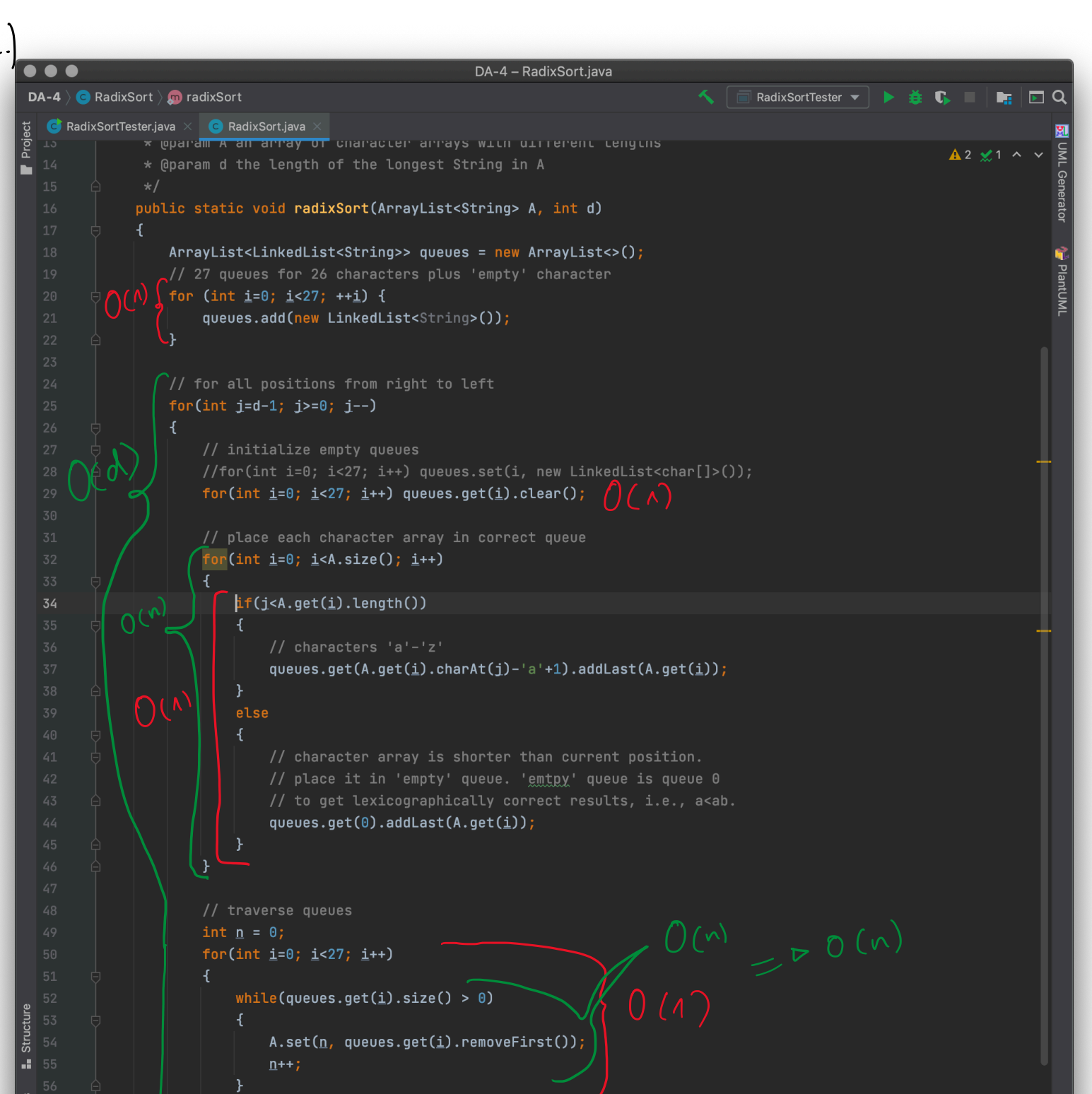
0.4/0.4

b) Beliebige Elemente werden in paare umgewandelt. Die zweite Komponente entspricht dabei jeweils der anfangsposition dann passt man die conditions an sodass jedes verglichene element in einen neuen Sortiertopf kommt welcher dann nach dem zweiten element sortiert wird hierbei bräuchte man zusäzlichen speicher um die position abzuspeichern und zusätzliche zeit um die untersortierungen zu unternehmen ✓

0.6/0.6

Programmierteil:

1.)



```java
 * @param A an array of character arrays with different lengths
 * @param d the length of the longest String in A
 */
public static void radixSort(ArrayList<String> A, int d)
{
    ArrayList<LinkedList<String>> queues = new ArrayList<>();
    // 27 queues for 26 characters plus 'empty' character
    for (int i=0; i<27; ++i) {
        queues.add(new LinkedList<String>());
    }

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        //for(int i=0; i<27; i++) queues.set(i, new LinkedList<char[]>());
        for(int i=0; i<27; i++) queues.get(i).clear();

        // place each character array in correct queue
        for(int i=0; i<A.size(); i++)
        {
            if(j<A.get(i).length())
            {
                // characters 'a'-'z'
                queues.get(A.get(i).charAt(j)-'a'+1).addLast(A.get(i));
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues.get(0).addLast(A.get(i));
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues.get(i).size() > 0)
            {
                A.set(n, queues.get(i).removeFirst());
                n++;
            }
        }
    }
}
```

O(n), O(d), O(n), O(n), O(1), O(n), O(1) ⇒ O(n)

$O(1) + O(d \cdot 2n) \Rightarrow O(d \cdot n)$ ✓

1/1