ÜBUNGSBLATT 6

Sortieralgorithmen

AUFGABE 6.1: SHAKERSORT (2 PUNKTE)

1. Implementieren Sie den Shakersort Algorithmus (siehe Vorlesung);
2. Bestimmen Sie die Laufzeit- und Speicherkomplexität des Algorithmus.

AUFGABE 6.2: COUNTINGSORT (2 PUNKTE)

Der Countingsort Algorithmus sortiert ein Array anhand folgender Schritte:

1. Liest ein Array *A* mit *N* natürlichen Zahlen ein (hier *N* = 8):

A[0]	*A*[1]	*A*[2]	*A*[3]	*A*[4]	*A*[5]	*A*[6]	*A*[7]
2	5	3	0	2	3	0	3

2. Erzeugt ein Histogramm, welches die Häufigkeit jeder Zahl im Array beschreibt. Das Arrayelement *C*[*I*] gibt an, wie oft die Zahl *I* im Gesamtarray *A* vorkommt:

C[0]	*C*[1]	*C*[2]	*C*[3]	*C*[4]	*C*[5]
2	0	2	3	0	1

3. Berechnet für jedes Element *C*[*I*] im Array *C* die Anzahl von Elementen im Array *A*, die kleiner als die Zahl *I* sind:

$$C[I] = \sum_{J=0}^{I-1} C[J].$$

C[0]	*C*[1]	*C*[2]	*C*[3]	*C*[4]	*C*[5]
0	2	2	4	7	7

4. Überträgt sukzessive die Werte aus *A* in ein sortiertes Array *B* und zwar genau an der Stelle *B*[*C*[*A*[*I*]]], die das Hilfsarray *C* für die entsprechende Zahl angibt. Nach dem Übertragen jedes Elements wird der entsprechende Wert in *C*[*I*] inkrementiert, damit das nächste gleiche Element eine Stelle weiter hinten in das Zielarray *B* eingefügt wird.

I	*A*[0]	*A*[1]	*A*[2]	*A*[3]	*A*[4]	*A*[5]	*A*[6]	*A*[7]	*C*[0]	*C*[1]	*C*[2]	*C*[3]	*C*[4]	*C*[5]	*B*[0]	*B*[1]	*B*[2]	*B*[3]	*B*[4]	*B*[5]	*B*[6]	*B*[7]
1	2	5	3	0	2	3	0	3	0	2	2	4	7	7			2					
2	2	5	3	0	2	3	0	3	0	2	3	4	7	7			2					5
3	2	5	3	0	2	3	0	3	0	2	3	4	7	8			2		3			5
4	2	5	3	0	2	3	0	3	0	2	3	5	7	8	0		2		3			5
5	2	5	3	0	2	3	0	3	1	2	3	5	7	8	0		2	2	3			5
6	2	5	3	0	2	3	0	3	1	2	4	5	7	8	0		2	2	3	3		5
7	2	5	3	0	2	3	0	3	1	2	4	6	7	8	0	0	2	2	3	3		5
8	2	5	3	0	2	3	0	3	2	2	4	6	7	8	0	0	2	2	3	3	3	5

1. Implementieren Sie den Countingsort Algorithmus;
2. Bestimmen Sie die Laufzeit- und Speicherkomplexität des Algorithmus.

AUFGABE 6.3: SWAPSORT (2 PUNKTE)

Swapsort ist ein Sortieralgorithmus, der ein Array aus paarweise verschiedenen Zahlen sortiert.

Die Idee von Swapsort ist, von jedem Element eines Arrays A mit n Elementen die Anzahl m der kleineren Werte (die in A sind) zu zählen und das Element dann mit dem Element in A[m] zu vertauschen. Somit ist sichergestellt, dass das ausgetauschte Element bereits an der richtigen, also endgültigen Stelle steht.

Nachteil dieses Algorithmus ist, dass jedes Element nur einmal vorkommen darf, da sonst keine Terminierung erfolgt.

Swapsort arbeitet in folgenden Schritten:

1. Beginne mit `i = 0`;
2. Zähle, wie viele Elemente kleiner als A[i] sind. Ist m diese Anzahl, so tausche A[i] mit A[m];
3. Ist `i = m`, so erhöhe i um 1;
4. Ist `i = n`, so ist die Sortierung beendet. Andernfalls gehe wieder zu Schritt 2.

Beispiel: `A = { 7, 8, 5, 2, 4, 9, 3, 1 }`

i	A								Schritt
0	7	8	5	2	4	9	3	1	5 Elemente sind kleiner als A[i]; Vertausche A[0] mit A[5] getauscht;
0	9	8	5	2	4	7	3	1	7 Elemente sind kleiner als A[i]; Vertausche A[0] mit A[7] getauscht;
0	1	8	5	2	4	7	3	9	0 Elemente sind kleiner als A[i]; Erhöhe i um 1;
1	1	8	5	2	4	7	3	9	6 Elemente sind kleiner als A[i]; Vertausche A[1] mit A[6] getauscht;
1	1	3	5	2	4	7	8	9	2 Elemente sind kleiner als A[i]; Vertausche A[1] mit A[2] getauscht;
1	1	5	3	2	4	7	8	9	4 Elemente sind kleiner als A[i]; Vertausche A[1] mit A[5] getauscht;
1	1	4	3	2	5	7	8	9	3 Elemente sind kleiner als A[i]; Vertausche A[1] mit A[3] getauscht;
1	1	2	3	4	5	7	8	9	0 Elemente sind kleiner als A[i]; Erhöhe i um 1;
2	1	2	3	4	5	7	8	9	0 Elemente sind kleiner als A[i]; Erhöhe i um 1;
3	1	2	3	4	5	7	8	9	0 Elemente sind kleiner als A[i]; Erhöhe i um 1;
4	1	2	3	4	5	7	8	9	0 Elemente sind kleiner als A[i]; Erhöhe i um 1;
5	1	2	3	4	5	7	8	9	0 Elemente sind kleiner als A[i]; Erhöhe i um 1;
6	1	2	3	4	5	7	8	9	0 Elemente sind kleiner als A[i]; Erhöhe i um 1;
7	1	2	3	4	5	7	8	9	Array wurde komplett durchlaufen und das Sortieren ist somit beendet.

1. Implementieren Sie den Swapsort Algorithmus;
2. Bestimmen Sie die Laufzeit- und Speicherkomplexität des Algorithmus.