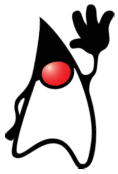
Sortieren in Java

„worst case“ - Aufwandsanalysen

Standardverfahren



BubbleSort
am Beispiel „4, 3, 2, 1“

i = 0 (geht bis 2)
j = 1 (geht bis 3)
a[j-1] > a[j] ? d.h. a[0] > a[1] ? d.h. 4 > 3 ? ja, also
temp = a[j-1] = a[0] = 4;
a[j-1] = a[j], d.h. a[0] = a[1] = 3; Zwischenstand – 3, 3, 2, 1
a[j] = temp, d.h. a[1] = temp = 4; Zwischenstand – 3, **4**, 2, 1
j = 2 (geht bis 3)
a[j-1] > a[j] ? d.h. a[1] > a[2] ? d.h. 4 > 2 ? ja, also
temp = a[j-1] = a[1] = 4;
a[j-1] = a[j], d.h. a[1] = a[2] = 2; Zwischenstand – 3, 2, 2, 1
a[j] = temp, d.h. a[2] = temp = 4; Zwischenstand – 3, 2, **4**, 1
j = 3 (geht bis 3)
a[j-1] > a[j] ? d.h. a[2] > a[3] ? d.h. 4 > 1 ? ja, also
temp = a[j-1] = a[2] = 4;
a[j-1] = a[j], d.h. a[2] = a[3] = 1; Zwischenstand – 3, 2, 1, 1
a[j] = temp, d.h. a[3] = temp = 4; Zwischenstand – 3, 2, 1, **4**
1. Durchlauf vorbei - größte Zahl ist ganz hinten!
i = 1 (geht bis 2)
… analog zu obigem Durchlauf
Zwischenstände: i = 1: 2, 1, **3, 4** **zweitgrößte Zahl ist an der richtigen Stelle!**
i = 2: 1, **2, 3, 4** **drittgrößte Zahl ist richtig - damit alles sortiert!**

Operationen

1. Durchlauf	2. Durchlauf	3. Durchlauf
Vergleiche: 3	Vergleiche: 3	Vergleiche: 3
Vertauschungen: 3	Vertauschungen: 2	Vertauschungen: 1

insgesamt: Vergleiche: 3 * 3 = 9, Vertauschungen: 3 + 2 + 1 = 6

Operationen mit beliebig vielen Elementen n im worst case

1. Durchlauf	2. Durchlauf	3. Durchlauf	…
Vergleiche: n-1	Vergleiche: n-1	Vergleiche: n-1	
Vertauschungen: n-1	Vertauschungen: n-2	Vertauschungen: n-3	

„quadratischer Aufwand“

insgesamt (n-1 Durchläufe):
Vergleiche: $(n-1)*(n-1) = n^2 - 2n + 1$, **also circa n^2**
Vertauschungen: $(n-1) + (n-2) + \ldots + 1 = \frac{(n-1)*n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$, **also circa n^2**

Selectionsort (Minsort)
am Beispiel „4, 3, 2, 1“

i = 0 (geht bis 2)
minpos = minimumposPosition(0)
minpos = from = 0
i = from + 1 = 0 + 1 = 1 (geht bis 3 - beachte: anders i als oben!)
a[i] < a[minpos] ? d.h. a[1] < a[0] ? d.h. 3 < 4 ? ja, also minpos = i = 1
i = 2
a[i] < a[minpos] ? d.h. a[2] < a[1] ? d.h. 2 < 3 ? ja, also minpos = i = 2
i = 3
a[i] < a[minpos] ? d.h. a[3] < a[2] ? d.h. 1 < 2 ? ja, also minpos = i = 3
return minpos = 3
minpos = 3
swap(minpos, i) = swap(3, 0) - beachte: i von „selectionSort()“
temp = a[i] = a[3] = 1
a[i] = a[j], d.h. a[3] = a[0] = 4
a[j] = temp, d.h. a[0] = temp = 1
Zwischenstand – **1**, 3, 2, 4
1. Durchlauf vorbei - kleinste Zahl ist ganz vorne!
i = 1 (geht bis 2)
… analog zu obigem Durchlauf

Zwischenstände:	i = 1:	**1, 2**, 3, 4	**zweitkleinste Zahl ist an der richtigen Stelle!**
	i = 2:	**1, 2, 3**, 4	**drittkleinste Zahl ist richtig - damit alles sortiert!**

Operationen

1. Durchlauf	2. Durchlauf	3. Durchlauf
Vergleiche: 3	Vergleiche: 2	Vergleiche: 1
Vertauschungen: 1	Vertauschungen: 1	Vertauschungen: 1

insgesamt: Vergleiche: 3 + 2 + 1 = 6, Vertauschungen: 3 * 1 = 3

Operationen mit beliebig vielen Elementen n im worst case

1. Durchlauf	2. Durchlauf	3. Durchlauf	…
Vergleiche: n-1	Vergleiche: n-2	Vergleiche: n-3	
Vertauschungen: 1	Vertauschungen: 1	Vertauschungen: 1	

„quadratischer Aufwand“

insgesamt (n-1 Durchläufe):

Vergleiche: $(n-1)+(n-2)+\ldots+1 \;=\; \frac{(n-1)*n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n,$ **also circa n^2**

Vertauschungen: $(n-1)*1 = n-1,$ **also circa n**

Insertionsort soll hier nicht näher betrachtet werden -
es ergibt sich aber genauso insgesamt ein quadratischer Aufwand!