# Alp2 - Übungsblatt 3

Bearbeitet von: Jasmine Cavael & Alexander Chmielus
Tutor: Fabian Halama
Tutorium 10 (Do. 16-18)

Aufgabe 3c)

Insertsort bietet sich vor allem für kleine Listen, oder Listen, die bereits (beinahe) sortiert sind an. Bei bereits sortierten läge Insertsort in O(n) (best-Case), während Quicksort immernoch in O(n * log n) und damit langsamer wäre.
Der Vorteil bei kleinen Listen kommt zustande, weil Insertsort keine Rekursion verwendet, Quicksort aber schon, was zusätzlichen Speicherplatz und mehr Vergleiche benötigt.

Aufgabe 4b)
**Rot** markiert die Stelle an der **i** in der Partitionsfunktion gerade ist, **lila** zeigt **j** an. 5(e) ist die erste 5, 5(z) die zweite und 5(d) die dritte.

*Quicksort([5, 6, 6, 1, 5, 4], 0, 5):*

[5(e), 6, 5(z), 1, 5(d), 4]
[**5(e)**, **6**, 5(z), 1, 5(d), 4]
[**5(e)**, 6, **5(z)**, 1, 5(d), 4]
[**5(e)**, 6, 5(z), **1**, 5(d), 4]
[5(e), **6**, 5(z), **1**, 5(d), 4]
[5(e), **1**, 5(z), **6**, 5(d), 4]
[5(e), **1**, 5(z), 6, **5(d)**, 4]
[5(e), **1**, 5(z), 6, 5(d), **4**]
[5(e), 1, **5(z)**, 6, 5(d), **4**]
[5(e), 1, **4**, 6, 5(d), **5(z)**]
[4, 1, **5(e)**, 6, 5(d), **5(z)**]

Bereits hier sehen wir, dass 5(d) und 5(z) vertauscht wurden, wodurch der Algorithmus nicht mehr stabil ist.

Aufgabe 4c)
Ja, siehe Code.

Aufgabe 4d)

Gehen wir von einer Liste (seq) mit n Elementen aus. Die Funktion würde in jedem Aufruf 3 neue Listen erstellen: Eine mit kleineren Elementen, eine mit größeren und eine mit dem Pivot. Erst, wenn die Listen nur noch aus einem Element bestehen, wird abgebrochen, d.h. die Funktion hat am Ende für jedes Element eine eigene Liste, die nur dieses enthält erstellt.
Also haben wir am Ende genau n Listen, die dann zu einer zusammengefügt werden - Speicherverbrauch = O(n)

*Aufgabe 4e)*

Zunächst erstmal: Es hängt von der Implementierung ab (ist zB. das Pivotelement immer das kleinste wird es in jedem Fall max. nur 1x bewegt). Aber es ist wahrscheinlich der Algorithmus aus der VL gemeint, daher beschränken wir uns darauf.

Gehen wir von einer Liste mit n Elementen aus, bei der alle Zahlen absteigend sortiert sind. Dadurch ist das kleinste Element am Anfang an Index (n-1) Nach dem ersten Durchlauf befindet sich das kleinste Element an Index (n-2). Beim zweiten Durchlauf an Index (n-3) usw.. bis es an Index 0 angekommen ist. Das ganze benötigt (n-1) Vertauschungen. Mehr ist auch nicht möglich, da es nach dem Algorithmus nie in die "falsche" Richtung getauscht wird.

Das kleinste Element kann also maximal (n-1) mal bewegt werden.