

Bubble-Sort

Diese Sortiermethode ist so ziemlich **die schlechteste der Welt**. Trotzdem ist sie in fast jedem Lehrbuch zu finden (Rühmliche Ausnahmen: Preuss (hier treten die Autoren sogar für ein strenges Verbot dieser Methode ein) und Knuth (er verwendet sie als abschreckendes Beispiel)). Interessanterweise kommen Schüler von selbst NICHT auf diese Technik, sondern eher auf Selection und Insertion. Also dürften tatsächlich wir Lehrer mit Schuld an der Verbreitung dieser Methode tragen.

Ich will den Algorithmus aber aus folgenden Gründen erwähnen:

- Es ist lehrreich zu erkennen, WARUM dieser Algorithmus so schlecht ist
- Man kann mit den Schülern den Versuch unternehmen, BubbleSort zu verbessern. Es gibt einige schlaue Ideen, die man verwirklichen kann. Lehrziel 1: wenn die Grundidee schlecht ist, helfen auch Detailverbesserungen nichts.
- BubbleSort lässt sich leicht 'auswendiglernen' und programmieren. Unerfahrene Programmierer denken oft: 'ich arbeite mit Assembler/C/Delphi/VB-Compiler. Das ist so schnell, da kann ich ja ruhig BubbleSort nehmen. Das ist in 3 Zeilen programmiert (in Assembler etwas mehr)'. Lernziel 2: eine 'langsame' Programmiersprache, die die Umsetzung von abstrakten Algorithmen erleichtert und unterstützt, schlägt schlechten Assemblercode mit Leichtigkeit. Beispiel: 10000 Zahlen Assembler-Bubble gegen Python-Quicksort.
- Die quadratische Laufzeit ist sofort erkennbar. Auch die Zahl der Vergleiche und Tauschvorgänge lässt sich einfach abschätzen. Ein guter Ausgangspunkt, die Unterschiede zu den anderen Algorithmen auf den Punkt zu bringen.
- Spezialthema für interessierte Schüler: finde Algorithmen, die noch schlechter sind.
Zwei meiner Lieblingsbeispiele:
 - 1.) Bilde alle Permutationen der Ausgangsliste. Dann suche die richtig geordnete Liste heraus. (Maximale Speicherplatzverschwendung)
 - 2.) while not liste_ist_sortiert: vertausche zwei beliebige Elemente. (Maximales Glücksspiel)

Das alles spricht so stark für den Bubble-Sort, dass man ihn gleich anfangs unterrichten sollte. Wenn Schüler von selbst drauf kommen, ist es durchaus legitim. Aber es besteht die Gefahr, dass der Schüler später einmal diesen Algorithmus tatsächlich anwendet und der Name des Informatiklehrers im Zusammenhang erwähnt wird. Das wäre wohl etwas peinlich....

Nun aber zum Algorithmus, jedoch mit der in der Musikindustrie so beliebten **parental advisory: This section contains very bad code. Don't try this at home.**

Algorithmus:

Man schaut die Zahlenfolge von Anfang bis Ende durch und vergleicht jeweils zwei Nachbarn. Stehen sie falsch herum, werden sie vertauscht. Wie oft muss man das tun: schlimmstenfalls so oft, wie es Elemente in der Folge gibt, da ein großes Element vorne einen ganzen Durchlauf braucht, um nach oben zu kommen (große Elemente 'blubbern' wie Luftblasen im Aquarium hinauf).

Worst Case: die gestürzte Liste.

Zeitverhalten: n^2 .

<pre>def bubble(L): for x in range(len(L)): for z in fullrange(0, len(L)-2): if L[z]>L[z+1]: swap(L, z, z+1)</pre>	<ul style="list-style-type: none"> - vom ersten bis zum vorletzten Element - dieses Element und sein rechter Nachbar
---	--

Bei den folgenden Ideen unbedingt jedesmal den Zeitbedarf/Zeitgewinn messen!

Verbesserung 1:

Nach dem ersten Durchgang ist das oberste Element sicher in Ordnung. Nach dem zweiten das vorletzte usw. Wir führen eine obere Grenze mit, die uns sagt, wie weit hinauf wir arbeiten müssen. Wir müssen nur bis zum zweiten Element heruntergehen, das letzte ist sicher richtig.

<pre>def bubble1(L): for oben in fullrange(len(L)-1,1,-1): for z in fullrange(0,oben-1): if L[z]>L[z+1]: swap(L,z,z+1)</pre>	- vom obersten Element nach vor
---	---------------------------------

Verbesserung 2:

Kann es nicht sein, dass die Liste schon vor dem Ende des Algorithmus fertig sortiert wäre? Ja sicher, etwa wäre [1,9,2,3,4] nach einem einzigen Durchlauf schon in Ordnung. Wir machen also keine for Schleife, sondern arbeiten so lange, bis kein Tausch mehr nötig war.

Verbesserung 3 – Shaker Sort:

Warum dauert die Sortierung von [2,3,4,5,1] lange, und die von [5,1,2,3,4] ist gleich erledigt? Weil die Elemente nur nach oben blubbern können. Ergänzen wir Verbesserung 2 doch dadurch, dass wir abwechselnd hinauf und hinunter blubbern. Die entstandene Sortiermethode heisst übrigens aus offensichtlichen Gründen **Shaker-Sort**.

<pre>TRUE = 1 FALSE = 0 def shaker(L): erstes = 0 letztes = len(L)-1 while TRUE: fertig = TRUE for i in fullrange(erstes,letztes-1): if L[i]>L[i+1]: swap(L,i,i+1) fertig = FALSE letztes = letztes-1 if fertig: return fertig = TRUE for i in fullrange(letztes,erstes+1,-1): if L[i-1]>L[i]: swap(L,i-1,i) fertig = FALSE erstes = erstes+1 if fertig: return</pre>	<p>ShakerSort</p> <ul style="list-style-type: none"> - untere Grenze - obere Grenze - nehmen wir mal an... - es muss getauscht werden - also sind wir noch nicht fertig - oberstes Element korrekt
--	--

Verbesserung 4:

Dies war ein Schülervorschlag: Wir zählen oben die Grenzen immer um 1 hinunter bzw. hinauf. Sollten wir uns nicht vielleicht merken, WO der letzte Tauschvorgang nötig war? Es ist erstaunlich, dass diese gute Idee

eigentlich wenig Wirkung zeigt (nur am Ende, doch da ist die zu bearbeitende Teilliste bereits klein). Unten folgt die Version des Schülers. Statt der 2 Werte TRUE und FALSE verwendet er die Variable *fertig*, um den **Ort** des letzten Tausches zu fixieren. Bleibt die Zahl -1 stehen, war kein Tausch mehr nötig.

```
def shakerbest(feld,erstes,letztes):
    while TRUE:
        fertig = -1
        for i in fullrange(erstes,letztes-1):
            if feld[i]>feld[i+1]:
                swap(feld,i,i+1)
                fertig = i
        letztes = fertig
        if fertig==-1: return

        fertig = -1
        for i in fullrange(letztes,erstes+1,-1):
            if feld[i-1]>feld[i]:
                swap(feld,i-1,i)
                fertig = i
        erstes = fertig
        if fertig==-1: return
```

Lernziel 4:

Dieses Programm sieht schon recht umfangreich aus. Viele Zeilen, viele Überlegungen, viele Variable. Leider ist es im Vergleich zu allen anderen Verfahren, die wir gelernt haben, immer noch 'schlecht'. Denn die Grundidee, immer nur direkte Nachbarn zu tauschen, ist einfach nicht gut.

P.S. Knuth (5.3.4. Aufgabe 68) verblüfft uns mit der Konstruktion einer Sortiermaschine, deren optimaler Algorithmus Bubblesort ist!