

Insertion Sort

Wie sortiert ein Kartenspieler seine Karten?

Anfangs hat er einen unsortierten Kartenhaufen vor sich liegen.

1.) wenn er noch keine Karte in der Hand hat, nimmt er die erste ohne Nachzudenken auf

2.) sonst nimmt er eine Karte auf und hält sie vor die erste Karte in seiner Hand.

2a) Passt sie hierher, wird sie hier aufgenommen. Falls nicht, lässt man die erste Karte in Ruhe und versucht das selbe Spielchen

2b) ab der nächsten Karte in der Hand.

2c) Falls keine nächste Karte mehr in der Hand ist, wird die neue hier am Ende hinter die letzte gesteckt (sie ist dann offenbar größer als alle bisherigen).

Diese Ideen sind leicht in einen Algorithmus zu übersetzen.

Wir teilen die Aufgabe in zwei Teile: der erste **insert** sortiert eine Karte in eine bereits vorsortierte Liste, der zweite **sort** sortiert eine ganze Liste. Jede Funktion liefert ihr Ergebnis zurück.

<pre>def insert(x,Liste): if Liste==[]: return [x] if x<=Liste[0]: return [x]+Liste return[Liste[0]]+insert(x,Liste[1:])</pre>	<ul style="list-style-type: none"> - Element x in sortierte Liste L einfügen - Liste war leer: x ist als einziges Element aufnehmen - x passt ganz vorne dran, also einfügen - sonst ist das Ergebnis das erste Listenelement und das Element x wird in die Restliste eingefügt
<pre>def sort(L): if L==[]: return [] return insert(L[0],sort(L[1:]))</pre>	<ul style="list-style-type: none"> - sortiere die Liste L - leere Liste bleibt leere Liste - sortiere das erste Element in den bereits sortierten Rest der Liste ein

Teste die Funktionen zumindest mit:

```
sort([1,9,-2000])
```

```
sort(['X','PYTHON','JAUSE','EDV'])
```

Forschen und denken:

- wie reagiert insert, wenn die übergebene Teil-Liste nicht sortiert ist. Etwa insert(3,[1,5,2])
- was passiert bei gemischten Typen wie sort(['eins',4,'three',2])
- was passiert bei Worten mit gemischter Groß- und Kleinschreibung
- wie werden Umlaute sortiert
- was passiert bei großen Listen, etwa sort(range(1000)+range(1000))

Aufgabe:

Sind die Worte „Abend“, „Aera“, „Ärmel“, „âne“, „Ångstrøm“, „Azteke“ richtig sortiert?

Befrage zum Thema Umlaute/scharfes S/groß-klein-Schreibung/ausländische Sonderzeichen a) den

Computer b) Deinen Deutschlehrer c) Euren Bibliothekar d) Wörterbücher und Lexika. Ziehe UNBEDINGT auch e) das Telefonbuch zu Rate! (âne ist französisch und heißt Esel)

Wir stellen also fest, dass

- 1.) **sort** gut sortieren kann
- 2.) es bei großen Listen langsam ist
- 3.) die Sortierung von Texten ein heikles Thema ist

Zum Thema Geschwindigkeit:

unsere obige rekursive Funktion erzeugt ständig neue Teillisten, die im Computer verwaltet werden müssen. Das ist sicher nicht effizient. Versuchen wir also mit einer einzigen Liste namens *liste* auszukommen. Diese ist anfangs unsortiert.

Wir arbeiten nun so:

- wir denken jetzt von rechts nach links: der Kartenspieler hält die neue Karte rechts an die bereits in seiner Hand befindlichen. Es ist technisch einfacher, die Liste am oberen Ende zu erweitern, da sie links fix mit Index 0 beginnt. Der Spieler vergleicht nun die Werte nach links hinunter und fügt die Karte ein, wenn sie passt.
- das erste Element nehmen wir so wie es ist, sortieren also erst ab dem zweiten Element wirklich ein.
- um `liste[n]` einzureihen, gehen wir davon aus, dass `liste[0]` bis `liste[n-1]`, die bisher aufgenommene Kartenfolge, bereits sortiert ist. Wir merken uns `liste[n]` in einer Hilfsvariable *wert*, womit wir diesen Platz freibekommen, und schieben ständig Elemente von links nach rechts (wobei wir immer wieder einen freien Platz erhalten) bis das gemerkte Element hineinpasst, da es größergleich dem nun linken Nachbarn ist. Dieser 'freie Platz' ist nötig, da ja beim Einsetzen des neuen Elements der sortierte Bereich um eine Stelle größer wird.

Wenn wir an den Kartenspieler denken: wir nehmen eine neue Karte auf und versuchen, ob sie ganz rechts dranpasst. Wenn nicht, schieben wir die rechte Karte in der Hand ein Stück weiter nach rechts und erhalten einen freien Platz. Passt die neue Karte hier hin, stecken wir sie einfach herein. Wenn nicht, schieben wir die Karte vor der freien Stelle in der Hand eine Position nach rechts und erhalten wieder einen neuen freien Platz und verfahren genauso weiter. Falls wir an der ersten Position angelangt sind, kommt die neue Karte sicher hier her, es kann ja keine kleinere mehr geben.

2 2 4 7 9 6	.	Wir wollen den Sechser einsortieren
2 2 4 7 9 .	6	Wir nehmen ihn heraus. Ist er ≥ 9 ? Nein, Neuer raufsetzen
2 2 4 7 . 9	6	6 ist nicht ≥ 7 . Also Siebener rauf .
2 2 4 . 7 9	6	6 ist ≥ 4 , also Sechser einfügen
2 2 4 6 7 9	.	Fertig.

<pre>def sort(liste): for stelle in range(1, len(liste)): wert = liste[stelle] i = stelle while 1: if i==0: break if liste[i-1]<=wert: break liste[i] = liste[i-1] i = i-1 liste[i] = wert</pre>	<pre># vom zweiten Element bis zum letzten # das Element ist einzureihen. # von hier abwärts suchen wir den richtigen Platz # ganz unten angelangt # links steht was kleineres – ok! # sonst linkes Element raufschieben # eine Stelle weiter links fortfahren # einreihen</pre>
---	---

Aufgabe:

- 1.) Wie lange dauert es, um 500, 1000 2000 oder 4000 Werte zu sortieren?
- 2.) Eine doppelt so lange Liste benötigt so viel Zeit
- 3.) Das Zeitverhalten von Insertion-Sort ist also quadratisch.

Ausblick:

befindet sich ein kleiner Wert sehr weit rechts, so braucht es viele einzelne Verschiebungen, um ihn nach vorne zu bringen. Ein Trick könnte darin bestehen, nicht nur den direkten linken Nachbarn zu vergleichen, sondern etwa den viert- oder siebt-nächsten und dadurch kleine Werte schnell nach vor zu bekommen. Diese Idee ist tatsächlich so gut,

dass sie weltweit Anwendung findet. Diese Sortiermethode heißt 'Shell-Sort'. Du wirst sie etwas später genauer kennenlernen.

Forschung:

ersetzen wir in unserer Funktion die Zeile `if liste[i-1]<=wert: break` durch die Anweisung `if vorher(liste[i-1],wert): break`, so können wir eine eigene Funktion `vorher(x,y)` erstellen, die auswertet, ob `x` in der Reihenfolge vor `y` kommt. Diese Funktion kann dann mit speziellen Abfragen ausgerüstet werden, die auch Umlaute, Apostrophe, usw. richtig bearbeitet.