

Merge-Sort

Wir haben bereits gesehen, dass längere Listen viel mehr Sortierzeit beanspruchen. Mit einer Methode wie Selection-Sort brauchen doppelt so viele Elemente 4 mal so viel Zeit. Wenn wir allerdings die untere Hälfte und die obere getrennt sortieren, so braucht das nur doppelt so viel Zeit.

Dann benötigen wir noch etwas Zeit, um die beiden Listen richtig zusammenzumischen. Doch müssen wir dabei immer nur das erste Element jedes Teils anschauen: das kleinere von beiden wird in die Ergebnisliste aufgenommen. Der Vorgang wiederholt sich, bis eine der Teillisten leer ist, dann kann die andere einfach hinten angesetzt werden.

Nach dem Prinzip des divide and conquer:

Wir sollen 16 Zahlen sortieren? Nein. Wir arbeiten zweimal mit 8 Zahlen. Oder noch besser vier mal mit vier. Oder 8 mal mit 2. Oder 16 mal ein Element. Denn eine einzige Zahl zu sortieren ist wirklich nicht schwer...

Danach mischen wir die Einzelzahlen zu Zweierlisten, diese zu Viererlisten usw. bis wir fertig sind.

```
x x x x x x x x  statt 8 Elementen
x x x x x x x x  sortiere 2 mal 4
x x x x x x x x  sortiere 4 mal 2
x x x x x x x x  8 mal eines
```

So einfach die Idee ist, so sehr gibt es Detailprobleme. Das Zusammenmischen ist doch nicht so leicht: Wir müssen eine neue Liste für das Ergebnis bilden (Die Programmierung ohne eine solche Hilfsliste ist SEHR mühsam...).

Weiters müssen wir beim Zusammenmischen immer kontrollieren, ob eine der Listen bereits leer ist. Abhilfe wäre: wir stellen ans Ende jeder Liste ein 'riesiges' Element, das sicher erst ganz am Ende drankommt und verwenden das als Signal. Besser ist folgender Trick: wir stellen die zwei Teillisten nebeneinander, drehen aber die rechte um. Jetzt ist das größte Element jeder Teilfolge gleichzeitig als Ende-Kennzeichen für die andere zu gebrauchen.

```
a f g p b f r t  ineinander mischen
i          j      Zeiger i und j wandern nach oben
                i++,j++,i++,j++,i++,i++, ACHTUNG links ist leer
```

```
a f g p b f r t  ineinander mischen
a f g p t r f b  rechts umdrehen
i                j  Zeiger i wandert rauf, Zeiger j hinunter, bis sie sich treffen
```

Wir verwenden folgenden zusätzlichen Trick: da wir ja wissen, wie viele Elemente zu bearbeiten sind, nehmen wir diesen Wert als Schleifenindex und passen i und j jeweils an.

<pre>def merge(L): mergel(L,L[:],0,len(L)-1)</pre>	<p>Mergesort: Liste, gleichgroße Hilfsliste und Indexgrenzen links rechts</p>
<pre>def mergel(L,H,links,rechts): if links<rechts: mitte = (rechts+links)/2 mergel(L,H,links,mitte) mergel(L,H,mitte+1,rechts) for i in fullrange(mitte,0,-1): H[i]=L[i] for i in fullrange(mitte+1,rechts): H[(mitte+1)+rechts-i]=L[i] l = links r = rechts for i in fullrange(links,rechts): if H[l]<H[r]: L[i]=H[l] l = l+1 else: L[i]=H[r] r = r-1</pre>	<p>L=Liste, H=Hilfsliste - wenn mehr als ein Element - in Hälften teilen - separat bearbeiten</p> <p>- linke Liste ins Hilfsfeld - rechte Teilliste gestürzt - Zeiger auf Listenbeginne</p> <p>- so oft wie Elemente zu bearbeiten - richtiges nehmen und Zeiger anpassen</p>

Theorie:

Mergesort ist ein schneller Algorithmus der Ordnung $n \log(n)$. Sein einziger Nachteil ist die Notwendigkeit eines zusätzlichen Hilfsfeldes gleicher Größe. Wenn die zu sortierende Liste aber nicht Millionen von Elementen hat, ist das kein wirkliches Problem.

(Bemerkung: Textlisten sind oft groß. Sie werden aber über Zeigerlisten sortiert, die wiederum klein sind)