Physik am Komputer 8. Mai 2010

# Übung 02

*Michael Kopp*

**Aufgabe 1** Siehe `fread03.cpp`.

**Aufgabe 2**

- Beim initialisieren verwende ich 0 als Startwert – damit sieht man, wenn am Schluss wieder 0 in der Variablen steht, dass etwas nicht geklappt hat.
- Schleife die Maximalwerte ermittelt in `fread04.cpp`.
- Schleife die größere Zahl nach unten tauscht in `fread05.cpp`
- Vollständiger Bubble-Sort: `fread06.cpp`. Wichtig: Hier kann eine Ausgabedatei als drittes Argument angegeben werden. Ohne das wird in `fread06_ausgabe` ausgegeben.
- `sort data > foo` ausgeführt; `diff foo fread06_ausgabe` zeigt *nichts* an – also keine Unterschiede.[1]
- Für die erste eingelesene Date (insg. $n$) werden $n$ Durchläufe gemacht, für die zweite $n-1$, für die $i$-te $n-i+1$. Für $n$ Zahlen braucht man also

$$\sum_{i=1}^{n} n - i + 1 = -\frac{n^2+n}{2} + n^2 + n \tag{1}$$

  Durchläufe. Der Algorithmus skaliert damit mit $O(n^2)$.

[1] Um das zu testen habe ich das Programm noch ohne Parameter laufen lassen woraufhin 250 Messwerte gezählt wurden – dann hat diff gemeckert... Bei dieser Gelegenheit musste ich auch das Error-handeling neu schreiben; bei meinem gcc scheint atoi ein NULL zurückzugeben, wenn keine Zahl eingegeben wird.

**Aufgabe 3** Der Quicksort findet sich in `quick01.cpp`; hier wurde auf Kontrollelemente verzichtet: Die Dateien müssen manuell im Quellcode angegeben werden.

Das besondere ist, dass dieses Programm jeden Schritt ausgibt; diese kann man verwenden, um die Arbeitsweise zu visualisieren (vgl. Kommentare am Programmanfang). Dazu wurde die Datei `anim.flv` erzeugt.

Der Quicksort braucht bei `data` für *alle* Daten ca. 9 sekunden (das kann man ermitteln indem man `date && ./a.out > foo && date` ablaufen lässt und die Sekunden bei `date` vergleicht. Der Bubble-Sort hat dafür wesentlich länger gebraucht.

Der Grund liegt darin, dass quicksort im Idealfall mit $O(n \log n)$ arbeitet.