Thema, Ziele: Sortieralgorithmen und Komplexitätstheorie

Aufgabe 1: Zeitmessung bei Fibonacci-Implementationen

Die iterative Version mit nativer Linux-Zeitmessung finden Sie unter
`./Loesung/FiboItWatch/fiboit.cpp`

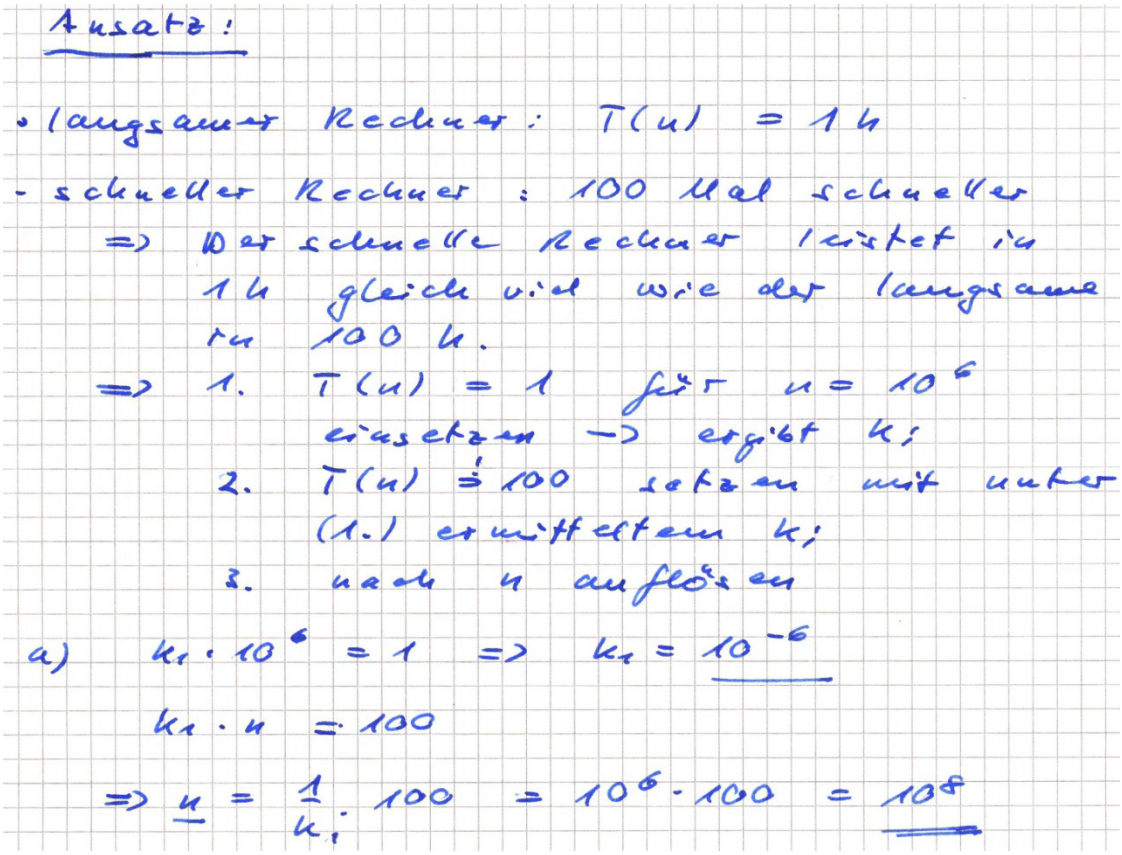
Das Eclipse-Projekt der rekursiven Version mit `clock()`-Zeitmessung finden Sie unter
`./Loesung/FiboRekWatch`

Das Eclipse-Projekt der rekursiven Version mit nativer Linux-Zeitmessung finden Sie unter
`./Loesung/FiboRekLinux`

Aufgabe 2: Klasse für Stoppuhr

Das Eclipse-Projekt finden Sie unter `./Loesung/StopWatch`

Aufgabe 3: Komplexität



b) $k_2 \cdot 10^6 \cdot \log_{10} 10^6 = 1$

$\Rightarrow k_2 \cdot 10^6 \cdot 6 = 1 \Rightarrow k_2 = \frac{1}{6} \cdot 10^{-6}$

$k_2 \cdot n \cdot \log_{10} n = 100$

$\Rightarrow n \cdot \log_{10} n = 6 \cdot 10^8 =: c_2$

$\Rightarrow 10^{n \cdot \log_{10} n} = 10^{c_2}$

$\Rightarrow n^n = 10^{c_2} = 10^{6 \cdot 10^8}$

Diese Gleichung lässt sich nicht geschlossen lösen.

Solver: $n = 76'127'253$

c) $k_3 \cdot (10^6)^2 = 1$

$\Rightarrow k_3 \cdot 10^{12} = 1 \Rightarrow k_3 = 10^{-12}$

$k_3 \cdot n^2 = 100$

$\Rightarrow n = \sqrt{10^{14}} = 10^7$

d) $k_4 \cdot (10^6)^3 = 1$

$\Rightarrow k_4 \cdot 10^{18} = 1 \Rightarrow k_4 = 10^{-18}$

$k_4 \cdot n^3 = 100 \Rightarrow n^3 = 10^{20}$

$\Rightarrow n = 10^{20/3} = 4'641'588$

e) $k_5 \cdot 10^{(10^6)} = 1 \Rightarrow k_5 = \frac{1}{10^{(10^6)}}$

$k_5 \cdot 10^n = 100$

$\Rightarrow 10^n = 100 \cdot 10^{(10^6)} = 10^{(10^6+2)}$

$\Rightarrow n = 10^6 + 2 = 1'000'002$

d.h. mit einem 100 mal schnelleren Rechner können sie 2 (!) zusätzliche Datensätze berechnen.

Aufgabe 4: Sortieralgorithmen

Das Eclipse-Projekt finden Sie unter `./Loesung/QuicksortFast`

Mit dem verwendeten Rechner wurde die Listengrösse mit 5 Mio. Elementen des Typs `int` gewählt. Die Liste musste auf dem Heap angelegt werden, beim Einsatz einer Stackvariablen resultierte ein Stackoverflow.

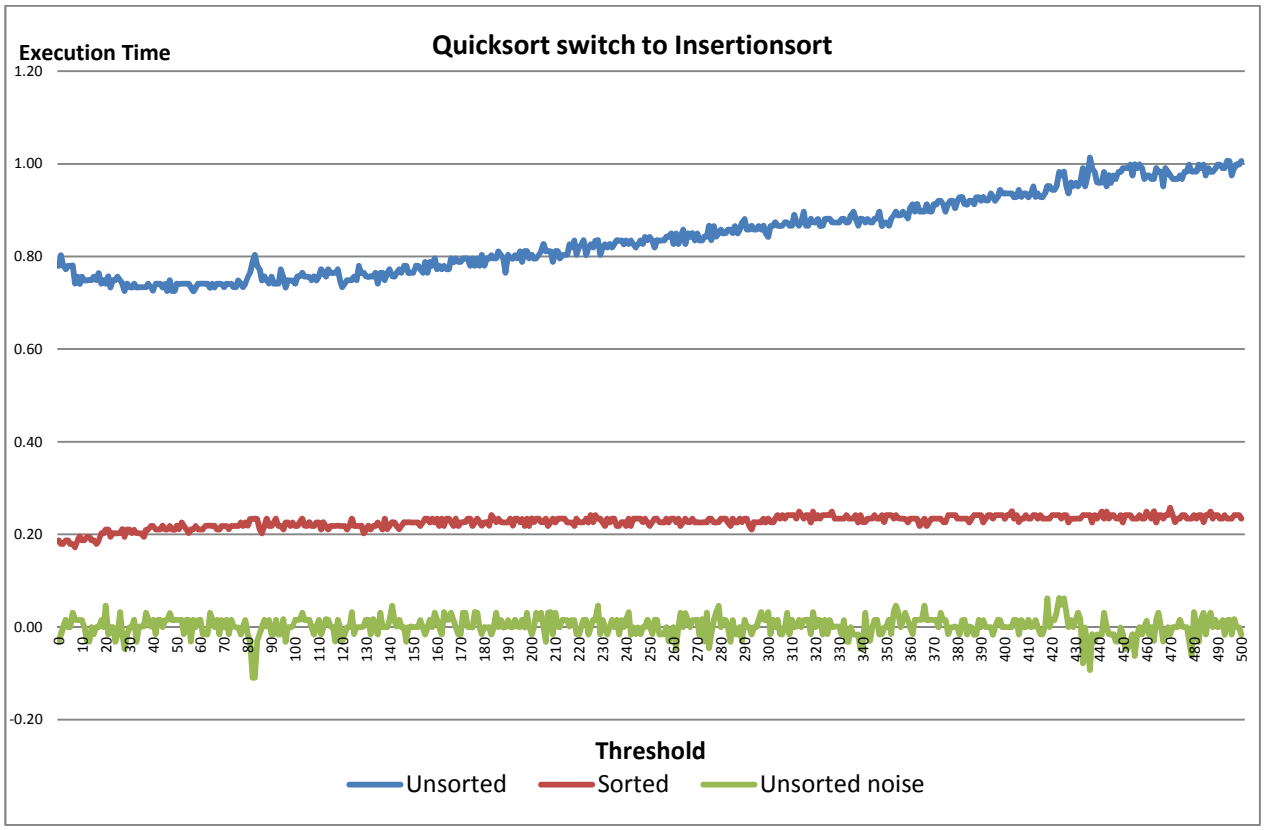


Abbildung 1: Laufzeitmessungen beim implementierten Algorithmus

In Abbildung 1 sind die Laufzeitmessungen dargestellt (Execution time in Sekunden). Wenn die Schwelle auf Null gesetzt wird, gibt es keinen Wechsel des Algorithmus, d.h. das ist der reine Quicksort-Algorithmus. Aus den Messungen ist ersichtlich, dass das Sortieren der unsortierten Liste am schnellsten ist, wenn unterhalb einer Listengrösse von ca. 30-80 auf den Insertionsort umgestellt wird (obere Kurve 'Unsorted'). Wenn die Liste bereits sortiert ist, dann schneidet der reine Quicksort nicht schlechter ab (mittlere Kurve 'Sorted'). Aufgrund der Kombination dieser *beider* Kurven ist es sinnvoll, die Schwelle auf ca. 30 zu setzen.
Die unterste Kurve stellt das Messrauschen, d.h. die Abweichung bei mehreren Messungen, dar bei der Sortierung von unsortierten Listen. Das Messrauschen ist doch recht hoch.