Algorithmische Mathematik I

Wintersemester 19/20
Prof. Dr. J. Gedicke
Johannes Rentrop und Jannik Schürg

UNIVERSITÄT BONN

Programmieraufgabenblatt 5.

Abgabedatum: **18.11.2019–22.11.2019**

Programmieraufgabe 1. (Sortieren)

a) Implementieren Sie Quicksort und Mergesort jeweils als C-Funktion. Die Funktionen sollen jeweils als Argument einen Pointer vom Typ `unsigned char*` auf das zu sortierende Array erhalten, sowie dessen Länge.

b) Erzeugen Sie ein Array vom Typ `usigned char` mit 1,000,000 pseudeozufälligen[1] Zahlen aus $\{0, 1, \ldots, 99\}$. Sortieren Sie das Array separat mit Ihrer Implementierung von Quicksort bzw. Mergesort und der Funktion `qsort` der C-Standardbibliothek und prüfen Sie durch Vergleich der Ergebnisse, ob ihre Implementierung korrekt ist.

c) Schreiben Sie eine Funktion, die zählt, wie viele Zahlen kleiner als 50 in einem Array vom Typ `unsigned char` vorkommen, und diese Anzahl zurückgibt.

Vergleichen Sie die Laufzeit dieser Funktion für das Array aus b), indem Sie einmal das unsortierte Array verwenden und einmal das sortierte. Nutzen Sie beim Kompilieren einmal `-Os` und einmal `-O3`. Überlegen Sie sich im Vorfeld welches Laufzeitverhalten Sie erwarten, d.h. welchen Unterschied sollte die Sortierung machen?

Für den Laufzeitvergleich kann der Code-Schnipsel unten auf dem Blatt verwendet werden.

(5+3+2 Punkte)

Programmieraufgabe 2. (Komplexe Zahlen und Fließkommaarithmetik)

Sei $c \in \mathbb{C}$ eine komplexe Zahl. Definiere die Folge $z_n(c)$ rekursiv durch

$$z_0(c) := 0,$$
$$z_{n+1}(c) := z_n(c)^2 + c.$$

Sei $M \subseteq \mathbb{C}$ die Menge der c, so dass $z_n(c)$ konvergiert.
Wir möchten in dieser Aufgabe die Menge M in einem Teilstück $I \subset \mathbb{C}$ visualisieren. Wir wählen für I ein Rechteck

$$I := [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \subset \mathbb{C}$$

und färben einen Punkt $c \in I$ in einem Bild umso heller ein, je langsamer $z_n(c)$ divergiert. Punkte c, die in M sind, werden also eher hell, die anderen dunkel.
Für die Implementierung möchten wir einige Approximationen/Heuristiken verwenden:

1. Wir „prüfen“ auf Konvergenz, indem nachgerechnet wird, ob $\|z_n(c)\| < 2$ für $n = 0, \ldots, n_{\max}$.

[1] Eine schnelle Recherche zeigt, wie Sie das erreichen können in C.

2. Das Rechteck I wird diskretisiert in $w \times h$ Pixel, d.h. in gleichmäßige Rechtecke zerlegt, die den Bildpunkten entsprechen.

Ein Überblick liefert Algorithmus 1 auf diesem Blatt. Erzeugen Sie die Visualisierung für $x_{\min} = -1.9$, $x_{\max} = 0.6$, $y_{\min} = -1.1$, $y_{\max} = 1.1$, $n_{\max} = 1024$, $w = 2500$, $h = 2200$ mit Hilfe einer C-Implementierung. Es empfielt sich, das Ergebnis als PGM-Bilddatei[2] zu speichern.
Sie können die Laufzeit verringern, indem Sie $n_{\max}$, w oder h verringern. Nutzen Sie `complex.h` und `float complex` in ihrem C-Programm.
Der Exponent 0.2 in Algorithmus 1 dient dazu, Farbwerte im unteren Bereich besser sichtbar zu machen (man spricht von einer Gamma-Korrektur), versuchen Sie auch andere Werte in $(0, 1]$. Das Array I enthält am Ende Graustufenwerte im Bereich $0, \ldots, 255$. Sie sollten diese auf ganze Zahlen runden.

(10 Punkte)

Laufzeitvergleich Falls Sie den Compiler GCC verwenden[3], können Sie folgenden Code-Schnipsel oben in ihrer Datei einfügen (mit „\“):

```
#include <time.h>
#define MEASURE(code, out) ({\
        clock();                                          \
        (code);                                           \
        asm volatile("" : : "m"(out) : "memory");         \
        clock_t begin = clock();                          \
        for (int i = 0; i < 500; ++i) {                   \
            (code);                                       \
            asm volatile("" : : "m"(out) : "memory");     \
        }                                                 \
        double elapsed = (double)(                        \
            clock() - begin) / CLOCKS_PER_SEC;            \
        elapsed;                                          \
    })
```

Um die Zeit in Sekunden auszugeben, können Sie dann wie folgt vorgehen:

```
int out;
printf("Laufzeit: %f\n", MEASURE(out = count_below50(arr, n), out));
```

Hier bezeichnet das erste Argument, „`out = count_below50(arr, n)`“ den Code, dessen Laufzeit Sie prüfen möchten. Die Variable `out` enthält das Ergebnis und muss angebenen werden, um zu verhindern, dass der Compiler den ganzen Code wegoptimiert, da er feststellt, dass die Berechnung keine Auswirkung hat.

[2] Dieses Bildformat ist sehr einfach, wie eine kurze Recherche zeigen wird.
[3] Sonst recherchieren Sie eine Alternative.

Algorithmus 1 Berechnet eine Visualisierung von M

function MVisualisierung($x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, w, h, $n_{\max}$)
    $I \leftarrow$ 2D-Array der Größe $w \times h$
    $\Delta x \leftarrow (x_{\max} - x_{\min})/w$ ▷ Breite eines Rechtecks
    $\Delta y \leftarrow (y_{\max} - y_{\min})/h$ ▷ Höhe eines Rechtecks
    for all $(k_x, k_y) \in \{1, \ldots, w\} \times \{1, \ldots, h\}$ **do**
        $x \leftarrow x_{\min} + (k_x - 0.5)\Delta x$ ▷ Rechne Index um
        $y \leftarrow y_{\min} + (k_y - 0.5)\Delta y$
        $c \leftarrow x + iy$
        $z \leftarrow 0$
        $n \leftarrow 0$
        while $\|z\| < 2$ **and** $n \leq n_{\max}$ **do**
            $z \leftarrow z^2 + c$
            $n \leftarrow n + 1$
        end while
        $I(k_x, k_y) \leftarrow \left(\frac{n}{n_{\max}}\right)^{0.2} \cdot 255$
    end for
    return I
end function