Fakultät IV Elektrotechnik und Informatik
Einführung in die Programmierung WS 2019/2020
Manfred Hauswirth / Damien Foucard / Christoph Seifert

Aufgabenblatt 3

Ausgabe: *22.11.2019*
Abgabe: *03.12.2019 09:59*

Thema: *Laufzeitbestimmung, Insertionsort, Countsort*

Abgabemodalitäten

1. Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des ZE-CM/eecsIT mittels `clang -std=c11 -Wall -g` kompilieren.
2. Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner. Nur wenn ein Test in Osiris angezeigt wird ist sichergestellt, dass die Abgabe erfolgt ist.
3. Du kannst bis zur Abgabefrist beliebig oft neue Versionen abgeben.
4. Die Abgabe erfolgt in folgendem Unterordner:
   `introprog-wise1920/Studierende/<L>/<TU-Login>/Abgaben/Blatt<XX>`
   wobei `<L>` durch den ersten Buchstabe des TU-Logins und `<XX>` durch die Nummer des Aufgabenblattes zu ersetzen sind. Die Ordner werden automatisch angelegt sobald die Abgabe freigeschaltet wird.
5. Du darfst den Abgabeordner für das Blatt nicht selbst erstellen. Die Ordner erstellen wir kurz nach der Ausgabe der Aufgabe.
6. Um die Änderungen (z. B. neue Abgabeordner) vom Server abzuholen, musst Du den Befehl `svn update` im Wurzelverzeichnis (also im Verzeichnis `introprog-wise1920`) des Repositories ausführen.
7. Im Abgabeordner gelten einige restriktive Regeln. Dort ist nur das Einchecken von Dateien mit den in der Aufgabe vorgegebenen Namen erlaubt, ausserdem werden die Abgabefristen vom Server überwacht. Beachte eventuelle Fehlermeldungen beim SVN-Commit. Lade nur Dateien hoch, die Du selbst bearbeiten sollst, insbesondere also keine Vorgaben.
8. Der Dateiname bei Abgaben mit C-Code soll dem der Vorgaben entsprechen. Entferne ausschließlich `_vorgabe` aus dem Name. Bei Textaufgaben sind, wenn nicht anders angegeben, `.txt` Dateien zugelassen, die als Plaintext-Datei zu speichern sind (keine Word-Dateien umbenennen).
9. Es gibt einen Ordner `Arbeitsverzeichnis`, in dem Du Dateien für Dich ablegen kannst.
10. Die Ergebnisse der automatischen Tests kannst Du auf OSIRIS einsehen:
    https://osiris.ods.tu-berlin.de/

Aufgabe 1 Laufzeitanalyse – Vergleich Count- und Insertionsort (10 Punkte)

Aufgabe 1.1 Vergleich Insertion- und Countsort (6,5 von 10 Punkten)

In dieser Aufgabe sollst Du (empirisch) die Laufzeit Deiner Insertion- und Countsort-Implementierungen vergleichen. Ähnlich zu Aufgabe 2 vom Blatt 2 erhälst Du eine Vorgabe, in welcher die komplette `main`-Funktion bereits vorgegeben ist. Du musst nur noch Deine Insertion- sowie Countsort Implementierungen (z. B. von Aufgabenblatt 1) einfügen und leicht anpassen. Du musst insgesamt vier verschiedene Funktionen schreiben bzw. anpassen:

1. `void count_sort_calculate_counts(int input_array[], int len, int count_array[], int* befehle)`
2. `void count_sort_write_output_array(int output_array[], int len, int count_array[], int* befehle)`
3. `void count_sort(int array[], int len, int* befehle)`
4. `void insertion_sort(int array[], int len, int* befehle)`

Hierbei sollen die Funktionen 1, 2, und 4 die gleiche Funktionalität wie auf dem Aufgabenblatt 1 haben, jedoch noch zusätzlich die ausgeführten Befehle mittels `int* befehle` zählen. Beachte zur Bestimmung der ausgeführten Befehle die Erläuterung aus Aufgabe 1 vom Blatt 2, sowie die Hinweise von Aufgabe 2, auch vom Blatt 2.

Die Funktion `count_sort` hat die gleiche Signatur, d. h. die gleichen Parameter und den gleichen Rückgabewert, wie die Funktion `insertion_sort`. Die Funktion `count_sort` soll hierbei die gesamte Funktionalität des Countsort-Algorithmus kapseln:

1. Erstelle zunächst mittels `malloc` ein Array zum Zählen der Häufigkeiten verschiedener Werte.
2. Rufe die Unterfunktionen `count_sort_calculate_counts` sowie `count_sort_write_output_array` so auf, dass das Ergebnis von Countsort (d. h. der Funktion `count_sort`) in das Eingabearray `int array[]` geschrieben wird.
3. Vergiss nicht, den allozierten Speicher wieder frei zu geben.
4. Die Anzahl ausgeführter Befehle von Countsort erstreckt sich über insgesamt 3 Funktionen und muss dementsprechend auch zusammengezählt werden.

Die Vorgabe (siehe Listing 1) erzeugt für Count- und Insertionsort jeweils ein Array mittels `malloc`: `array_countsort` sowie `array_insertionsort`. Das Array `array_countsort` wird zunächst mittels des Funktionsaufrufs `fill_array_randomly(array_countsort, n, MAX_VALUE);` mit Zufallswerten beschrieben. Anschließend werden mittels `copy_array_elements(array_insertionsort, array_countsort, n);` die gleichen Werte auch in das Array `array_insertionsort` geschrieben.

Aufgabe und Abgabemodalitäten:

- Implementiere die vier Funktionen und werte die Laufzeit sowie die Anzahl der benötigten Befehle beider Algorithmen aus.
- Teste die Algorithmen für verschiedene Werte der Konstanten `MAX_VALUE`. Finde Kombinationen aus `MAX_VALUE` und der Größe des Arrays n, bei denen jeweils entweder Insertionsort oder Countsort vorzuziehen ist.

Check Deinen Code als `introprog_complexity_steps_sorting.c` im SVN (im Ordner `introprog-wise1920/Studierende/<L>/<TU-Login>/Abgaben/Blatt03`, wobei <L> durch den ersten Buchstabe des TU-Logins zu ersetzen ist) ein.

Hinweise:

- Beachte, dass Du zum erfolgreichen Kompilieren des Programms zusätzlich die Quelldatei `introprog_complexity_steps_input.c` im Aufruf von `clang` übergeben musst.
- Beachte, dass Du den Code der `main`-Funktion weder anpassen musst noch anpassen sollst. Natürlich steht es Dir offen, die Werte des Arrays `int WERTE[]` oder die Konstante `MAX_VALUE` anzupassen. Dies kann z. B. nützlich sein, falls die Ausführung des Programms zu lange dauert.
- Du musst die main-Funktion keinerlei editieren.
- Bitte kommentiere Deinen Code genügend, um Klarkheit zu schaffen. Jedoch nicht zu viel (bspw. in jeder einzelnen Zeile), denn das würde Deinen Code wiederum unübersichtlicher machen.

Listing 1: Vorgabe `introprog_complexity_steps_sorting_vorgabe.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include "introprog_complexity_steps_input.h"

const int MAX_VALUE = 5000000;

void count_sort_calculate_counts(int input_array[], int len, int
    ↪ count_array[], int* befehle) {
    // Muss implementiert werden
}

void count_sort_write_output_array(int output_array[], int len, int
    ↪  count_array[], int* befehle) {
    // Muss implementiert werden
}

void count_sort(int array[], int len, int* befehle) {
    // Muss implementiert werden
}


void insertion_sort(int array[], int len, int* befehle) {
    // Muss implementiert werden
}


int main(int argc, char *argv[]) {

    const int WERTE[] = {10000,20000,30000,40000,50000};
    const int LEN_WERTE = 5;
    const int LEN_ALGORITHMEN = 2;

    int rc = 0;
    long befehle_array[LEN_ALGORITHMEN][LEN_WERTE];
    double laufzeit_array[LEN_ALGORITHMEN][LEN_WERTE];

    for(int j = 0; j < LEN_WERTE; ++j)
    {
        int n = WERTE[j];

        // Reserviere Speicher für  Arrays der Länge n
        int* array_countsort = malloc(sizeof(int) * n);
        int* array_insertionsort = malloc(sizeof(int) * n);

        // Fülle array_countsort mit Zufallswerten ..
        fill_array_randomly(array_countsort, n, MAX_VALUE);
        // ... und kopiere die erzeugten Werte in das Array
        // array_insertionsort
        copy_array_elements(array_insertionsort, array_countsort, n
            ↪ );

        // Teste ob beide Arrays auch wirklich die gleichen Werte
        // enthalten
        if(!check_equality_of_arrays(array_countsort,
            ↪ array_insertionsort, n))
        {
            printf("Die Eingaben für beide Algorithmen müssen für
                ↪ die Vergleichbarkeit gleich sein!\n");
            return -1;
        }

        for(int i = 0; i < LEN_ALGORITHMEN; ++i)
        {
```

```
59              int anzahl_befehle = 0;
60
61              start_timer();
62
63              // Aufruf der entsprechenden Sortieralgorithmen
64              if(i==0)
65              {
66                      count_sort(array_countsort, n, &anzahl_befehle)
                            ↪ ;
67              }
68              else if(i==1)
69              {
70                      insertion_sort(array_insertionsort, n, &
                            ↪ anzahl_befehle);
71              }
72
73              // Speichere die Laufzeit sowie die Anzahl benötigter
74              // Befehle
75              laufzeit_array[i][j] = end_timer();
76              befehle_array[i][j] = anzahl_befehle;
77          }
78
79          // Teste, ob die Ausgabe beider Algorithmen gleich sind
80          if(!check_equality_of_arrays(array_countsort,
                ↪ array_insertionsort, n))
81          {
82              printf("Die Arrays sind nicht gleich. Eines muss (
                    ↪ falsch) sortiert worden sein!\n");
83              rc = -1;
84          }
85
86          // Gib den Speicherplatz wieder frei
87          free(array_countsort);
88          free(array_insertionsort);
89      }
90
91      // Ausgabe der Anzahl ausgeführter Befehle sowie der gemessenen
92      // Laufzeiten (in Millisekunden)
93      printf("Parameter MAX_VALUE hat den Wert %d\n", MAX_VALUE);
94      printf("\t %32s           %32s \n", "Countsort","Insertionsort"
            ↪ );
95      printf("%8s \t %16s %16s \t %16s %16s \n", "n","Befehle", "
            ↪ Laufzeit","Befehle","Laufzeit");
96
97      for(int j = 0; j < LEN_WERTE; ++j)
98      {
99          printf("%8d \t ",WERTE[j]);
100         for(int i = 0; i < LEN_ALGORITHMEN; ++i)
101         {
102             printf("%16ld %16.4f \t ",  befehle_array[i][j],
                    ↪ laufzeit_array[i][j]);
103         }
104         printf("\n");
105     }
106
107     return rc;
108 }
```

Aufgabe 1.2 Fragen: Laufzeitvergleich (3,5 von 10 Punkten)

Im SVN findest Du unter
`Aufgaben/Blatt03/Vorgaben/introprog_complexity_steps_sorting_vorgabe.txt`
eine ausfüllbare Textdatei mit den folgenden Fragen:

- Inwieweit stehen die Anzahl der gezählten Befehle und die (empirisch gemessenen) Laufzeiten in Beziehung zueinander?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $k = n$ ist. Somit wächst die Größe des Wertebereichs linear mit der Größe der Eingabe. Welcher der beiden Algorithmen (Insertionsort bzw. Countsort) weist für große n im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $k = n^2$ ist. Somit wächst die Größe des Wertebereichs quadratisch mit der Größe der Eingabe. Welcher der beiden Algorithmen (Insertionsort bzw. Countsort) weist für große n im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $k = n^3$ ist. Somit wächst die Größe des Wertebereichs kubisch mit der Größe der Eingabe. Welcher der beiden Algorithmen (Insertionsort bzw. Countsort) weist für große n im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $n = 100.000$ und $k = 100$ ist. Welcher der beiden Algorithmen weist im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $n = 100.000$ ist und die Größe des Wertebereichs zur Entwicklungszeit nicht bekannt ist. k kann somit beliebig viel größer als n sein. Welcher der beiden Algorithmen weist im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?

- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $n = 100.000$ ist und die Größe des Wertebereichs bei $k = 100.000.000$ liegt. Wir nehmen an, dass die Eingabe schon maßgeblich aufsteigend sortiert ist. Konkret gelte $A[i] < A[i + j]$ für alle natürlichen Zahlen i in $\{1, 2, 3, \ldots, 99.998\}$ und alle j in $\{2, 3, 4, \ldots, n - i\}$ (unter Verwendung der Pseudocode-Konvention, dass Arrays bei 1 beginnen). Welcher der beiden Algorithmen weist im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?

Aufgabe und Abgabemodalitäten:
Kreuze in der Datei (unter Beachtung von sämtlichen Hinweisen und Vorgaben) die korrekte Antwort an und checke die modifizierte Datei im Abgabeordner für dieses Blatt als `introprog_complexity_steps_sorting.txt` im SVN ein.