**Fakultät IV - Elektrotechnik und Informatik**
*Einführung in die Programmierung WS 2014/15*
*Feldmann / Rost / Streibelt / Lichtblau / Deimel / Lucas*

# Aufgabenblatt 4

letzte Aktualisierung: 03. Dezember, 12:23 Uhr

(98a669abebca938e8d3e87e4eb06c37feb31c312)

*Ausgabe:* Mittwoch, 26.11.2014
*Abgabe:* spätestens Dienstag, 09.12.2014, 18:00

***Thema:*** *Divide-and-Conquer Prinzip: Implementierung Mergesort, Binary Search, Laufzeitmessung*

## Abgabemodalitäten

- Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des tubIT/IRB mittels `gcc -std=c99 -Wall` kompilieren.
- Abgaben erfolgen prinzipiell immer in Gruppen à 2 Personen, welche in den Tutorien festgelegt wurden. Einzelabgaben sind explizit als solche gekennzeichnet.
- Die Abgabe erfolgt ausschließlich über SVN. Die finale Abgabe
  - für Gruppenabgaben erfolgt im Unterordner `Tutorien/t<xx>/Gruppen/g<xx>/Blatt<xx>/submission/`
  - für Einzelabgaben erfolgt im Unterordner `Tutorien/t<xx>/Studierende/<tuBIT-Login>/Blatt<xx>/submission/`
- Benutzen Sie für alle Abgaben von Programmcode das folgende Namensschema: `introprog_blatt0X_aufgabe0Y_Z.c`, wobei X durch die Blattnummer, Y durch die Aufgabe und Z durch die Unteraufgabe ersetzen ist.
  *Beispiel:* `introprog_blatt01_aufgabe01_1.2.c`
  Für jede Unteraufgabe geben Sie maximal eine Quellcodedatei ab, es sei denn, die Aufgabenstellung erfordert explizit die Abgabe mehrerer Dateien pro Aufgabe.

  Alle anderen Abgaben (Pseudocode, Textaufgaben) benennen Sie wie oben beschrieben. Die zugelassenen Abgabeformate sind PDF, ODT und Text (txt). Auch hier verwenden Sie eine Datei pro Aufgabe.



## 1. Aufgabe: Implementierung von Mergesort (15 Punkte)

**Hinweis:** Geben Sie den Quelltext in einer Datei mit folgendem Namen ab:
`introprog_blatt04_aufgabe01.c`

In dieser Aufgabe sollen Sie den rekursiven Algorithmus *Merge Sort* implementieren, der auf dem Divide-and-Conquer Prinzip beruht.

Implementieren Sie die C Funktion `merge_sort()` anhand des Pseudocodes, der Ihnen in der Vorlesung vorgestellt wurde. Die Funktion bekommt als Argumente die Startadresse eines Integer Arrays sowie den Index des ersten Elements und die Länge des Arrays. Sie finden den Pseudocode in der Vorgabe `pseudocode-mergesort.txt`.

Verwenden Sie die Sortierfunktion in einem lauffähigen Programm. Ihr Programm bekommt als Argumente die maximale Länge des zu sortierenden Arrays und einen Dateinamen. Aus der angegebenen Datei lesen Sie die zu sortierenden Werte aus.

Ihr Programm soll für ein Array mit maximal 20 Einträgen wie folgt aufgerufen werden:

```
./introprog_blatt04_aufgabe01 20 zahlen_unsortiert.txt
```

Ausgabe des Programms sind die sortierten Werte, wobei jede Zahl durch einen Zeilenumbruch getrennt sein muss. Also im Format:

```
1
2
3
...
```

Zur Implementierung können Sie folgende Vorgabe verwenden:

Listing 1: Vorgabe `introprog_blatt04_aufgabe01_vorgabe.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

// Gibt das Array im korrekten Format aus.
void print_array(int array[], int len) {
    printf("Werte:");
    for (int i = 0; i < len ; i++) {
        printf("%d␣\n", array[i]);
    }
}

/*
Diese Funktion implementiert den Mergesort Algorithmus auf einem Array

    array:  Pointer auf den Beginn des Arrays
    first:  Index des ersten Elements
    length: Anzahl der Elemente des Arrays
*/

/*
In der folgenden Zeile findet ihr die alte Vorgabe. In der neuen
Funktionssignatur wird ein explizites Startelement übergeben.
void merge_sort(int* array, int length)
*/
```

```c
void merge_sort(int* array, int first, int last)
{
 // HIER Funktion merge_sort() implementieren
}

/*
Hauptprogramm.

Liest Integerwerte aus einer Datei (eine Zahl pro Zeile) und gibt
diese sortiert im selben Format (eine Zahl pro Zeile) über die
Standardausgabe wieder aus.

Aufruf: ./introprog_blatt04_aufgabe01_2 <maximale anzahl>  <dateipfad>
*/
int main (int argc, char *argv[])
{
    if (argc!=3){
        printf ("usage: %s <maximale anzahl> <dateipfad>\n", argv[0]);
        exit(2);
    }
    // HIER "array" einlesen

    printf("Eingabe:\n");
    print_array(array, len);

    // HIER Aufruf von "merge_sort()"

    printf("Sortiert:\n");
    print_array(array, len);

    return 0;
}
```



## 2. Aufgabe: Implementierung: Suche in sortierten Arrays (15 Punkte)

In dieser Aufgabe sollen Sie zwei verschiedene Suchalgorithmen implementieren, die feststellen, ob ein gegebener Integerwert in einem *aufsteigend sortierten* Array von Integerwerten enthalten ist oder nicht. Der `Binary Search` Algorithmus verwendet so wie der Merge Sort Algorithmus das Divide-And-Conquer Prinzip, während die lineare Suche dieses nicht nutzt. Implementieren Sie beide Algorithmen als Funktionen.

Um die Funktion zu testen und abzugeben, kombinieren Sie sie mit einer `main` Funktion. Das Programm wird wie folgt aufgerufen:

```
./introprog_blatt04_aufgabe02_1 10 suchwerte.txt 1000000
   ↪ zahlen_sortiert.txt
./introprog_blatt04_aufgabe02_2 10 suchwerte.txt 1000000
   ↪ zahlen_sortiert.txt
```

Jeweils das erstes Argument enthält die maximale Anzahl an Suchwerten und das zweite Argument den Pfad zur Datei, die die zu suchenden Werte enthält.
Das dritte Argument enthält die maximale Anzahl der Elemente des Arrays (in dem gesucht werden soll) und das vierte Argument den Pfad zur Datei mit den einzulesenden Werten.
Die entsprechenden Dateien finden Sie im SVN sowie auf ISIS.

Das Programm soll für jede Zahl aus der Datei `suchwerte.txt` jeweils eine Zeile ausgeben. Die Zeile enthält zuerst die gesuchte Zahl und dann `ja` für gefunden und `nein` für nicht gefunden. Ein Beispiel:

```
183260 ja
28772 nein
470323 nein
239 ja
```

Zur Implementierung können Sie folgende Vorgabe verwenden:

Listing 2: Vorgabe `introprog_blatt04_aufgabe02_vorgabe.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

// HIER die Funktion lineare Suche bzw. binäre Suche implementieren.
// Die Funktionen sollen in jeweils separaten Dateien abgegeben werden.
// Siehe Aufgabenblatt

/*
Hauptprogramm.

Überprüft, ob Zahlenwerten in einem Array enthalten sind

Aufruf: ./introprog_blatt04_aufgabe01_2 <max anzahl> <datei suchwerte> <max
   ↪ anzahl> <datei zahlen>

Ausgabe: Per Suchwert eine Zeile mit dem String:
    "<Suchwert> ja" bzw. "<Suchwert> nein"

    Also z.B. "12345 ja"
*/
int main (int argc, char *argv[])
{
    if (argc!=5){
```

```
24          printf ("usage: %s <max sucherte> <datei suchwerte> <max zahlen> <
               ↪ datei zahlen>\n", argv[0]);
25          exit(2);
26      }
27
28      // HIER "array" und "suchwerte" initialsieren
29      int len = // HIER "array" einlesen
30      int len2 =  // HIER "suchwerte" einlesen
31
32      for (int i=0; i<len2 ; i++) {
33          // HIER Suchfunktion für suchwerte[i] aufrufen
34          //Ausgabe ist entweder
35          //printf("%d ja\n", suchwerte[i]);
36          // oder
37          //printf("%d nein\n", suchwerte[i]);
38      }
39      return 0;
40  }
```



**2.1. (5 Punkte)** **Hinweis:** Geben Sie den Quelltext dieser Unteraufgabe in einer Datei mit folgendem Namen ab:
`introprog_blatt04_aufgabe02_1.c`

Implementieren Sie *Linear Search* als Funktion auf einem Array.

```
1  LinearSearch(Array A, start, end, value)
2      i ← start
3      while i <= end do
4          if A[i] == value then
5              return True
6          i ← i+1
7      return False
```

**2.2. (10 Punkte)** **Hinweis:** Geben Sie den Quelltext dieser Unteraufgabe in einer Datei mit folgendem Namen ab:
`introprog_blatt04_aufgabe02_2.c`

Implementieren Sie *Binary Search* als rekursive Funktion auf einem Array.

```
1  BinarySearch(Array A, start, end, value)
2      if start > end then
3          return False
4
5      middle ← (start + end)/2
6      if value == A[middle] then
7          return True
8      else if value < A[middle] then
9          end ← middle − 1
10     else then
11         start ← middle + 1
12     return BinarySearch(A, start, end, value)
```

## 3. Aufgabe: Laufzeitvergleich: Suche in sortierten Arrays (10 Punkte)

**Hinweis:** Leider hat sich in unsere erste Aufgabenstellung ein Denkfehler eingeschlichen. Daher hier die korrigierte Fassung mit aussagekräftigen Messungen. Korrektur vom 1.12.14, 22:15 Uhr

Geben Sie Ihre Lösung der Aufgabe in einem Textdokument mit einem der folgenden Namen ab: `introprog_blatt04_aufgabe03.{txt|odt|pdf}`(Der Quellcode muss bei dieser Aufgabe nicht mit abgegeben werden.)

Messen Sie die Laufzeit der zwei von Ihnen implementierten Suchalgorithmen! Kopieren Sie dazu jeweils einen Ihrer beiden Suchalgorithmen in die Programmvorlage und führen Sie das Programm wie folgt aus:

Ihnen stehen 4 verschiedene Eingabedateien mit 1000, 10000, 100000 und 1000000 Werten zur Verfügung. Zu jeder Eingabedatei gibt es eine korrespondierende Datei mit je 5 zu suchenden Werten.

Führen Sie insgesamt 8 Messungen durch: Messen Sie die reale Laufzeit der beiden Algorithmen mit jeweils allen 4 Eingabedateien und den korrespondierenden Suchwerten.

Ein Beispielaufruf sieht wie folgt aus:

```
./linear 5 eingabedaten/suche_1000.txt 1000 eingabedaten/zahlen_1000.txt
./binary 5 eingabedaten/suche_1000.txt 1000 eingabedaten/zahlen_1000.txt
```

**Hinweis:** Die Codevorgabe enthält einige Kommentare, die zum Verständnis der von Ihnen durchzuführenden Messungen wichtig sind. Lesen Sie diese sorgfältig durch!

Listing 3: Vorgabe

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <assert.h>
4  #include <time.h>
5  #include <string.h>
6
7  /*
8   TODO: Fuegen Sie hier alle benoetigten Funktionen ein,
9         um Ihren jeweiligen Algorithmus zu benutzen.
10 */
11
12 /*
13 Hauptprogramm.
14
15 Führt Laufzeitmessungen über einen Suchalgorithmus aus.
16
17 Aufruf: ./introprog_blatt04_aufgabe03_suchXXX <max suchwerte> <datei
     ↪ suchwerte> <max zahlen> <datei zahlen>
18
19 */
20 int main (int argc, char *argv[])
21 {
22     if (argc!=5){
23         printf ("usage: %s <max suchwerte> <datei suchwerte> <max
              ↪ zahlen> <datei zahlen>\n", argv[0]);
24         exit(2);
25     }
```



```
26
27     // Arrays anhand der Kommandozeilenparameter initialisieren
28     // int *array = ...
29     // int *suchwerte = ...
30
31     // Dateien in Arrays einlesen
32     // int len = ...
33     // int len2 = ...
34
35
36     /*
37        Wie oft soll der Algorithmus ausgefuehrt werden?
38        Muss fuer alle Messungen identisch sein, kann verringert
39        werden wenn der Rechner zu langsam ist und dadurch die
40        Ausfuehrung zu lange dauert.
41        ACHTUNG: Dann muessen alle Messungen wiederholt werden!
42     */
43     int durchgaenge = 2000;
44
45
46     // TODO: Hier den Namen des Algorithmus eintragen:
47     char* algoname = "XXX Suche";
48
49     clock_t start_ticks = clock();
50
51     for (int repeat=0; repeat < durchgaenge; repeat++) {
52         for (int i=0; i<len2; i++) {
53
54             // TODO: HIER DEN SUCHALGORITHMUS EXAKT EINMAL AUFRUFEN:
55             // z.B.: search_binary(...); ODER search_linear(...);
56
57         }
58     }
59     clock_t ticks = clock() - start_ticks;
60     printf("%s: %d Durchgaenge in %ju ticks, durchschnittlich %f Ticks
         ↪ pro Durchgang\n", algoname, durchgaenge, ticks, ticks/(
         ↪ durchgaenge*1.0));
61
62     //TODO: Speicher ggf. freigeben:
63     // free(array);
64     // free(suchwerte);
65     return 0;
66 }
```

Geben Sie ein Textdokument mit den Ergebnissen Ihrer Messung in Form einer Tabelle oder eines Graphen ab. Den verwendeten Code brauchen Sie nicht abgeben. Beantworten Sie zusätzlich folgende Frage:

Welche Laufzeitkomplexitäten der beiden Algorithmen können Sie anhand Ihrer Messdaten abschätzen? Bitte geben Sie eine kurze Begründung an.