Prof. Dr. E. Ábrahám

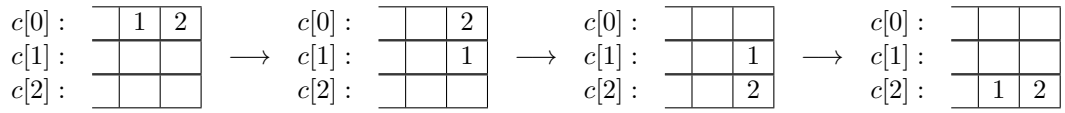
F. Corzilius, S. Schupp, T. Ströder

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2 bis 3 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Montag, den 28.04.2014 um 9:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch in Ihrem Tutorium vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.
- Sofern nicht näher spezifiziert, steht die Notation log ohne Angabe einer Basis für den natürlichen Logarithmus.
- Wenn Sie in einer Aufgabe einen **Algorithmus** erstellen müssen, so soll dieser in **Pseudo-Code** (gerne an `Java` bzw. `C++` orientiert) oder in `Java` bzw. `C++` verfasst werden. Reichen Sie den Algorithmus **nur in Maschinenschrift** ein, entweder Ihrem Tutor vor dem Abgabedatum per Email oder ausgedruckt an die Lösung geheftet. Einfache Syntaxfehler (z. B. ein fehlendes Semikolon) geben keine Punktabzüge. Inwieweit im Pseudo-Code abstrahiert werden darf, wird in der Aufgabenstellung bekannt gegeben. Ansonsten gilt es, sich möglichst nahe an den Möglichkeiten, die `Java` bzw. `C++` bieten, zu orientieren.

Tutoraufgabe 1 (Algorithmenentwurf):

Entwerfen Sie einen iterativen (also nicht rekursiven) Algorithmus, der ein Array entgegennimmt, das zwei leere und einen gefüllten Container enthält. In den Containern liegen alle Elemente (paarweise verschiedene ganze Zahlen) **stets aufsteigend sortiert** (*) vor. Der Algorithmus soll die Elemente im gefüllten Container vollständig in einen der leeren Container verschieben, wobei alle Container so genutzt werden dürfen, dass (*) nie verletzt wird.
Beispiel (für a)):



Wir setzen voraus, dass der gefüllte Container der erste (`c[0]`) ist und alle Container beliebig viele Elemente enthalten dürfen.

a) Die Datenstruktur der Container ist ein **Stapelspeicher/Stack**, d. h. man darf immer nur ein Element von vorne (also das kleinste) wegnehmen oder ein neues (kleinstes) vorne anfügen. Außerdem kann man nur das oberste Element eines Stacks einsehen und prüfen, ob er leer ist.

b) Die Datenstruktur der Container ist eine **Warteschlange/Queue**, d. h. man darf immer nur ein Element von vorne (also das kleinste) wegnehmen oder ein neues (größtes) hinten anfügen. Außerdem kann man nur das oberste Element einer Queue einsehen und prüfen, ob sie leer ist.

Aufgabe 2 (Algorithmenentwurf): (2 + 4 = 6 Punkte)

Entwerfen Sie einen iterativen (also nicht rekursiven) Algorithmus mit genau einer Schleife, der ermittelt, ob sich eine gegebene ganze Zahl in einer Reihe (Array) ganzer Zahlen mit Mindestlänge 3 befindet.

a) Die gegebene Reihe ist dabei **unsortiert** und der Algorithmus soll im schlimmsten Fall so viele Schleifen-Iterationen benötigen wie es Elemente in der gegebenen Reihe gibt.

b) Die gegebene Reihe ist dabei **sortiert** und der Algorithmus soll im schlimmsten Fall weniger Schleifen-Iterationen benötigen als es Elemente in der gegebenen Reihe gibt.

Tutoraufgabe 3 (Best- und Worst-Case):

Ein Algorithmus zum effizienten Potenzieren von der Zahl 5 (Annahme: $e \geq 0$):

```
int power5(int e)
{
    int res = 1;
    int tmp = 5;
    while(e > 0)
    {
        if(e.isOdd()) // Prüft, ob e ungerade ist.
        {
            e -= 1;
            res *= tmp;
        }
        e /= 2;
        tmp = tmp^2;
    }
    return res;
}
```

Den Hauptanteil an der Laufzeit für diesen Algorithmus nehmen dabei die nötigen **Multiplikationen** ein. Dabei ist das Quadrieren ebenfalls als Multiplikation zu zählen.

a) Geben Sie die Anzahl der nötigen Multiplikationen für den Parameter $e = 24$ an.

b) Gesucht sind zu obigem Algorithmus zwei Eingaben bestehend aus $0 < e_w < n, n < e_b < \infty$ zu einem beliebigen, aber festen $n \in \mathbb{N}$, welche die **Worst-Case** und die **Best-Case** Laufzeit erzeugen unter der Eingabe der Länge n. Die Eingabe für die Worst-Case Laufzeit soll dabei das kleinste mögliche $e < n$ sein, welches diese erzeugt. Die Eingabe für die Best-Case Laufzeit soll das kleinste mögliche $e > n$ sein, welches diese erzeugt. Erklären Sie bitte kurz (informell), wie Sie zu den angegebenen Lösungen gekommen sind.

Aufgabe 4 (Best- und Worst-Case): (1 + 3 = 4 Punkte)

```
void sort(int[] _array)
{
    bool changed = false;
    do
    {
        changed = false;
        for(int i = 0; i <= _array.size()-2; ++i)
        {
            if(_array[i] > _array[i+1])
            {
                int tmp = _array[i];
                _array[i] = _array[i+1];
                _array[i+1] = tmp;
                changed = true;
            }
        }
    }
    while(changed)
}
```

a) Führen Sie den Algorithmus (auf Papier) für die Eingabe [1,5,2,4,3] aus.

b) Gesucht sind zu dem gegebenen Algorithmus zwei Eingaben, welche zu einem beliebigen, aber festen n, welches die Größe der Eingabe beschreibt, die Worst-Case und die Best-Case Laufzeit erzeugen. Erklären Sie kurz Ihre Lösung.

Tutoraufgabe 5 ($\mathcal{O}$-Notation):

Beweisen oder widerlegen Sie die folgenden Aussagen. Hierbei sind f, g und h positive Funktionen von $\mathbb{N}$ nach $\mathbb{R}^{>0}$.

a) $2^{n+1} \in \Omega(2^n)$

b) $\max\{f(n), g(n)\} \in \Theta(f(n) + g(n))$

c) $f(n) \in \mathcal{O}(g(n)) \wedge \lim_{n\to\infty} \frac{h(n)}{g(n)} = 0 \implies f(n) + h(n) \in \mathcal{O}(g(n))$

d) $0 \in \Theta(1)$

Aufgabe 6 ($\mathcal{O}$-Notation): (3 + 3 + 3 + 3 = 12 Punkte)

Beweisen oder widerlegen Sie die folgenden Aussagen. Hierbei sind f, g und h positive Funktionen von $\mathbb{N}$ nach $\mathbb{R}^{>0}$.

a) $n^2 + \log(n) \in \mathcal{O}(\frac{n^5}{n^2+n})$

b) $\log(n) \in \Omega(\sqrt{n})$

c) $f(n) \in \mathcal{O}(g(n)) \wedge g(n) \in \mathcal{O}(h(n)) \implies f(n) \in \mathcal{O}(h(n))$

d) $\log_a(n) \in \Theta(\log_b(n))$ für zwei beliebige Konstanten $a, b > 1$