# Serie 4

*Komplexität von Algorithmen*

## Übungsaufgaben

### *Aufgabe 1 – Komplexität Matrixmultiplikation*

Gegeben sei die folgende einfache Implementierung zur Multiplikation von n×n Matritzen C = AB.

```
void mult(float** A, float** B, float** C, int n) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      C[i][j] = 0;
      for (int k = 0; k < n; k++) {
        C[i][j] = C[i][j] + A[i][k] * B[k][j];
      }
    }
  }
}
```

Bestimmen Sie den Zeitaufwand T(n) des Algorithmus in Abhängigkeit der Matrixgröße n. Zur Vereinfachung soll angenommen werden, dass Zuweisungen, Vergleiche sowie mathematische Operationen jeweils 1 Zeiteinheit benötigen. Welcher Komplexitätsklasse (in Θ-Notation) lässt sich der Algorithmus zuordnen?

### *Aufgabe 2 – Schnellere Matrixmultiplikation*

Ran Raz hat im Jahr 2002 nachgewiesen, dass die untere Schranke für die Matrixmultiplikation bei $\Omega(n^2 \log(n))$ liegt. Ein solcher Algorithmus ist jedoch bisher nicht bekannt. Der derzeit schnellste bekannte Algorithmus aus dem Jahr 2014 stammt von François Le Gall und erreicht $O(n^{2,3728639})$.

Wie lange benötigt unser Algorithmus aus Aufgabe 1 um 10×10, 50×50 und 100×100 Matritzen zu multiplizieren? Hierzu gehen wir davon aus, dass T(n) die Anzahl der Zeiteinheiten bestimmt, die der Algorithmus benötigt. Ferner nehmen wir der Einfachheit halber an, dass auf einem konkreten Rechner eine Zeiteinheit circa 1ms entspricht.

Wieviel Zeit benötigen wir beim Multiplizieren mit den Algorithmen von Le Gall und was wäre theoretisch laut Raz zu erreichen? Konstante Faktoren seien zur Vereinfachung c = 1.

| n | Aufgabe 1 $\Theta(n^3)$ | Le Gall $O(n^{2,3728639})$ | Raz $\Omega(n^2 \log(n))$ |
|---|---|---|---|
| 10 | 1 s | | |
| 50 | | | |
| 100 | | | |

### *Aufgabe 3 – Aufwand von Minimum- und Maximumsuche*

Aus der Vorlesung sind drei Algorithmen zur gleichzeitigen Bestimmung des Minimums und des Maximums in einem Array von *n* Zahlen bekannt.

Algorithmus 1: Vergleich auf Maximum *und* Vergleich auf Minimum

Algorithmus 2: Vergleich auf Maximum mit bedingtem Vergleich auf Minimum

Algorithmus 3: Vergleich benachbarter Zahlen mit anschließendem Vergleich auf Max. und Min.

Analysieren Sie die Algorithmen bezüglich Ihres Zeitaufwandes T(n)! Als prinzipielle Kosten verursachende Operation betrachten wir der Einfachheit halber **nur Elementvergleiche**, das heißt Vergleiche an denen ein Element des Arrays beteiligt ist.

a) Übernehmen Sie die drei Algorithmen als C++ Funktionen in Ihrem Programm.

b) Erweitern Sie die Funktionen so, dass die **Anzahl der notwendigen Elementvergleiche** ermittelt und via ***call-by-reference*** zurückgegeben wird. Dazu sind die Funktionen um einen Referenzparameter zu erweitern.

c) Geben Sie im Hauptprogramm die Anzahl der Vergleiche aller drei Funktionen jeweils für den *Worst-case*, *Avg-case* und den *Best-case* aus. Füllen Sie hierfür ein zu durchsuchendes Array geeignet mit Zahlen, um den schlechtesten, besten und durchschnittlichen Fall zu testen. Vergleichen Sie Ihre Testläufe mit $n = 100$, $n = 1000$ und $n = 10000$.

# Hausaufgaben

### *Aufgabe 1 – Obere, Untere und Enge Schranke (Punkte: 7)*

Gegeben ist eine Tabelle mit Funktionen $f(n)$ und $g(n)$ . Bestimmen Sie für jede Zeile der Tabelle, welche der Aussagen gilt $f(n)=O(g(n))$ , $f(n)=\Omega(g(n))$ oder $f(n)=\Theta(g(n))$ ! Begründen Sie Ihre Antwort!

| $f(n)$ | $g(n)$ |
|---|---|
| $\sqrt{n}$ | $1000\,n$ |
| $\log_{10} n$ | $\log_2 n$ |
| $\sqrt[3]{n}$ | $\sqrt{n}$ |
| $n^2$ | $n\cdot\log n$ |
| $6n^2-5n+4$ | $n^2$ |
| $n\cdot\log n+\sqrt{n}$ | $n\cdot\log^2 n$ |
| $(n+a)^b$ | $n^b$ ($a$, $b$ konstant, $b > 0$) |

Hinweis:

Für $f(n)=O(g(n))$ ist zu zeigen, dass $\lim\limits_{n\to\infty}\frac{f(n)}{g(n)}=0$ .

Für $f(n)=\Omega(g(n))$ ist zu zeigen, dass $\lim\limits_{n\to\infty}\frac{g(n)}{f(n)}=0$ .

Für $f(n)=\Theta(g(n))$ ist zu zeigen, dass $\lim\limits_{n\to\infty}\frac{f(n)}{g(n)}=c$ , $c$ konstant.

### *Aufgabe 2 – Fehlerfindung und Messen der Laufzeit (Punkte: 3)*

Ein Programmierer hat den Auftrag ein Programm zu schreiben, dass die Laufzeit von geschachtelten Schleifen misst. Leider sind ihm beim Programmieren Fehler unterlaufen. Folgende fehlerbehaftete Lösung existiert:

```
#include <iostream>
#include <iomanip>
#include <ctime>
using namespace;

int main() {
      long c; const int n = 50; time_t t;

      cout << "Two nested loops O(n^2)" << endl;
      time(&t);
      c = 0;
      for (int i = 0; i < n; i++) {
            for (const int j = 0; j < n; j++) {
                  c++;
                  cout << "Count: " << setw(10) << c << "\r";
            }
      }
      t = time(0) - t;
      cout << endl << "Required time: " << t << "s" << endl;

      cout << "Three nested loops O(n^3)" << edl;
      time(&t);
      c = 0;
      for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                  for (k = 0; k < n; k++) {
                        c++;
                        cout << "Count: " << setw(10) << c << "\r";
                  }
            }
      }
      t = time(0) - t;
      cout << endl << "Required time: " << "s" << endl;
}
```

Finden und korrigieren Sie die Fehler! Benutzen Sie das Programm anschließend, um die Laufzeit der doppelt und der dreifach geschachtelten Schleife auf Ihrem Rechner zu messen. Messen Sie für $n = 50$, $n = 100$ und $n = 150$ die Laufzeit!

*Hinweis: Ihre Lösung soll den korrigierten Quelltext, die Messergebnisse als auch eine kurze Beschreibung des Testsystems (CPU, RAM, Betriebssystem) umfassen.*