# VL02, Lösung 1

| Methode | Anzahl Aufrufe für $n$=100 | Als Funktion von $n$ | O-Notation |
|---|---|---|---|
| `proz1` | 200 | $f(n) = 2n$ | $O(n)$ |
| `proz2` | 10000 | $f(n) = n^2$ | $O(n^2)$ |
| `proz3` | 11 | $f(n) = \lfloor\sqrt{n}\rfloor + 1$ | $O(\sqrt{n})$ |
| `proz4` | 98990100 | $f(n) = 100(n^2-1)(n-1)$ | $O(n^3)$ |
| `proz5` | 5050 | $f(n) = 1+2+3+\ldots+n = \frac{n(n+1)}{2}$ | $O(n^2)$ |
| `proz6` | 7 | $f(n) = \lfloor\log_2 n\rfloor + 1$ | $O(\log_2 n)$ |

$\lfloor\,\rfloor$ bezeichnet die nächst kleinere ganze Zahl (Gaußsche Klammer).

| Aufwandsfunktion $f(n)$ | $f(n) = O(\ldots)$ | Beweis |
|---|---|---|
| `proz1`: $f(n) = 2n$ | $f(n) = O(n)$ | $f(n)$ ist bereits ein Vielfaches von $n$. Mit $k = 2$ und $n_0 = 1$ gilt: $f(n) \le k*n$ für alle $n \ge n_0$. |
| `proz2`: $f(n) = n^2$ | $f(n) = O(n^2)$ | $f(n) = n^2$. Mit $k = 1$ und $n_0 = 1$ gilt: $f(n) \le k*n^2$. |
| `proz3`: $f(n) = \lfloor\sqrt{n}\rfloor + 1$ | $f(n) = O(\sqrt{n})$ | Dazu müssen wir zeigen, dass es Konstanten $k$ und $n_0$ gibt, so dass $f(n) \le k*\sqrt{n}$ für alle $n \ge n_0$. Wir versuchen, $f(n)$ nach oben hin durch ein Vielfaches von $\sqrt{n}$ abzuschätzen.<br>$\lfloor\sqrt{n}\rfloor+1 \le \sqrt{n}+1$, da der ganzzahlige Anteil $\lfloor\sqrt{n}\rfloor$ von $\sqrt{n}$ niemals größer werden kann als $\sqrt{n}$ (die Nachkommastellen gehen beim ganzzahligen Anteil verloren). Wir haben also $\lfloor\sqrt{n}\rfloor$ nach oben hin durch $\sqrt{n}$ abgeschätzt.<br>$\sqrt{n}+1 \le \sqrt{n}+\sqrt{n}$, da für $n \ge 1$ gilt: $1 \le \sqrt{n}$. Wir haben also 1 nach oben hin durch $\sqrt{n}$ abgeschätzt: $\sqrt{n}+\sqrt{n} = 2*\sqrt{n}$.<br>Insgesamt gilt also: $f(n) \le 2*\sqrt{n}$, für alle $n_0 \ge 1$. |
| `proz4`: $f(n) = 100(n^2-1)(n-1)$ | $f(n) = O(n^3)$ | $100*(n^2\text{-}1)(n\text{-}1) < 100*n^2*n$ für alle $n \ge 1$, da gilt: $n^2\text{-}1 < n^2$ und $n\text{-}1 < n$.<br>Wenn Faktoren größer werden, wird auch das Produkt größer. Insgesamt gilt also: $f(n) < 100*n^3$.<br>Also gilt mit $k = 100$ und $n_0 = 1$: $f(n) \le k*n^3$. |

| Aufwandsfunktion f(n) | f(n) = O(...) | Beweis |
|---|---|---|
| `proz5`: $f(n) = \frac{n(n+1)}{2}$ | $f(n) = O(n^2)$ | $n*(n+1)/2 < n*2*(n+1)/2$; der Wert $n$ wird verdoppelt, wird also größer. $n*2*(n+1)/2 = n*(n+1) \leq n*(n+n)$, da gilt: $1 \leq n$. Insgesamt ergibt sich also: $f(n) = n*(n+1)/2 < 2*n^2$. Somit ist $f(n) \leq k*n^2$ mit $k = 2$ und $n_0 = 1$. |
| `proz6`: $f(n) = \lfloor \log_2 n \rfloor + 1$ | $f(n) = O(\log_2 n)$ | $\lfloor \log_2(n) \rfloor + 1 \leq \log_2(n)+1$ aufgrund derselben Argumentation wie bei `proz3`. $\log_2(n)+1 \leq \log_2(n)+\log_2(n)$ für $n \geq 2$, da dann gilt: $1 \leq \log_2(n)$. Somit folgt: $\log_2(n)+\log_2(n) = 2*\log_2(n)$. Insgesamt gilt also: $f(n) = \lfloor \log_2(n) \rfloor + 1 \leq 2*\log_2(n)$. Also gilt $f(n) \leq k*\log_2(n)$ mit $k = 2$ und $n_0 = 2$. |

Faustregeln:

1) Ist $f(n)$ ein Polynom, so ist $f(n)$ von der Ordnung der größten Potenz, die in diesem Polynom vorkommt. **Beispiel:** $200n^4 + 3n^3 + n + 20 = O(n^4)$.
2) Ist $f(n) = k_1*\log_2 n + k_2$ für zwei Konstanten $k_1$ und $k_2$, so gilt: $\mathbf{f(n) = O(\log_2 n)}$. **Beispiel:** $\mathbf{f(n) = 10 * \log_2 n + 32 = O(\log_2 n)}$.
3) Ist $\mathbf{f(n) = k_1 * \sqrt{n} + k_2}$ für zwei Konstanten $k_1$ und $k_2$, so gilt: $\mathbf{f(n) = O(\sqrt{n})}$. **Beispiel:** $\mathbf{f(n) = 1050 * \sqrt{n} + 113 = O(\sqrt{n})}$.

## VL02, Lösung 2

1) Gilt $7n \log_2 n = O(n)$?

   **Die Aussage ist falsch**, denn $\mathbf{\lim_{n\to\infty} \frac{7n \log_2 n}{n}}$ divergiert bestimmt gegen ∞.

2) Gilt $n^3+2^n = O(2^n)$?

   **Die Aussage ist richtig**, denn $\mathbf{\lim_{n\to\infty} \frac{n^3+2^n}{2^n}}$ konvergiert gegen 1. Da auch $\mathbf{\lim_{n\to\infty} \frac{2^n}{n^3+2^n}}$ konvergiert (reziproker Bruch), ist $O(2^n)$ sogar die genaue Ordnung.

3) Gilt $3((n \bmod 3)+1) = O(1)$?

   **Die Aussage ist richtig.** $\mathbf{\lim_{n\to\infty} \frac{3((n \bmod 3)+1)}{1}}$ divergiert zwar unbestimmt, denn es handelt sich um die alternierende Folge 6, 9, 3, 6, 9, 3, ... . Wir können in diesem Sonderfall, der in der Praxis nur sehr selten auftritt, jedoch mit 9 eine konstante obere Schranke angeben, so dass gilt: $3((n \bmod 3)+1) \leq 9$.

4) Gilt $n^2+15n-3 = O(n^3)$?

**Die Aussage ist richtig.** $f(n)$ ist von der Ordnung des Summanden, der am schnellsten wächst. Daher gilt: $n^2+15n-3 = O(n^3)$. Außerdem gilt: $O(n^2) \subseteq O(n^3)$. Alternativ stellen wir fest, dass $\mathbf{lim}_{n\to\infty} \frac{\mathbf{n^2+15n-3}}{\mathbf{n^3}}$ gegen 0 konvergiert. Umgekehrt divergiert $\mathbf{lim}_{n\to\infty} \frac{\mathbf{n^3}}{\mathbf{n^2+15n-3}}$ jedoch gegen $\infty$, so dass $n^2+15n-3$ **nicht** die Ordnung $O(n^3)$ hat.

## VL02, Lösung 3

Die Funktionen werden paarweise verglichen:

| | | |
|---|---|---|
| $A_1/A_2$ | $4n + 32 < 6n$ | $\mid - 4n$ |
| $\Leftrightarrow$ | $32 < 2n$ | $\mid \div 2$ |
| $\Leftrightarrow$ | $16 < n$ | |
| $\Rightarrow$ | **Für n > 16 ist $A_1$ besser als $A_2$, für 1 ≤ n ≤ 16 ist $A_2$ besser als $A_1$.** | |

| | | |
|---|---|---|
| $A_1/A_3$ | $4n + 32 < n^2$ | $\mid - 4n$ |
| $\Leftrightarrow$ | $32 < n^2 - 4n$ | $\mid$ Quadratische Ergänzung |
| $\Leftrightarrow$ | $36 < (n - 2)^2$ | $\mid \sqrt{}$ |
| $\Leftrightarrow$ | $6 < n - 2$ | $\mid +2$ |
| $\Leftrightarrow$ | $8 < n$ | |
| $\Rightarrow$ | **Für n > 8 ist $A_1$ besser als $A_3$, für 1 ≤ n ≤ 8 ist $A_3$ besser als $A_1$.** | |

| | | |
|---|---|---|
| $A_2/A_3$ | $6n < n^2$ | $\mid \div n$ |
| $\Leftrightarrow$ | $6 < n$ | |
| $\Rightarrow$ | **Für n > 6 ist $A_2$ besser als $A_3$, für 1 ≤ n ≤ 6 ist $A_3$ besser als $A_2$.** | |

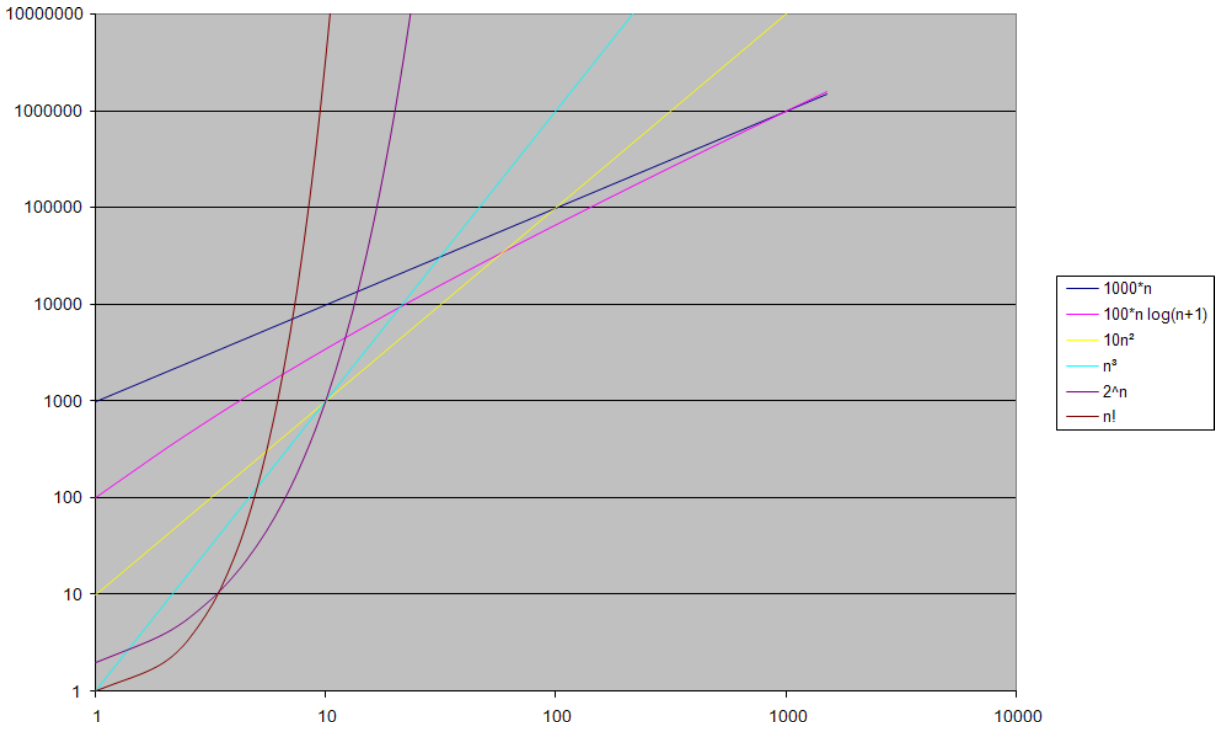
Also gilt:

1) Für $1 \leq n \leq 6$ ist $A_3$ besser als $A_2$ und besser als $A_1$
(Folgt aus $A_2/A_3$ und $A_1/A_3$)
2) Für $6 < n \leq 16$ ist $A_2$ besser als $A_3$ und besser als $A_1$
(Folgt aus $A_2/A_3$ und $A_1/A_2$)
3) Für $n > 16$ ist $A_1$ besser als $A_2$ und $A_3$
(Folgt aus $A_1/A_2$ und $A_1/A_3$)

| Problemgröße $n$ | Bester Algorithmus | Anmerkungen |
|---|---|---|
| 1…6 | $A_3$ $g_3(n) = n^2$ | Bei $n = 6$: $g_3(n) = g_2(n)$ |
| 7…16 | $A_2$ $g_2(n) = 6n$ | |
| Ab 17 | $A_1$ $g_1(n) = 4n + 32$ | Bei $n = 16$: $g_1(n) = g_2(n)$ |

# VL02, Lösung 4

Grafik, die das Zeitverhalten veranschaulicht:




Java-Programm:

```
public class BesterAlgorithmus
{
    // 1000n
    public static double g1(int n)
    {
        return 1000.0 * n;
    }

    // 100n * log2(n+1)
    public static double g2(int n)
    {
        return 100.0 * n * Math.log10(n+1) / Math.log10(2.0);
    }

    // 10 * n * n
    public static double g3(int n)
    {
        return 10.0 * n * n;
    }

    // n * n * n
    public static double g4(int n)
    {
        return (double)n * n * n;
    }
```

```
    // 2 hoch n
    public static double g5(int n)
    {
        double wert = 1.0;

        for (int a = 0; a < n; a++)
            wert *= 2.0;

        return wert;
    }

    // n!
    public static double g6(int n)
    {
        double wert = 1.0;

        for (int a = 2; a <= n; a++)
            wert *= a;

        return wert;
    }

    // Bestimmt fuer alle 6 Funktionen den Wert von
    // gi(n) und gibt den Index der Funktion mit dem
    // minimalen Wert zurueck
    public static int gewinnerFuerN(int n)
    {
        double[] werte = { g1(n), g2(n), g3(n), g4(n), g5(n), g6(n) };

        int minimum = 0;
        for (int a = 1; a < werte.length; a++)
            if (werte[a] < werte[minimum])
                minimum = a;

        return minimum + 1;
    }

    // Gibt für jede Zahl n zwischen 1 und 2000 aus, welcher der
    // 6 Algorithmen (A1 .. A6) für das betrachtete n der beste ist
    public static void main(String[] args)
    {
        for (int n = 1; n <= 2000; n++)
            System.out.println("Der Algorithmus A" + gewinnerFuerN(n) +
                " ist der Beste für n = " + n);
    }
}
```

Ergebnis des Vergleichs:

| **Problemgröße n** | **Bester Algorithmus** | **Anmerkungen** |
|---|---|---|
| 1…3 | $A_6$: $n!$ | **Bei 1**: $A_4$ genauso gut wie $A_6$ |
| 4…9 | $A_5$: $2^n$ | |
| 10…59 | $A_3$: $10n^2$ | **Bei 10**: $A_3$ genauso gut wie $A_4$ |
| 60…1023 | $A_2$: $100n * \log_2(n+1)$ | **Bei 1023**: $A_1$ genauso gut wie $A_2$ |
| Ab 1024 | $A_1$: $1000n$ | |

# VL02, Lösung 5

```
public static void main(String[] args)
{
    Scanner sc = new Scanner(System.in);
    System.out.println("Geben Sie die Problemgröße (n) ein: ");

    // n enthält die Problemgröße
    int n = sc.nextInt();

    StopUhr meineUhr = new StopUhr();

    meineUhr.start();
    Zeitmessung.func1(n);
    meineUhr.stop();
    System.out.println("Func1 (lineare Laufzeit): " +
        meineUhr.getDuration()/1000000.0 + " msec");

    meineUhr.start();
    Zeitmessung.func2(n);
    meineUhr.stop();
    System.out.println("Func2 (quadratische Laufzeit): " +
        meineUhr.getDuration()/1000000.0 + " msec");

    meineUhr.start();
    Zeitmessung.func6(n);
    meineUhr.stop();
    System.out.println("Func6 (logarithmische Laufzeit): " +
        meineUhr.getDuration()/1000000.0 + " msec");
}
```