Aufgabe 1. Algorithmische Komplexität

a) Laufzeiten der verschiedenen algorithmischen Komplexitäten $f(n)$ für $2n$ in Abhängigkeit von der Ausgangslaufzeit für n.

$f(n)$	4 Sekunden	10 Sekunden	100 Sekunden
$\mathrm{ld}(2n)$	5s	11s	101s
$2n$	8s	20s	200s
$2n\ \mathrm{ld}(2n)$	13,49s	29,13s	244,64s
$(2n)^3$	32s	80s	800s
2^{2n}	16s	100s	10000s

b) 1. 1
2. $\log\log n$
3. $\log n$
4. n^ϵ
5. n^c
6. $n^{\log n}$
7. c^n
8. n^n
9. $c^{(c^n)}$

c) Beweisen Sie folgende Behauptungen

(a) $x^a = O(x^b) \iff a - b \leq 0$

Beweis. Seien $a, b, x \in \mathbb{R}$ mit $x > 1$
Zu zeigen: $\exists c \in \mathbb{R}$: $x^a \leq cx^b \iff a \leq b$
Wegen $x > 1$ ist $\ln(x) > 0$ und $\ln(x)$ streng monoton steigend, wähle $c \geq 1$, dann folgt:

$$\begin{aligned} & a \leq b \\ \iff & \ln(x) \cdot a \leq \ln(x) \cdot b \leq \ln(c) + \ln(x) \cdot b \\ \iff & \ln(x^a) \leq \ln(c \cdot b^x) \\ \iff & x^a \leq cx^b. \end{aligned}$$

□

(b) $\log_a(x) = \Theta(\log_b(x))\ \forall a, b \in \mathbb{R}^+$

Beweis. Seien $a, b, x \in \mathbb{R}^+$.
Zu zeigen: $\exists c \in \mathbb{R}$: $\log_a(x) = c \cdot \log_b(x)$.

$$\begin{aligned} & \log_a(x) = \log_a(x) \\ \implies & \log_a(x) = \log_a(b) \cdot \frac{\log_a(x)}{\log_a(b)} \\ \implies & \log_a(x) = \log_a(b) \cdot \log_b(x). \end{aligned}$$

Mit $c := \log_a(b)$ folgt damit:

$$\log_a(x) = c \cdot \log_b(x).$$

□

(c) $a^x = O(b^x) \iff 0 \leq a \leq b$

Beweis. Seien $a, b, x \in \mathbb{R}$ mit $x \geq 0$
Zu zeigen: $\exists c \in \mathbb{R}$: $a^x \leq cb^x \iff 0 \leq a \leq b$

$$\begin{aligned} & 0 \leq a \leq b \\ \iff & a^x \leq b^x \leq b^{x+1} = b \cdot b^x. \end{aligned}$$

Mit $c := b$ folgt damit:

$$a^x \leq c \cdot b^x \iff 0 \leq a \leq b.$$

□

Aufgabe 2. Klassischer Euklidischer Algorithmus

Sei $a, b \in \mathbb{N}_0, a + b > 0$ gegeben.

$$\mathrm{ggT}(a,b) = \begin{cases} a & b = 0 \\ \mathrm{ggT}(b,a) & a < b \\ \mathrm{ggT}(a-b,b) & a \geq b \end{cases}.$$

1. Für $a \geq b > 0$ gilt:
$$\mathrm{ggT}(a,b) = \mathrm{ggT}(a-b,b).$$

Beweis. Wegen $b \neq 0$ und $a \geq b$ folgt nach Definition:
$$\mathrm{ggT}(a,b) = \mathrm{ggT}(a-b,b).$$
□

2. Der klassische Euklidische Algorithmus terminiert.

Beweis. Seien $(a_n)_{n\in\mathbb{N}} \in \mathbb{N}_0$ und $(b_n)_{n\in\mathbb{N}} \in \mathbb{N}_0$ Folgen mit $a_1 = a$ und $b_1 = b$.
Sei $n \in \mathbb{N}$ beliebig.

- Falls $b_n = 0$ terminiert der Algorithmus direkt.
- Falls $a_n < b_n$, folgt nach Definition:
$$a_{n+1} = b_n \text{ und } b_{n+1} = a_n < b_n.$$
Damit folgt:
$$b_{n+1} < b_n.$$
- Falls $a_n \geq b_n$ folgt nach Definition:
$$a_{n+1} = (a_n - b_n) \in \mathbb{N}_0 \text{ und } b_{n+1} = b_n.$$
Wegen $a_{n+1} < a_n$ folgt, dass $\exists k \in \mathbb{N}$: $a_{n+k} < b_n$. Dann tritt wieder der zweite Fall ein, d.h.
$$b_{n+k+1} < b_n.$$

Damit folgt, dass $(b_n)_{n\in\mathbb{N}}$ für fast alle $n \in \mathbb{N}$ streng monoton fällt. Da $(b_n)_{n\in\mathbb{N}} \in \mathbb{N}_0$, folgt:
$$\exists k \in \mathbb{N}\text{: } b_k = 0.$$
Damit terminiert der *klassische Euklidische Algorithmus* immer. □

Aufgabe 3. Binomialkoeffizient

a) Programm siehe *binomial.cc*

Für $n = 35$ und $k = 18$ benötigt das Programm mehr als 20 Sekunden für die Berechnung.

Für $n = 34$ und $k = 18$ liefert das Programm -2091005866. Das liegt an der begrenzten Größe des Datentyps **int**. Bei Überschreitung der maximalen Größe beginnt der Wert erneut bei dem Minimalwert des Datentyps **int**. Deshalb können wir dann negative Ergebnisse erhalten.

b) Der Rechenaufwand für die rekursive Berechnung des Binomialkoeffizienten ist:
$$A_{n,0} = A_{n,n} = A_{n,k>n} = 1$$
$$A_{n,k} = A_{n-1,k-1} + A_{n-1,k}.$$

c) Programm siehe *binomial_fast.cc*

Die schnellere Variante hat die Komplexität $O(n)$, da die Komplexität der Fakultätsfunktion $O(n)$ ist und diese einfach dreimal ausgeführt wird.

Programm (c) ist deutlich schneller als Programm (a) für größere Zahlen n und k. Aufgrund der Verwendung der Fakultät, wird allerdings schneller die maximale Größe eines **int**'s erreicht. Dadurch erhalten wir bereits für recht kleine Werte für n und k falsche Ergebnisse.

d) Ein effizienter Algorithmus berechnet jede Zeile linear iterativ, aus der vorhergehenden Zeile, damit ist die Komplexität $O(n)$.

Der Unterschied entsteht daraus, dass bei (a) einzelne Binomialkoeffizienten mehrfach ausgerechnet werden müssen und bei (d) jeder Binomialkoeffizient genau einmal ausgerechnet wird.