Übung 1 *Algorithmische Komplexität*

a) Ein Programm benötigt zur Bearbeitung einer Menge von Daten der Anzahl n 4 Sekunden auf einem Computer. Das Programm habe die algorithmische Komplexität $f(n)$. Wenn sich die Menge der Eingabedaten auf $2n$ verdoppelt, wie lange läuft dann das Programm wenn seine algorithmische Komplexität $f(n)$

i) $\operatorname{ld} n$ ii) n iii) $n \operatorname{ld} n$ iv) n^3 v) 2^n

beträgt? Wie sieht es aus für eine Ausgangslaufzeit von 10 und 100 Sekunden? Hier ist ld der Logarithmus zur Basis 2.

b) Sortieren Sie die folgenden Funktionen danach, wie schnell sie mit n wachsen. Beginnen Sie mit der langsamsten.

$$n^{\log n} \qquad c^n \qquad c^{(c^n)} \qquad \log\log n \qquad n^\epsilon \qquad \log n \qquad 1 \qquad n^n \qquad n^c$$

Hier gilt $0 < \epsilon < 1 < c$. Aus Aufgabe c) folgt, dass es nicht entscheidend ist welche Basis der Logarithmus hat.

c) Beweisen Sie folgende Behauptungen:

1. $x^a = O(x^b)$ genau dann, wenn $a - b \leq 0$.
2. $log_a(x) = \Theta(log_b(x))$ für alle a, b.
3. $a^x = O(b^x)$ genau dann, wenn $0 \leq a \leq b$.

(5 Punkte)

Übung 2 *Klassischer Euklidscher Algorithmus*

Sei $a, b \in \mathbb{N}_0$, $a + b > 0$ gegeben. Der *klassische Euklidsche Algorithmus* verwendet folgende Rekursion:

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

1. Beweisen Sie, dass für $a \geq b > 0$ gilt:

$$\mathrm{ggT}(a,b) = \mathrm{ggT}(a-b,b).$$

2. Beweisen Sie, dass der klassische Euklidsche Algorithmus terminiert.

(5 Punkte)

Übung 3 *Binomialkoeffizient*

In der Vorlesung haben Sie mit den Fibonaccizahlen ein Beispiel für eine Rekursionformel mit einem Argument n kennengelernt. Ein weiteres Beispiel ist der Binomialkoeffizient

$$
\begin{aligned}
B_{n,0} &= B_{n,n} = 1 && n \geq 0 \\
B_{n,k} &= B_{n-1,k-1} + B_{n-1,k} && 0 < k < n
\end{aligned}
\tag{1}
$$

mit zwei Argumenten n und k. Dieser lässt sich graphisch in Form des Pascalschen Dreiecks veranschaulichen:

$$
\begin{array}{ccccccccc|ccccccccc}
 & & & & B_{0,0} & & & & & & & & & 1 & & & & \\
 & & & B_{1,0} & & B_{1,1} & & & & & & & 1 & & 1 & & & \\
 & & B_{2,0} & & B_{2,1} & & B_{2,2} & & & & & 1 & & 2 & & 1 & & \\
 & B_{3,0} & & B_{3,1} & & B_{3,2} & & B_{3,3} & & & 1 & & 3 & & 3 & & 1 & \\
B_{4,0} & & B_{4,1} & & B_{4,2} & & B_{4,3} & & B_{4,4} & 1 & & 4 & & 6 & & 4 & & 1
\end{array}
$$

Eine explizite Formel zur Berechnung des Binomialkoeffizienten ist gegeben durch

$$
B_{n,k} = \binom{n}{k} = \frac{n!}{k!(n-k)!} \,. \tag{2}
$$

a) Schreiben Sie ein Programm, das den Binomialkoeffizient für beliebige n und k nach der in (1) gegebenen rekursiven Formel berechnet. Benutzen Sie hierzu den Datentyp `int`. Mit den Funktionen `enter_int` oder `readarg_int` können Sie n und k von der Standardeingabe einlesen oder als Kommandozeilenparameter übergeben. Fertigen Sie Ihre Implementierung in der Datei `binomial.cc` an.

   Benutzen Sie die Unix-Funktion `time`, um die Laufzeit Ihres Programms für verschiedene Kombinationen von n und k in der Konsole zu analysieren. Dabei schreiben Sie das Wort `time` vor das zu messende Kommando. Nach Ausführung des Kommandos, wird die dazu benötigte Zeit ausgegeben. Ein Beispiel:

```
> time ls
...
> real    0m0.007s
> user    0m0.004s
> sys     0m0.004s
```

   Von Interesse ist hier die Echtzeit der Dauer, dargestellt als `real` in diesem Fall.

   Geben Sie eine Kombination für n und k an, bei der Ihr Programm länger als 10 Sekunden für die Berechnung benötigt. Was liefert Ihr Programm für $n = 34$, $k = 18$? Können Sie dieses Ergebnis erklären?

b) In der Vorlesung wurde der (exponentielle) Rechenaufwand für die rekursive Berechnung der Fibonaccizahlen ermittelt. Hier bezeichne nun $A_{n,k}$ die Komplexität zur rekursiven Berechnung der Binomialkoeffizienten. Stellen Sie den Aufwand $A_{n,k}$ als Ausdruck in $B_{n,k}$ dar.

c) Schreiben Sie ein Programm, welches schneller arbeitet als das aus Aufgabenteil a). Benutzen Sie weiterhin den Datentyp `int`, nun aber die explizite Formulierung aus (2). Fertigen Sie Ihre Implementierung in der Datei `binomial_fast.cc` an. Welche asymptotische Komplexität hat Ihre schnellere Variante?

   Vergleichen Sie die beiden Programme aus a) und c) hinsichtlich der Geschwindigkeit und der berechneten Ergebnisse. Was stellen Sie für höhere Werte fest? Können Sie dies erklären?

d) Welchen Aufwand hat ein effizienter Algorithmus, der die ersten n Zeilen des Pascalschen Dreiecks auf den Bildschirm ausgibt? Vergleichen Sie den Aufwand mit dem aus a) und versuchen Sie eine Erklärung für den Unterschied zu geben.

(10 Punkte)