Aufgabe 1. Installiere GMP und kompiliere folgendes Beispielprogramm

```
/* Gebe einige Zufallszahlen aus. Anwendungsbeispiel
   fuer die Verwendung der GMP. */

#include <stdio.h>
#include <gmp.h>
#include <time.h>

int main() {
  mpz_t a,r;
  int i;

  /* Zustand des Zufallszahlengenerators: */
  gmp_randstate_t state;

  /* Initialisiere den Zufallszahlengenerator */
  gmp_randinit_default(state);
  gmp_randseed_ui(state, time(NULL));

  mpz_init(a);
  mpz_init(r);
  mpz_set_str(a,"856490000",10);

  for (i=0; i<20; i++) {
    mpz_urandomm(r,state,a);
    mpz_out_str(stdout,10,r);
    putchar('\n');
  }

  mpz_clear(r);
  mpz_clear(a);
  return 0;
}
```

oder ein ähnliches (in Eclipse).

Aufgabe 2. a) Implementiere den naiven Faktorisierungsalgorithmus vom ersten Tag mit GMP und faktorisiere die Zahl

```
272963285971849714829857456457
```

b) Verwende OpenMP um deinen Primzahlalgorithmus zu parallelisieren.

c) Implementiere den

Algorithmus 1 Miller–Rabin Test

Input: Zahlen $a, n \in \mathbb{N}$ mit $a < n$.
Output: "Ja", falls a Zeuge für n, andernfalls "Nein".

1: **if** n gerade oder $\text{ggT}(a, n) \neq 1$ **then**
2: **return** "Ja"
3: **end if**
4: Schreibe $n - 1 = 2^k \cdot q$ mit q ungerade.
5: **set** $a := a^q \bmod n$
6: **if** $a \equiv 1 \pmod{n}$ **then**
7: **return** "Nein"
8: **end if**
9: **for** $i = 1$ **to** k **do**
10: **if** $a \equiv -1 \pmod{n}$ **then**
11: **return** "Nein"
12: **else**
13: **set** $a := a^2 \bmod n$
14: **end if**
15: **end for**
16: **return** "Ja"

und verwende ihn, um eine Funktion zu schreiben, die zu einer gegebenen Bitlänge n eine Primzahl mit mindestens n Bits findet.

d) * Schreibe ein Modul, welches RSA–Verschlüsselung von beliebigen Arrays ermöglicht. Es ist dir überlassen, ob du zum Finden der Primzahlen deine eigene Funktion oder die Funktionen der GMP verwendest.