Programmieraufgabe 3


```
Robert Wettstädt 535161
Sona Pecenakova 540607
```


1. Implementieren Sie einen Algorithmus, der die n-te Potenz a^n einer natürlichen Zahl a in O (log n) Schritten berechnet.

```
    Code: squareandmultiply.c
    zum Vergleich: multiply.c - einfache Multiplikation
```

```
Aufruf:
    ./squaremultiply <a> <n>
```

```
Test:
    2^1 = 2
    2^5 = 32
    2^19 = 524288
    2^25 = 33554432
    3^12 = 531441
    3^30 = 205891132094649
```

2. Modifizieren Sie Ihr Programm aus 1. derart, dass die modulare Potenz a^n mod m berechnet wird.

```
    Code: modularexp.c
```

```
Aufruf:
    ./modularexp <a> <n> <m>
```

```
Test:
    2^5 mod 13 = 6
    2^19 mod 24 = 8
    2^123 mod 35 = 8
    2^123456789 mod 123 = 20
    2^123456789012 mod 123456 = 55552
```

3. Testen Sie Ihre Programme und messen Sie dabei die Laufzeit

Simple multiplication

```
    2^5             | Time: 0.000009 seconds
    2^12               | Time: 0.000010 seconds
    2^123              | Time: 0.000013 seconds
```

```
    2^123456            | Time: 0.319531 seconds
    2^1234567           | Time: 33.291480 seconds
```

Square-and-Multiply

```
    2^5                 | Time: 0.000029 seconds
    2^12                | Time: 0.000034 seconds
    2^123               | Time: 0.000034 seconds
    2^123456            | Time: 0.000696 seconds
    2^123456789         | Time: 2.209452 seconds
```

Modular exponentiation

```
    2^5 mod 13          | Time: 0.000028 seconds
    2^19 mod 24         | Time: 0.000031 seconds
    2^123 mod 35        | Time: 0.000033 seconds
    2^123456789 mod 123 | Time: 0.000043 seconds
```

- Man kann merken, dass der "Simple Multiplication" Algorithmus, der die Zahl einfach n-mal multipliziert schneller ist, bei kleineren Zahlen. Aber je hoeher die Zahlen sind, wird es wesentlich langsamer als der Square-and-Multiply Algorithmus.
- Modular exponentiation ist viel schneller als Square-and-Multiply, dadurch dass alle Werte waehrend der Berechnung bei modulo reduziert werden.

4. Wie lange dauert die Berechnung von an mod m in 2., wenn a, n und m 1000-bit-Zahlen sind?

```
    Code: modularexp1000.c
```

```
Aufruf:
    ./modularexp1000
```

```
    Time: 0.004307 seconds
    Time: 0.004021 seconds
    Time: 0.004608 seconds
```

- Die Zahlen werden mit einem Random generiert, so dass die Bit-Anzahl 1000-bit ist.
- Die Zeit ist immer ungefähr 4.5 Millisekunden, auch bei sehr großen Zahlen.