MATHEMATICAL ALGORITHMS I

1. Problem sheet 1

(1) "Implement" means that you should write a `C` or `C++` program (other languages are also okay but we won't provide any detailed technical help)
(2) "Design an algorithm" means that you should provide pseudocode (as done in class)
(3) At the very least, a program should *compile*, and ideally work correctly.
(4) For this first problem sheet, use only `stdlib.h`, `stdio.h` (for C) or `iostream` and/or `vector`.

The goal is to work with these notions yourself, so you should not use specialized libraries such as `gmp`, which provide all functionality regarding multiprecision for example. Such libraries are often very good, so when you need the relevant algorithms in real life, it is good to use these libraries after you make sure you understand them. In fact, your own implementations are likely to be slower. However, I repeat: in this course the actual work should be done by yourself, not by such an external library.

1.1. **Problem:** Implement Euclid algorithm for finding the greatest common divisor of two integers. As a reminder, you can and should use the standard operations `*, /, %`.

(1) input: two `int`'s m and n
(2) output: an `int` representing the greatest common divisor.

This serves as a warmup.

1.2. **Problem:** Implement addition and classical multiplication of polynomials via array (say a vector in a finite-dimensional vector space).

Print the results in latex-code or another human-readable format.

Hints:

- if you don't know how to make an array look below.
- you *don't* need to write a parser that reads a polynomial and converts it to an array. Instead, just enter the polynomial coefficient by coefficient as indicated below.

  You only need write a function that takes arrays and returns an array containing the sum and product.
- Here are a couple of ways to allocate memory for an array:

```
const int order=10;
int *v=(int *)malloc(sizeof(int)*order);
//alternative use
//int *v = (int*) calloc (order,sizeof(int));
//this clears the array as well.
v[0]=1; //sets first entry to 1
v[order-1]=1; //sets last entry to 1:
//other entries still undefined if malloc is used.
```

```
//do stuff
free(v);
```

Here is the classical `C++`-way.

```
const int order=10;
int *v=new int[order];

//do stuff
delete[] v;
```

More modern and safer is the use of vector (slightly slower than naked pointers)

```
const int order=10;
std::vector<int> v;
v.resize(order); //for fixed

//do stuff
```

1.3. **Problem:** Implement addition of multiprecision integers with equal signs via an array. This problem just asks you to implement the primary school algorithm. Hint: write down precisely what this means, and take care of the carry.

1.4. **More operations:**

(1) Design and implement an algorithm that takes as input two multiprecision integers a and b, and returns `true` if $a \geq b$ and `false` if $a < b$.
(2) Design and implement an algorithm for the subtraction of two integers: first work out convenient cases, such as $a \geq b$, and equal sign cases. The input are two multiprecision integers a, b and the output should consist of $c = a - b$.
(3) Implement classical multiplication of multiprecision integers.
(4) Design and implement division with remainder for multiprcision
(5) Design and implement an algorithm to print a multiprecision integer in decimal representation.
    (a) Input: a multiprecision integer a
    (b) Output: screen output of the decimal representation of a.

1.5. **Problem: Karatsuba multiplication.**

(1) Implement Karatsuba multiplication of polynomials via an array.
(2) Compare the speed to that of maple or another computer algebra package (for example sympy in python). Can you optimize by using less dynamical memory allocation?