Mathematics 360 **Homework (due Oct 23)** A. Hulpke

35) a) Consider two numbers a, b given in binary form, each with n bits. Let $f(n)$ be the cost of calculating $a+b$ in binary form (using the method you learned in school). Determine a function g such that $f(n) = \mathcal{O}(g(n))$.
b) Solve the corresponding problem for calculating $a \cdot b$.

36) Let $p > 2$ be a prime.
a) Show that there is exactly one element a such that $\mathrm{OrderMod}_p(a) = 2$, namely $r^{\frac{p-1}{2}}$, where r is a primitive root. (**Hint:** Consider the orders of r^e, using problem 19.)
b) More generally, show that there are exactly $\varphi(k)$ numbers in $1, \ldots, p-1$ that have order k.

37) Let p be a prime and $p-1 = a \cdot b$ with $\gcd(a,b) = 1$ and g be a primitive root modulo p.
a) We let $x \equiv g^a \pmod{p}$, $y \equiv g^b \pmod{p}$. Determine $\mathrm{OrderMod}_p(x)$ and $\mathrm{OrderMod}_p(y)$.
b) Show that for $1 \le e \le \mathrm{OrderMod}_p(x)$ and $1 \le f \le \mathrm{OrderMod}_p(y)$ we have that $x^e \not\equiv y^f \pmod{p}$ unless $x^e \equiv 1 \equiv y^f \pmod{p}$. (**Hint:** Consider the orders of the elements.)
c) Show that for any $1 \le z \le p-1$ there exist exponents e, f such that $z \equiv x^e \cdot y^f \pmod{p}$.(**Hint:** Use $\gcd(a,b) = 1$)
d) Let $p = 1031$, $a = 10$ and $b = 103$ and $g = 14$. Then $g^a \equiv 320 \pmod{p}$ and $g^b \equiv 513 \pmod{p}$. You are also given the information that $2 \equiv 14^{796}$. (You can assume all of this as fact without proof.) Determine e, f such that $2 \equiv 320^e \cdot 513^f \pmod{p}$.

38) Show that $n! \neq \mathcal{O}(n^k)$ for any k, but $n! = \mathcal{O}(2^n)$. (I.e. an algorithm running through all permutations is exponential time, not polynomial time.)

39) Imagine that in the Diffie-Hellman key exchange Alice has chosen an exponent a such that $\gcd(p-1, a)$ is very large. Why is this problematic?

40) Future homework will involve calculations that can be repetitive and very time-consuming using just a basic calculator, but require the use of a computer algebra system which provides

- Arbitrary size long integer arithmetic
- Some basic number-theoretic functionality, such as the fast powering algorithm modulo m or $\mathrm{OrderMod}_p(a)$.
- The facilty to perform an operation over a list of objects or search for list elements with particular properties.

If you like to use a system you know already, you are welcome to do so, I will provide information for the system GAP, see below. Other suitable choices would be for example Maple, Mathematica, Sage, or even Python with suitable modules for long integers and for number theory. (Please be

aware that Matlab, Mathcad or similar systems are unsuitable as they use floating point arithmetic for larger numbers and thus do not represent them exactly.)
Get familiar with one such system.

Task (which takes seconds on the computer and days with a calculator): For all primes $p \in \{1000, \ldots, 9999\}$ determine the maximal order of 2 modulo p.
Hint: In GAP you could do this by the command:

```
Maximum(List(Filtered([1000..9999],IsPrime),p->OrderMod(2,p)));
```

where `Filtered([1000..9999],IsPrime)` forms a list of all primes in the range, `List(listofprimes,p->OrderMod(2,p))` then determines the order of 2 modulo each of these primes, and `Maximum(list)` finally takes the maximum of this list.

Practice Problems: **2.3**, **2.4**, **2.6**, **2.16**,

Introduction to GAP

GAP is a free and open computer algebra system for discrete mathematics. It started under Unix (and still shows its heritage) but now also has versions for Windows.
You can run it on the machines in the mathematics lab (Weber 205):

> Username: m360
> DOMAIN: MATHSTAT3
> Password: (Given in class. I'm not allowed to write it down) _____

However you likely will want to be able to run it on your own machine. There are installers (Don't worry that this is not the newest version) with a nicer shell for Windows and OSX under `http://www.math.colostate.edu/~hulpke/CGT/education.html`
Alternatively (Unix, OSX if you have a C-compiler installed) you can download the source from `http://www.gap-system.org`

Once the program starts you will get a text prompt, looking like this:

```
gap>
```

You now can type in commands (followed by a semicolon) and GAP will return the result.
To leave GAP you can either call `quit;` or type `<ctl>-d`
The editing commands are similar as in the EMACS editor (depending on the setup, also cursor keys might work):

`<ctl>-p` gets the previous line. (If the cursor is not at the start of the line, it gets the last line that starts like the current one.)

`<ctl>-n` gets the next line.

`<ctl>-f` goes one character to the right.

`<ctl>-b` goes one character to the left.

`<ctl>-a` goes to the start of the line.

`<ctl>-e` goes to the end of the line.

`<ctl>-d` deletes the character under the cursor (insert is automatic).

`<ctl>-k` deletes the rest of the line.

GAP knows integers of arbitrary length and rational numbers:

```
gap> -3; 17 - 23;
-3
-6
gap> 2^200-1;
1606938044258990275541962092341162602522202993782792835301375
gap> 123456/7891011+1;
2671489/2630337
```

GAP knows a precedence between operators that may be overridden by parentheses and can compare objects:

```
gap> (9 - 7) * 5 = 9 - 7  * 5;
false
gap> 5/3<2;
true
gap> 5/3>=2;
false
```

You can assign numbers (or more general: every GAP object to variables, by using the assignment operator `:=`. Once a variable is assigned to, you can refer to it as if it was a number. The special variables `last`, `last2`, and `last3` contain the results of the last three commands.

```
gap> a:=2^16-1;
65535
gap> b:=a/(2^4+1);
3855
gap> 5*b-3*a;
-177330
gap> last+5;
-177325
gap> last+2;
-177323
```

The following commands are useful for number theoretic calculations:

```
gap> Int(8/3); # round down
2
gap> QuoInt(76,23); # integral part of quotient
3
gap> 76 mod 23; # remainder (note the blanks)
7
gap> EvalF(76/23); # numeric approximation
"3.3043478260"
gap> IsPrime(6);   #primality test
false
gap> IsPrime(73);
true
gap> NextPrimeInt(73); # next bigger prime
79
gap> 17/2 > 9;
false
gap> 17/2 <= 9;
true
gap> Gcd(15,72);
3
gap> GcdRepresentation(15,72);
[ 5, -1 ]
gap> 5*15-1*72;
3
gap> RootInt(1000,2); # rounded root
31
gap> PowerMod(5,255,65537); # fast powering: 5^255 mod 655
64050
```

Try the following example (testing Fermat's theorem) to see how much faster `PowerMod` is than calculating a huge power first:

```
gap> p:=NextPrimeInt(10^8);
100000007
gap> PowerMod(5,p-1,p);
1
gap> 5^(p-1) mod p; # calculates huge number first, slow
1
```

There are also special commands for $\mathrm{OrderMod}_p(a)$, finding a primitive root, φ and discrete logarithm as well as a (moderately powerful) factorization.

```
gap> OrderMod(2,p);
50000003
gap> PrimitiveRootMod(p);
5
gap> Phi(768);
256
gap> LogMod(17,5,p);
41658447
gap> PowerMod(5,last,p);
17
gap> Factors(2^67-1);
[ 193707721, 761838257287 ]
```

By enclosing objects with square brackets, and separating them by commas, you can create a list.
Collections of numbers (or other objects) are represented by such lists. Lists are also used to represent sets.

```
gap> l:=[5,3,99,17,2]; # create a list
[ 5, 3, 99, 17, 2 ]
gap> l[4]; # access to list entry
17
gap> l[3]:=22; # assignment to list entry
22
gap> l;
[ 5, 3, 22, 17, 2 ]
```

```
gap> Length(l);
5
gap> 3 in l; # element test
true
gap> 4 in l;
false
gap> Position(l,2);
5
gap> Add(l,17); # extension of list at end
gap> l;
[ 5, 3, 22, 17, 2, 17 ]
gap> s:=Set(l); # new list, sorted, duplicate free
[ 2, 3, 5, 17, 22 ]
gap> l;
[ 5, 3, 22, 17, 2, 17 ]
gap> AddSet(s,4); # insert in sorted position
gap> AddSet(s,5); # and avoid duplicates
gap> s;
[ 2, 3, 4, 5, 17, 22 ]
```

Results that consist of several numbers are represented as list.

```
gap> DivisorsInt(96);
[ 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96 ]
gap> Factors(2^127-1);
[ 170141183460469231731687303715884105727 ]
gap> Factors(2^126-1);
[ 3, 3, 3, 7, 7, 19, 43, 73, 127, 337, 5419,
   92737, 649657, 77158673929 ]
```

There are powerful list functions that often can save programming loops: `List`, `Filtered`, `ForAll`, `ForAny`, `First`. The notation `i -> xyz` is a shorthand for a one parameter function.

```
gap> l:=[5,3,99,17,2];
[ 5, 3, 99, 17, 2 ]
gap> List(l,IsPrime);
[ true, true, false, true, true ]
gap> List(l,i -> i^2);
[ 25, 9, 9801, 289, 4 ]
gap> Filtered(l,IsPrime);
[ 5, 3, 17, 2 ]
gap> ForAll(l,i -> i>10);
false
gap> ForAny(l,i -> i>10);
true
gap> First(l,i -> i>10);
99
```

You can use the online help to get documentation

```
gap> ?List
Help: Showing 'Reference: List'
> List( <list> ) ...
```

(Use `?Line Editing` to get a list of edit commands.)
A special case of lists are *ranges*, indicated by double dots. They can also be used to create arithmetic progressions:

```
gap> l:=[10..100];
[ 10 .. 100 ]
gap> Length(l);
91
gap> First(l,IsPrime);
11
gap> l2:=[3,7..99];
[ 3, 7 .. 99 ]
gap> Length(l2);
25
gap> Filtered(l2,IsPrime);
[ 3, 7, 11, 19, 23, 31, 43, 47, 59, 67, 71, 79, 83 ]
gap> Filtered(l2,i-> not IsPrime(i));
[ 15, 27, 35, 39, 51, 55, 63, 75, 87, 91, 95, 99 ]
```

For a more complex example, we test whether numbers that can be represented as the sum of two squares.

```
gap> max:=1000;
1000
gap> lim:=RootInt(max,2)+1; # rounded up root
32
gap> cands:=[0..lim]; # candidate numbers to be squared
[ 0 .. 32 ]
gap> ForAny(cands,x->ForAny(cands,y->x^2+y^2= 19));
false
gap> ForAny(cands,x->ForAny(cands,y->x^2+y^2= 20));
true
```

Here, we test whether there is any combination of squares that sums up to the number we want to test. (This is far from optimal. Can you think of a better test?)
We now can combine this with a `Filtered` command to test all numbers up to 1000 that have this property. (We look for those n, such that an x exists, and that an y exists such that $x^2 + y^2 = n$.)

```
gap> Filtered([1..max],
> n -> ForAny(cands,x->ForAny(cands,y->x^2+y^2=n)) );
[ 1, 2, 4, 5, 8, 9, 10, 13, 16, 17, 18, 20, 25, 26, 29, 32
```