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 \(\text{OrderMod}_p(a) = 2\), namely \(r^{p-1} \mod p\), 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 \(\text{OrderMod}_p(x)\) and \(\text{OrderMod}_p(y)\).

b) Show that for \(1 \leq e \leq \text{OrderMod}_p(x)\) and \(1 \leq f \leq \text{OrderMod}_p(y)\) we have that \(x^e \not\equiv y^f \pmod{p}\) unless \(x^e \equiv 1 \pmod{p}\). (Hint: Consider the orders of the elements.)

c) Show that for any \(1 \leq z \leq p - 1\) 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 \(2 \equiv 14^{796} \pmod{p}\). (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 \(n! \not\in \mathcal{O}(n^k)\) 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 \(\text{OrderMod}_p(a)\).
- The facility 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:

\[
\text{Maximum} \left( \text{List} \left( \text{Filtered} \left( \left[ 1000 \ldots 9999 \right], \text{IsPrime} \right), p \to \text{OrderMod} (2, p) \right) \right);
\]

where \( \text{Filtered} \left( \left[ 1000 \ldots 9999 \right], \text{IsPrime} \right) \) forms a list of all primes in the range, \( \text{List} \left( \text{listofprimes}, p \to \text{OrderMod} (2, p) \right) \) then determines the order of 2 modulo each of these primes, and \( \text{Maximum} \left( \text{list} \right) \) 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 \(<\text{ctl}-d>\)
The editing commands are similar as in the EMACS editor (depending on the setup, also cursor keys might work):

\(<\text{ctl}-p>\) 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.)

\(<\text{ctl}-n>\) gets the next line.

\(<\text{ctl}-f>\) goes one character to the right.

\(<\text{ctl}-b>\) goes one character to the left.

\(<\text{ctl}-a>\) goes to the start of the line.

\(<\text{ctl}-e>\) goes to the end of the line.

\(<\text{ctl}-d>\) deletes the character under the cursor (insert is automatic).

\(<\text{ctl}-k>\) deletes the rest of the line.

GAP knows integers of arbitrary length and rational numbers:
GAP knows a precedence between operators that may be overridden by parentheses and can compare objects:

```
gap> (9 - 7) * 5 = 9 - 7 * 5;
gap> false

gap> 5/3<2;
gap> true

gap> 5/3>=2;
gap> 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
```

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 OrderMod\(_p\)(a), finding a primitive root, \(\varphi\) 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> 1:=[5,3,99,17,2]; # create a list
[ 5, 3, 99, 17, 2 ]

gap> 1[4]; # access to list entry
17

gap> 1[3]:=22; # assignment to list entry
22

gap> 1;
[ 5, 3, 22, 17, 2 ]
```
You can use the online help to get documentation

```gap
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
gap> 1:=[10..100];
[ 10 .. 100 ]
gap> Length(1);
91
```

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

```gap
gap> l:=[5,3,99,17,2];
[ 5, 3, 99, 17, 2 ]
gap> List(l,IsPrime);
true, true, false, true, true
```

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

```gap
gap> max:=1000;
1000
gap> lim:=RootInt(max,2)+1; # rounded up root
32
```

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
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, ... ]
```

Results that consist of several numbers are represented as list.

```gap
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);
```

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

```gap
gap> l:=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'];
You can use the online help to get documentation