GAP Exercises

Version 1.0

February 17, 2010

Stefan Kohl

Stefan Kohl Email: `stefan@mcs.st-and.ac.uk`
Homepage: `http://www.gap-system.org/DevelopersPages/StefanKohl/`
Address: Department of Mathematics
University of Vlora
Lagjja: Pavaresia
Vlore / Albania

Contents

Chapter 1

The Exercises

1.1 On Commutators and Derived Subgroups

1.1.1 Exercise

The *derived subgroup* G' of a group G is defined as the group *generated* by all *commutators* $[a,b] := a^{-1}b^{-1}ab$, where $a,b \in G$. In general, not all elements of the derived subgroup of a group are actually commutators themselves. Find a group G of smallest possible order such that the set of all commutators of elements of G does not *form* a group!

1.1.2 Hints

Commutators can be computed by the operation `Comm`, and the derived subgroup can be computed by the operation `DerivedSubgroup`. The function `Cartesian` and the operations `Set` and `AsList` can be helpful in this context as well. If you are lazy but patient, you may simply use `CommutatorLength`. Otherwise, this exercise requires writing a couple of lines of GAP code. The wanted group has order greater than 50, but less than 100.

For a solution, see Section 2.1.

1.2 On Outer Automorphisms Fixing Conjugacy Classes

1.2.1 Exercise

Is there a finite group which has an automorphism which is not inner, but which nevertheless fixes all conjugacy classes setwise? – If so, then find an example of least possible order!

1.2.2 Hints

This exercise is slightly more difficult than that given in Section 1.1. Useful functions and operations are `AllGroups`, `AutomorphismGroup`, `IsInnerAutomorphism`, `IsConjugate`, `Image` and `AsList`. It is a good exercise to try to find a way to write down the group and the automorphism in a nice and human-readable form. One possible way to achieve this is to determine a “nice” permutation representation of the group, and to choose an automorphism with the desired property which fixes all but one of the generators.

For a solution, see Section 2.2.

1.3 Drawing the Ulam Spiral

1.3.1 Exercise

Write a GAP function which draws the Ulam spiral and saves the picture in a file!

The arguments of the function should be the size (width / height) of the picture to be drawn and the name of the output file.

1.3.2 Hints

This is more or less just an easy programming exercise, which does not require particular mathematical knowledge. Use the function `NullMat` to create a zero matrix over the field with two elements, and use this matrix as a grid to draw the spiral. The RCWA package [Koh07b] provides a function `SaveAsBitmapPicture`, which can be used to produce a picture file from the matrix.

For a solution, see Section 2.3.

1.4 Automorphism Group of the Smallest Projective Plane

1.4.1 Exercise

The automorphisms of a finite projective plane are determined by the permutations of the set of points which move lines to lines. Thus, labelling the points with integers $1,2,\dots,n$, the automorphism group can be described as a subgroup of the symmetric group S_n.

The smallest projective plane has 7 points and 7 lines. Any point is incident with 3 lines, and any line is incident with 3 points. We label the points with integers $1,2,\dots,7$. Then the lines are given by the sets $g_1 := \{1,2,3\}$, $g_2 := \{1,4,7\}$, $g_3 := \{1,5,6\}$, $g_4 := \{2,4,6\}$, $g_5 := \{2,5,7\}$, $g_6 := \{3,4,5\}$ and $g_7 := \{3,6,7\}$ of points which are incident with them.

Compute the automorphism group of the smallest projective plane!

1.4.2 Hints

A useful function in this context is `SubgroupProperty`. The actual determination of the group can be done in one statement. Can you figure out the isomorphism type of the group by theoretical means?

For a solution, see Section 2.4.

1.5 Installing a Missing Method

1.5.1 Exercise

What happens when you enter the command `Centre(AlternatingGroup(100));` in GAP? Are you satisfied with the performance, given that the centre of a nonabelian simple group is always trivial and that GAP knows that the alternating group of degree 100 is simple?

Probably not. – Thus improve the performance radically by implementing a method for `Centre` for simple groups, which returns either the group itself or the trivial subgroup, depending on whether the group is abelian or not!

1.5.2 Hints

Methods are installed with `InstallMethod`. The exercise is easy – basically all you need to do is to look up in the documentation how this function is used. If your solution is correct, then `Centre(AlternatingGroup(100));` returns the trivial subgroup immediately, and `Centre(G)` still computes the centre of a non-simple group `G` using methods implemented in the GAP Library.

For a solution, see Section 2.5.

1.6 Finding Good abc Triples

1.6.1 Exercise

Given a positive integer n, let $\mathrm{rad}(n)$ denote the product of distinct prime divisors of n. The abc conjecture states that for any $\varepsilon > 0$ there is a constant K_ε such that for any triple (a,b,c) of coprime positive integers satisfying the equation $a+b=c$ we have $c < K_\varepsilon \mathrm{rad}(abc)^{1+\varepsilon}$.

A triple (a,b,c) of coprime integers satisfying $a+b=c$ is called an *abc triple* if $\mathrm{rad}(abc)$ is less than c. An abc triple (a,b,c) is called a *good* abc triple if it satisfies even $\ln(c)/\ln(\mathrm{rad}(abc)) > 1.4$. The left-hand side of the inequality is sometimes called the *ratio* of the abc triple.

It can be shown easily that there are infinitely many abc triples, but if the abc conjecture holds, there are only finitely many good abc triples.

Write a GAP function which finds all good abc triples (a,b,c) with given radical $\mathrm{rad}(abc)$ and with c less than a given bound!

Can you find a new triple, which is not yet on Abderrahmane Nitaj's list of known good abc triples?

1.6.2 Hints

You can start by determining all positive integers less than the given bound all of whose prime factors divide the given radical. This can be done much more efficiently than by the Sieve of Eratosthenes, or even by looping over all integers in the range and factoring – it is easy and very elementary to find out how. Then you can loop over all pairs of these integers, test whether they are coprime and compute the ratio if they are. To compute the ratio, you need the operation `Float` which converts integers and rationals to floating point numbers, and the function `LOG_FLOAT` which computes the natural logarithm of a floating point number.

For a solution, see Section 2.6.

1.7 Automorphism Groups of Finite Graphs

1.7.1 Exercise

a) Determine all undirected graphs with 6 vertices up to isomorphism! – How many isomorphism types of such graphs are there?

The graphs should be represented as sets of edges, where the edges should be written as sets of two vertices, each.

Example: In this representation, the graphs `[[1,2],[1,6],[2,3],[3,4],[4,5],[5,6]]` and `[[1,5],[1,6],[2,4],[2,6],[3,4],[3,5]]` are both isomorphic to the regular hexagon.

b) Write a function `GraphAutomorphismGroup(`*`Gamma`*`,`*`n`*`)` which computes the automorphism group of the graph *`Gamma`* with *`n`* vertices. – The automorphisms are precisely the permutations of the set of vertices which move edges to edges.

Note that the cardinality *`n`* of the set of vertices needs to be specified, as there may be isolated vertices.

c) Find out which of the (up to conjugation in S_6) 16 transitive permutation groups of degree 6 occur as automorphism groups of graphs with 6 vertices! – How do the corresponding graphs look like?

1.7.2 Hints

ad a)

You can obtain the set of all graphs with n vertices in the suggested notation by `Combinations(Combinations([1..n],2));`. Then you need to find a suitable group action on this set such that two graphs are isomorphic if and only if they lie in the same orbit. Useful operations are `Orbits` and `Representative`.

ad b)

In principle you can implement a fancy algorithm here, but for our purposes a very basic one is perfectly sufficient. First find out how to check whether a given permutation of the vertices induces a graph automorphism. Then you can use `SubgroupProperty` to determine the group formed by all such permutations.

ad c)

Given Part a) and b), this is more or less straightforward. Useful functions / operations are `AllTransitiveGroups`, `NrMovedPoints` and `TransitiveIdentification`.

For a solution, see Section 2.7.

1.8 Enumerating Paths

1.8.1 Exercise

Answer the following questions using GAP:

a) In how many ways can $1 \in S_4$ be written as a product of exactly 100 transpositions?

b) In how many ways can a horse cross the chess board from the upper left to the lower right corner with exactly 100 moves?

1.8.2 Hints

Construct a matrix $x \in \mathbb{Z}^{n \times n}$ with ones at suitable positions and zeros everywhere else and compute powers. For Part a), choose $n := 24$, and for Part b), choose $n := 8^2 = 64$. Useful functions are e.g. `NullMat`, `SymmetricGroup`, `Combinations`, `Cartesian` and `Position`.

For a solution, see Section 2.8.

1.9 Wieferich Primes

1.9.1 Exercise

By Fermat's little theorem, for any prime number p we have $p|2^{p-1}-1$. A prime number p is called a Wieferich prime if it satisfies even $p^2|2^{p-1}-1$.

Write a GAP function which checks whether a given positive integer is a Wieferich prime, and try to find as many Wieferich primes as you can!

1.9.2 Hints

Writing the GAP function is easy. Use `PowerModInt` instead of first computing 2^{p-1} and then reducing modulo p^2. Two Wieferich primes can be found easily, but finding a third one is at least hard.

For a solution, see Section 2.9.

1.10 Counting Words in a File

1.10.1 Exercise

Write a GAP function which, given a filename, returns a word distribution statistics. The function should return a list with entries of the form `[ "word", 237 ]` for any word in the file, indicating the word and the number of times it occurs. A "word" in our sense is a sequence of letters with no non-letter characters in between, i.e. it does not need to be a dictionary word. Can you put your function into one line with no more than 80 characters?

1.10.2 Hints

You will probably need the operation `Collected`. Further, the functions `StringFile` and `WordsString` from the GAPDoc package [LN06] will be useful.

For a solution, see Section 2.10.

1.11 Non-Metabelian p-Groups

1.11.1 Exercise

A group G is called *metabelian* if it has an abelian normal subgroup N such that the quotient G/N is abelian as well. Further, a group is called a *p-group* if its order is a power of a prime.

Find a non-metabelian p-group of least possible order!

1.11.2 Hints

First determine conceivable orders of non-metabelian groups by means of theory. Then just run a *brute-force* search over the groups of the smallest few of these orders in the Small Groups Library [BEO07]. Useful functions / operations are `AllGroups`, `DerivedSubgroup` and `IsAbelian`. For a solution, see Section 2.11.

1.12 The Growth of the Sum-of-Divisors Function

1.12.1 Exercise

Let σ denote the sum-of-divisors function. Given a positive integer n, let $H(n) := \sum_{k=1}^{n} 1/k$ be the nth harmonic number, and put $B(n) := H(n) + \ln(H(n)) \cdot e^{H(n)}$.

Examples are $\sigma(24) = 60$ and $B(24) \approx 61.7575$, as well as $\sigma(60) = 168$ and $B(60) \approx 170.977$.

Can you find an integer $n > 1$ such that $\sigma(n)$ is larger than $B(n)$?

1.12.2 Hints

In GAP, $\sigma(n)$ can be computed by `Sigma(n)`. Use the operation `Float` to convert integers and rationals to floating point numbers, and use the functions `LOG_FLOAT` and `EXP_FLOAT` to compute logarithms and to evaluate the exponential function, respectively. Be prepared that finding an integer $n > 1$ such that $\sigma(n)$ is larger than $B(n)$ is not easy.

For a solution, see Section 2.12.

1.13 Pell's Equation

1.13.1 Exercise

Pell's equation is any Diophantine equation of the form $x^2 - ny^2 = 1$, where n is a nonsquare integer.

Write a GAP function which takes an argument n and returns the smallest solution of the equation $x^2 - ny^2 = 1$ in positive integers!

This solution is called the *fundamental solution*. Your function should return for example the answer for $n = 421$ very quickly.

1.13.2 Hints

Useful functions in this context are `ContinuedFractionExpansionOfRoot` and `ContinuedFractionApproximationOfRoot`.

For a solution, see Section 2.13.

1.14 Automorphism Groups of Odd Order

1.14.1 Exercise

Obviously, the automorphism groups of both the trivial group and the cyclic group of order 2 are trivial, and have therefore odd order. Find the smallest group of order greater than 2 whose automorphism group has odd order!

1.14.2 Hints

First try to exclude the groups of even order by means of theory. Then run a *brute-force* search over the Small Groups Library [BEO07]. Useful functions / operations in this context are `AllGroups` and `AutomorphismGroup`.

For a solution, see Section 2.14.

1.15 Composite Sums

1.15.1 Exercise

Find an odd positive integer n such that $n+2^k$ is composite for any positive integer k! – Can you find the least possible such n?

1.15.2 Hints

First perform a *brute-force* search to find candidates. Then try to verify whether these candidates indeed have the desired property. A useful function in this context is `OrderMod`. In addition, it may be worth to have a look at the ResClasses package [Koh07c], and there in particular at the function `ResidueClass`. Finding a good candidate for the least possible number with the given property is reasonably easy, but the verification that it is indeed the smallest one is computationally difficult.

For a solution, see Section 2.15.

1.16 Rational Points on the Unit Sphere

1.16.1 Exercise

Write a GAP function which computes all rational points on the unit sphere $x^2+y^2+z^2=1$ which correspond to solutions of the diophantine equation $a^2+b^2+c^2=d^2$ with a, b and c not exceeding a given bound. Further, your function should draw a picture showing the projection of one octant of the sphere to the x-y-plane, where the rational points are marked by black pixels.

1.16.2 Hints

Just write a nested loop to determine the solutions. Note that the variables can be permuted, thus you can assume $a \geq b \geq c$ and generate the solutions not satisfying this inequality by permuting a, b and c. This saves almost 5/6 of the time. Maybe a useful function in this context is `Arrangements`. Create an empty grid over GF(2) by the function `NullMat`, invert it (i.e. replace zeros by ones) and mark the solutions by zeros there. Finally, the RCWA package [Koh07b] provides a function `SaveAsBitmapPicture`, which can be used to write the picture to a file.

For a solution, see Section 2.16.

1.17 Aliquot Sequences

1.17.1 Exercise

Given a positive integer n, the *Aliquot sequence* $n=a_1,a_2,a_3,a_4,\ldots$ starting at n is defined by $a_{i+1}=\sigma(a_i)-a_i$, where σ denotes the sum-of-divisors function. We say that the Aliquot sequence starting at n *stops* if there is an index i such that $a_i=1$, and we say that it *runs into a cycle* if there are distinct indices i and j such that $a_i=a_j$.

Find out whether all Aliquot sequences starting at integers $n<100$ either stop or run into cycles! – Can you do the same for all Aliquot sequences starting at integers $n<200$ or $n<300$? Do you see algorithmic problems, and of which kind are they?

1.17.2 Hints

In GAP, $\sigma(n)$ can be computed by `Sigma(n)`. Computing $\sigma(n)$ requires factoring n. For this, the GAP Library method for the operation `Sigma` calls the GAP Library function `FactorsInt` directly. This works for small n, but for larger n, `FactorsInt` will often give up and raise an error message which suggests to use the FactInt package [Koh07a]. If you have loaded FactInt, you may find this strange. However, this has nothing to do with FactInt, as this package does not get a chance to help with factoring. You can make `Sigma` benefit from FactInt if you fetch the method for `Sigma` from `lib/numtheor.gi`, put it into a separate file, replace `FactorsInt` by `Factors`, increase the method rank to something like `SUM_FLAGS` and read this file into GAP. Alternatively you can make the change directly in the GAP Library. Then you do not need to increase the method rank, but (as after every Library change) you need to rebuild the completion files. For this, start GAP with option -N and enter `CreateCompletionFiles();`.

For a solution, see Section 2.17.

1.18 The Q Sequence

1.18.1 Exercise

Hofstadter's Q sequence is defined by $Q_1 = Q_2 = 1$ and $Q_n = Q_{n-Q_{n-1}} + Q_{n-Q_{n-2}}$ for $n > 2$.

a) Write a GAP function which takes an integer argument l and computes the first l terms of the Q sequence.

b) Write a GAP function which plots the graph of the Q sequence.

1.18.2 Hints

ad a)

The Q sequence is defined recursively. Ask yourself the question why a recursive implementation is not a particularly good idea in this case, anyway.

ad b)

Use the function `NullMat` to create a zero matrix over the field with two elements, turn the zeros into ones if you prefer a black graph on a white background to a white graph on a black background, and use this matrix as a grid to draw the graph. The RCWA package [Koh07b] provides a function `SaveAsBitmapPicture`, which can be used to produce a picture file from the matrix.

For a solution, see Section 2.18.

1.19 A Quickly Growing Function

1.19.1 Exercise

Have a look at the following function, which takes as arguments three nonnegative integers and which returns a positive integer:

GAP code

```
f := function ( i, j, k )
  if i = 0 then
    if   j = 0
    then if k = 0 then return 2; else return 2^f(i,j,k-1); fi;
    else return f(i,j-1,f(i,j-1,k)); fi;
  else
    return f(i-1,f(i-1,j,k),f(i-1,j,k));
  fi;
end;
```

Try to evaluate `f` for small values of `i`, `j` and `k`! – How far can you get? Can you evaluate `f(1,1,1)` or `f(2,2,2)`, or can you perhaps write down these values as non-recursive expressions?

The function `f` is still a computable function – recall however that there are functions which grow faster than *any* computable function!

1.19.2 Hints

Some values this function takes: `f(0,0,0)` is 2, `f(0,0,1)` is 4, `f(0,0,2)` and `f(0,1,0)` are both 16, `f(0,0,3)` is 65536, `f(0,0,4)` and `f(0,1,1)` are both 2^{65536}, `f(0,0,5)` is $2^{2^{65536}}$, and already `f(1,1,1)` is basically too large to be written down in a non-recursive way. The exercise asks just for some experimentation – thus there is no solution given.

1.20 The 3n+1 Conjecture

1.20.1 Exercise

The $3n+1$ *conjecture*, also known as *Collatz conjecture*, asserts that iterated application of the *Collatz mapping*

$$T: \ \mathbb{Z} \longrightarrow \mathbb{Z}, \quad n \longmapsto \begin{cases} \frac{n}{2} & \text{if } n \text{ is even,} \\ \frac{3n+1}{2} & \text{if } n \text{ is odd} \end{cases}$$

to any given positive integer eventually yields 1. This problem has been posed by Lothar Collatz in the 1930's, and it is still open today.

Investigate Collatz' conjecture by means of computation with GAP, and try to find a proof or a counterexample!

1.20.2 Hints

The $3n+1$ conjecture is generally believed to be true, but if it is false, this may be for two reasons: firstly, there may be unbounded sequences, and secondly, there may be sequences which run into cycles not containing 1. Jeffrey C. Lagarias has compiled a comprehensive annotated bibliography [Lag07] on this conjecture. The GAP package RCWA [Koh07b] provides a large variety of methods to compute with mappings like the Collatz mapping. We show just how to enter the Collatz mapping and how to compute the sequences we are interested in:

GAP session log
```
gap> T := RcwaMapping([[1,0,2],[3,1,2]]);
<rcwa mapping of Z with modulus 2>
gap> SetName(T,"T");
gap> Display(T);

Rcwa mapping of Z with modulus 2

              n mod 2               |                n^T
------------------------------------+------------------------------------
  0                                 | n/2
  1                                 | (3n + 1)/2

gap> Trajectory(T,15,[1]);
[ 15, 23, 35, 53, 80, 40, 20, 10, 5, 8, 4, 2, 1 ]
gap> Trajectory(T,27,[1]);
[ 27, 41, 62, 31, 47, 71, 107, 161, 242, 121, 182, 91, 137, 206, 103,
  155, 233, 350, 175, 263, 395, 593, 890, 445, 668, 334, 167, 251, 377,
  566, 283, 425, 638, 319, 479, 719, 1079, 1619, 2429, 3644, 1822, 911,
  1367, 2051, 3077, 4616, 2308, 1154, 577, 866, 433, 650, 325, 488, 244,
  122, 61, 92, 46, 23, 35, 53, 80, 40, 20, 10, 5, 8, 4, 2, 1 ]
```

There is (of course!) no solution given for this exercise.

Chapter 2

The Solutions

2.1 On Commutators and Derived Subgroups (Solution)

This is a suggested solution for Exercise 1.1.

There is a function `CommutatorLength(G)`, which returns the minimal number *n* such that each element in the derived subgroup of the group `G` can be written as a product of (at most) *n* commutators of elements of `G`. For the groups we are looking for, this function returns a value greater than 1. However, `CommutatorLength` is reasonably slow and we need to check many groups. Therefore we write an own little function as follows:

GAP code
```
CommutatorsFormGroup := G -> Length(Set(Cartesian(AsList(G),AsList(G)),
                                        t -> Comm(t[1],t[2])))
                          = Size(DerivedSubgroup(G));
```

Of course this function uses just a *brute-force* approach, and you might find it worth to look for a better algorithm to check whether the commutators form a subgroup – but for our purposes it is sufficient as it stands. Now we simply need to loop over the groups stored in the Small Groups Library [BEO07] until we find one for which this function returns `false`:

GAP session log
```
gap> n := 0;;
gap> repeat
>      n := n + 1; Print("n = ",n,"\n");
>      G := First(AllGroups(n),G->not CommutatorsFormGroup(G));
>    until G <> fail;
```

Certainly we could think about which groups we can already exclude by means of theory, but for our purposes here this turns out to be not necessary. We obtain the following:

GAP session log
```
gap> IdGroup(G);
[ 96, 3 ]
gap> StructureDescription(G);
"((C4 x C2) : C4) : C3"
```

The possible orders ≤ 256 of groups which have the desired property are 96, 128, 144, 162, 168, 192, 216, 240, 256. A low-degree permutation group with our property is

GAP session log

```
gap> G := Group((1,3)(2,4),(5,7)(6,8),(9,11)(10,12),(13,15)(14,16),
>               (1,3)(5,7)(9,11),(1,2)(3,4)(13,15),(5,6)(7,8)(13,14)(15,16),
>               (9,10)(11,12));;
gap> CommutatorLength(G);
2
```

2.2 On Outer Automorphisms Fixing Conjugacy Classes (Solution)

This is a suggested solution for Exercise 1.2.

We perform a *brute-force* search. For this we write straightforward code that uses a "problem-oriented" approach:

GAP session log

```
gap> n := 1;;
gap> repeat
>      Print(n,"\n");
>      G := First(AllGroups(n),
>                 G -> not IsAbelian(G)
>                  and ForAny(AsList(AutomorphismGroup(G)),
>                             aut -> not IsInnerAutomorphism(aut)
>                                and ForAll(AsList(G),
>                                           g->IsConjugate(G,g,
>                                                          Image(aut,g)))));
>      n := n + 1;
>    until G <> fail;
```

This loop runs a couple of minutes. This is likely less than the additional time it would take to exclude many groups from the search by means of theory or to write code which tests given groups faster. Nevertheless you might be interested in looking at this question from a theoretical point of view and in trying to find out more about the structure of groups which permit non-inner conjugacy class fixing automorphisms.

We obtain the following:

GAP session log

```
gap> G;
<pc group of size 32 with 5 generators>
gap> IdGroup(G);
[ 32, 43 ]
```

We would like to find a nice faithful permutation representation of G which permits us to see how our automorphism looks like. For this purpose we look for suitable subgroups, and let G act from the right on right cosets. Since $32 > 4!$, there is no hope to find a faithful permutation representation of degree 4. However we can find one of degree 8:

```
GAP session log
gap> S4 := Filtered(List(ConjugacyClassesSubgroups(G),Representative),
>                   S->Size(S)=4);;
gap> permreps := Set(Filtered(List(S4,S->Action(G,RightCosets(G,S),OnRight)),
>                             H->Size(H)=32));
[ Group([ (1,2)(3,8)(4,6)(5,7), (2,8)(3,7)(5,6), (2,6)(5,8),
      (1,3,4,7)(2,5,6,8), (1,4)(2,6)(3,7)(5,8) ]) ]
gap> G := permreps[1];;
gap> ConjugacyClasses(G);
[ ()^G, (2,5)(3,7)(6,8)^G, (2,6)(5,8)^G, (2,8)(3,7)(5,6)^G,
  (1,2)(3,8)(4,6)(5,7)^G, (1,2,3,5,4,6,7,8)^G, (1,2,4,6)(3,8,7,5)^G,
  (1,2,7,8,4,6,3,5)^G, (1,3,4,7)(2,5,6,8)^G, (1,3,4,7)(2,8,6,5)^G,
  (1,4)(2,6)(3,7)(5,8)^G ]
gap> G := G^(4,5);; # we want (1,2,3,4,5,6,7,8) in G
gap> List(ConjugacyClasses(G),Representative); # pick nicer generators ...
[ (), (2,4)(3,7)(6,8), (2,6)(4,8), (2,8)(3,7)(4,6), (1,2)(3,8)(4,7)(5,6),
  (1,2,3,4,5,6,7,8), (1,2,5,6)(3,8,7,4), (1,2,7,8,5,6,3,4),
  (1,3,5,7)(2,4,6,8), (1,3,5,7)(2,8,6,4), (1,5)(2,6)(3,7)(4,8) ]
gap> G = Group((1,2,3,4,5,6,7,8),(2,4)(3,7)(6,8),(2,8)(3,7)(4,6));
true
```

Now let's have a look at our automorphism:

```
GAP session log
gap> a := (1,2,3,4,5,6,7,8);; b := (2,4)(3,7)(6,8);; c := (2,8)(3,7)(4,6);;
gap> G := Group(a,b,c);; # the nice representation determined above
gap> A := AutomorphismGroup(G);
<group of size 64 with 6 generators>
gap> auts := Filtered(AsList(A),
>                     aut ->  not IsInnerAutomorphism(aut)
>                         and ForAll(AsList(G),
>                                    g -> IsConjugate(G,g,Image(aut,g))));;
gap> Length(auts); # there are 16 suitable aut's -- find the nicest of them:
16
gap> auts := Filtered(auts,aut->Image(aut,a)=a);; # those fixing a
gap> Length(auts);
4
gap> auts := Filtered(auts,aut->Image(aut,b)=b);; # those fixing b as well
gap> Length(auts);
1
gap> aut := auts[1];; # this one fixes a and b, and moves the generator c
gap> c; Image(aut,c);
(2,8)(3,7)(4,6)
(1,5)(2,4)(6,8)
gap> c/Image(aut,c);
(1,5)(2,6)(3,7)(4,8)
gap> last = a^4; # our automorphism multiplies c by a^4
true
gap> aut = GroupHomomorphismByImages(G,G,[a,b,c],[a,b,a^4*c]); # check
true
```

2.3 Drawing the Ulam Spiral (Solution)

This is a suggested solution for Exercise 1.3.

We can write the function as follows:

GAP code

```
UlamSpiral := function ( size, filename )

  local  spiral, smallprimes, n, p, r,
         middle, edgelength, edgepos, direction, i, j, zero, one;

  smallprimes := Filtered([2..size],IsPrimeInt);
  spiral      := NullMat(size,size,GF(2));

  if size mod 2 = 0 then middle := [size/2,size/2];
                    else middle := [(size + 1)/2,(size + 1)/2]; fi;

  zero := Zero(GF(2)); one := One(GF(2));

  spiral[middle[1]][middle[2]] := one;

  for p in smallprimes do
    i := middle[1]; j := middle[2];
    edgelength := 2; edgepos := 1; direction := 0; r := 1;
    for n in [2..size^2] do
      if   direction = 0 then j := j + 1;
      elif direction = 1 then i := i + 1;
      elif direction = 2 then j := j - 1;
      elif direction = 3 then i := i - 1; fi;
      r       := r + 1;
      edgepos := edgepos + 1;
      if r = p then
        if n > p then spiral[i][j] := one; fi;
        r := 0;
      fi;
      if edgepos = edgelength then
        direction := (direction + 1) mod 4;
        if direction in [0,2] then edgelength := edgelength + 1; fi;
        edgepos := 1;
      fi;
    od;
  od;

  SaveAsBitmapPicture(spiral,filename);
end;
```

In this solution, we do a *Sieve of Eratosthenes* inside the spiral. This saves some memory, but doing the sieving before drawing the spiral would be faster.

2.4 Automorphism Group of the Smallest Projective Plane (Solution)

This is a suggested solution for Exercise 1.4.

We determine the requested automorphism group as a subgroup of the symmetric group of degree 7 as follows:

GAP session log

```
gap> p7 := [[1,2,3],[1,4,7],[1,5,6],[2,4,6],[2,5,7],[3,4,5],[3,6,7]];;
gap> G := SubgroupProperty(SymmetricGroup(7),x->Set(p7,g->Set(g,p->p^x))=p7);
Group([ (2,6)(3,5), (2,5)(3,6), (2,5,7)(3,6,4), (1,2,5,6,4,3,7) ])
gap> StructureDescription(G);
"PSL(3,2)"
```

The group G is isomorphic to PSL(3,2) since the smallest projective plane is the one over the field GF(2) with 2 elements, and since the group PSL(n,q) is defined as the automorphism group of the projective space of affine dimension n (i.e. of projective dimension $n-1$) over GF(q). Our approach works since all groups we are dealing with are small, thus `SubgroupProperty` is practical.

2.5 Installing a Missing Method (Solution)

This is a suggested solution for Exercise 1.5.

We can implement the needed method as follows:

GAP code

```
InstallMethod( Centre, "for simple groups", true, [ IsSimpleGroup ], 50,
               function( G )
                 if   IsAbelian( G ) then return G;
                 else return TrivialSubgroup( G ); fi;
               end );
```

We set the rank to 50 to make GAP's method selection choose this method in cases where multiple methods are applicable.

2.6 Finding Good abc Triples (Solution)

This is a suggested solution for Exercise 1.6.

We can write the function as follows:

GAP code

```
abcTriplesByRadical := function ( radical, bound )

  local  triples, values, values_last, factors, a, b, c, rad_abc, ratio;

  factors := Union(Factors(radical),[1]);
  values  := [1];
  repeat
    values_last := values;
    values := Filtered(Union(List(factors,p->values*p)),n->n<bound);
  until values = values_last;
  triples := [];
  for a in values do
    for b in values do
      if b >= a then break; fi;
      if a + b in values and Gcd(a,b) = 1 then
        c       := a + b;
        rad_abc := Product(Set(Factors(a*b*c)));
        ratio   := LOG_FLOAT(Float(c))/LOG_FLOAT(Float(rad_abc));
        if ratio > 7/5 then Add(triples,[a,b,c]); fi;
      fi;
    od;
  od;
  return triples;
end;
```

In the first loop we determine all positive integers less than *bound* all of whose prime factors divide *radical*. For this we neither perform a loop over all integers in the range from 1 to *bound* nor we factor integers. In the nested loop afterwards we determine the abc triples by a *brute-force* search.

We obtain for example the following:

GAP session log
```
gap> abcTriplesByRadical( 2 * 3 * 5 * 7, 10000 );
[ [ 125, 3, 128 ], [ 2400, 1, 2401 ], [ 4374, 1, 4375 ] ]
gap> abcTriplesByRadical( 2 * 3 * 23 * 109, 10^7 );
[ [ 6436341, 2, 6436343 ] ]
```

The latter is currently the best known abc triple. Its ratio is $\ln(c)/\ln(\mathrm{rad}(abc)) \approx 1.62991$.

A more elaborate method to search for good abc triples is described in [Dok03].

2.7 Automorphism Groups of Finite Graphs (Solution)

This is a suggested solution for Exercise 1.7.

ad a)

We let the symmetric group of the set of vertices act on the set of all graphs with the given number of vertices. The orbits under that action are the isomorphism classes. As said in the hints, we obtain the set of all graphs with *n* vertices by `Combinations(Combinations([1..n],2))`. We can write a GAP function which takes an argument *n* and which returns a set of representatives for the isomorphism classes of graphs with *n* vertices:

GAP code
```
AllGraphs := n -> List(Orbits(SymmetricGroup(n),
                              Combinations(Combinations([1..n],2)),
                              function(Gamma,g)
                                return Set(Gamma,k->OnSets(k,g));
                              end),
                    Representative);
```

For $n = 1, 2, 3, 4, 5$ and 6, we obtain 1, 2, 4, 11, 34 and 156 graphs, respectively. We observe a significant increase in runtime requirements between $n = 5$ and $n = 6$.

ad b)

We can write the GAP function as follows:

GAP code

```
GraphAutomorphismGroup := function( Gamma, n )
  return SubgroupProperty(SymmetricGroup(n),
                          g -> Set(Gamma,k->OnSets(k,g)) = Set(Gamma));
end;
```

Of course we could put much more effort into writing such a function in order to obtain a satisfactory performance also for reasonably large graphs – but for our purposes the given one is already good enough.

ad c)

We write a GAP function which determines all transitive permutation groups of given degree n which occur as automorphism groups of graphs:

GAP code

```
TransitiveGraphAutomorphismGroups := function( n )

  local  graphs, groups;

  graphs := AllGraphs(n);
  groups := Filtered(List(graphs,Gamma->GraphAutomorphismGroup(Gamma,n)),
                     G -> IsTransitive(G,[1..n]));
  return AllTransitiveGroups(NrMovedPoints,n)
           {Set(groups,TransitiveIdentification)};
end;
```

As often, it is possible to abridge this function without performance loss:

GAP code

```
TransitiveGraphAutomorphismGroups :=
  n -> AllTransitiveGroups(NrMovedPoints,n)
         {Set(Filtered(List(AllGraphs(n),
                            Gamma->GraphAutomorphismGroup(Gamma,n)),
                       G -> IsTransitive(G,[1..n])),
              TransitiveIdentification)};
```

In general one needs to be a bit careful to avoid computing the same things again and again, to avoid filling up the memory with junk objects and to recognize other possible sources of performance problems when one shrinks functions in such a way.

We obtain the following:

GAP session log
```
gap> TransitiveGraphAutomorphismGroups(3);
[ S3 ]
gap> TransitiveGraphAutomorphismGroups(4);
[ D(4), S4 ]
gap> TransitiveGraphAutomorphismGroups(5);
[ D(5) = 5:2, S5 ]
gap> TransitiveGraphAutomorphismGroups(6);
[ D(6) = S(3)[x]2, 2S_4(6) = [2^3]S(3) = 2 wr S(3),
  F_36(6):2 = [S(3)^2]2 = S(3) wr 2, S6 ]
gap> List(last, Size);
[ 12, 48, 72, 720 ]
gap> List(last2, GeneratorsOfGroup);
[ [ (1,2,3,4,5,6), (1,4)(2,3)(5,6) ], [ (3,6), (1,3,5)(2,4,6), (1,5)(2,4) ],
  [ (2,4,6), (2,4), (1,4)(2,5)(3,6) ], [ (1,2,3,4,5,6), (1,2) ] ]
gap> List(last3, StructureDescription);
[ "D12", "C2 x S4", "(S3 x S3) : C2", "S6" ]
```

Now we determine the corresponding graphs. Since the automorphism group of a graph is invariant under taking the complement, we can restrict our considerations to graphs with at most $[\frac{1}{2} \cdot \binom{6}{2}] = 7$ edges – complements of solutions are then solutions as well.

GAP session log
```
gap> HomogeneousGraphs :=
>      Filtered(AllGraphs(6),
>               Gamma -> Length(Gamma) <= 7 and
>                        IsTransitive(GraphAutomorphismGroup(Gamma,6),
>                                     [1..6]));
[ [  ], [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ], [ 4, 5 ], [ 4, 6 ], [ 5, 6 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 2, 4 ], [ 3, 5 ], [ 4, 6 ], [ 5, 6 ] ],
  [ [ 1, 2 ], [ 3, 4 ], [ 5, 6 ] ] ]
gap> List(HomogeneousGraphs, Gamma -> Size(GraphAutomorphismGroup(Gamma, 6)));
[ 720, 72, 12, 48 ]
```

Thus for the dihedral group of order 12 we get the hexagon, for $C_2 \wr S_3$ we get the graph consisting of 3 disconnected edges, for $S_3 \wr C_2$ we get the graph consisting of 2 separate triangles, and for S_6 we get the empty graph.

There is a GAP package GRAPE [Soi02], which is dedicated to computations with graphs.

2.8 Enumerating Paths (Solution)

This is a suggested solution for Exercise 1.8.

ad a)

We can write the following function to determine a suitable $x \in \mathbb{Z}^{24\times 24}$:

GAP code

```
TranspositionMatrixSn := function ( n )

  local  x, Sn, transpositions, line, g, t;

  Sn := AsList(SymmetricGroup(n));
  transpositions := List(Combinations([1..n],2),t->(t[1],t[2]));
  x := [];
  for g in Sn do
    line := ListWithIdenticalEntries(Factorial(n), 0);
    for t in transpositions do line[Position(Sn,g*t)] := 1; od;
    Add(x, line);
  od;
  return x;
end;
```

The lines and columns of the matrix correspond to the elements $1 = g_1, \dots, g_{24}$ of S_4, and it is

$$x_{ij} = \begin{cases} 1 & \text{if there is a transposition } t \text{ such that } g_i \cdot t = g_j, \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to check that $(x^n)_{ij}$ is the number of ways to take g_i to g_j by multiplication from the right by n transpositions. Therefore we get the solution as follows:

GAP session log

```
gap> x := TranspositionMatrixSn(4);; y := x^100;; y[1][1];
54443218625005908841390855596504818378095309207030310578760502581913955860480
```

ad b)

We can write the following function to determine a suitable $x \in \mathbb{Z}^{64 \times 64}$:

GAP code

```
HorsesMatrix := function ( )

  local  x, board, moves, m, i, j;

  x := [];
  for i in [1..8] do
    for j in [1..8] do
      board := NullMat(8,8);
      moves := [[i-2,j-1], [i-2,j+1], [i-1,j-2], [i-1,j+2],
                [i+2,j+1], [i+2,j-1], [i+1,j+2], [i+1,j-2]];
      for m in Intersection(moves,Cartesian([1..8],[1..8]))
      do board[m[1]][m[2]] := 1; od;
      Add(x,Flat(board));
    od;
  od;
  return x;
```

```
end;
```

We proceed analogous to Part a). Here the lines and columns of the correspond to the 64 squares of the chess board, and the matrix is filled with zeros and ones in such a way that $x_{ij} = 1$ if and only if the horse can jump from square i to square j in one move. Similar as above, we get the solution as follows:

GAP session log

```
gap> x := HorsesMatrix();;
gap> y := x^100;; y[1][64];
2593244602149234588139078903115618952040745476069710377506002611030781169300
```

2.9 Wieferich Primes (Solution)

This is a suggested solution for Exercise 1.9.

We can write the function as follows:

GAP code

```
IsWieferichPrimeInt := p -> IsPrimeInt(p) and PowerModInt(2,p-1,p^2) = 1;
```

The first two Wieferich primes can be found very quickly:

GAP session log

```
gap> Filtered([1..10000],IsWieferichPrimeInt);
[ 1093, 3511 ]
```

However, presently no third Wieferich prime is known. In case there is one, it must be greater than $1.25 \cdot 10^{15}$ (cf. McIntosh 2004).

Assuming "equidistribution" of the residues $(2^{p-1} - 1)/p \bmod p$, one might argue that the "probability" of a prime p to be a Wieferich prime should be about $1/p$. Since the series $\sum_{p \text{ prime}} 1/p$ diverges, this would suggest that there are infinitely many Wieferich primes. More concisely, one would expect that there are roughly $\ln(\ln(n))$ Wieferich primes less than a given bound n. Following these speculations, the expected number of Wieferich primes below the current bound of $1.25 \cdot 10^{15}$ would be 3.548, while the actual number is 2. Obviously no reasonable statistical conclusions can be made from that difference.

Well – when trying random primes, it seems that you might perhaps have a little chance of finding a new Wieferich prime!

2.10 Counting Words in a File (Solution)

This is a suggested solution for Exercise 1.10.

We can write the function as follows:

GAP code
```
WordCount := filename -> Collected(WordsString(StringFile(filename)));
```

2.11 Non-Metabelian p-Groups (Solution)

This is a suggested solution for Exercise 1.11.

Let p be a prime. It is well-known that groups of order p or p^2 are always abelian, and that groups of prime-power order are solvable. Therefore a non-metabelian p-group must at least have order p^6. Further it is easy to see that a group is metabelian if and only if its derived subgroup is abelian. Thus we can proceed as follows:

GAP session log
```
gap> G := First( AllGroups( 64), G -> not IsAbelian( DerivedSubgroup(G) ) );
fail
gap> G := First( AllGroups(128), G -> not IsAbelian( DerivedSubgroup(G) ) );
<pc group of size 128 with 7 generators>
gap> IdGroup(G);
[ 128, 134 ]
gap> StructureDescription(G);
"((C4 : C8) : C2) : C2"
gap> StructureDescription( DerivedSubgroup(G) );
"C2 x D8"
```

There is a complete classification of the positive integers n such that all groups of order n are metabelian. See [Paz80].

2.12 The Growth of the Sum-of-Divisors Function (Solution)

This is a suggested solution for Exercise 1.12.

We can compute floating point approximations for $H(n)$ and $B(n)$ in GAP as follows:

GAP code
```
H := n -> Sum([1..n],i->Float(1/i));
B := n -> H(n) + LOG_FLOAT(H(n)) * EXP_FLOAT(H(n));
```

We obtain for example

GAP session log
```
gap> List( [ 1 .. 60 ], n -> B(n) - Sigma(n) );
[ 0, 0.317169, 1.62453, 0.977983, 4.38227, 0.834179, 7.32927, 2.86332,
  7.43259, 5.03387, 13.6644, 0.321837, 17.0041, 9.70942, 12.4362, 8.1831,
  23.9489, 5.73238, 27.5327, 8.34888, 21.1802, 20.0258, 34.8851, 1.7575,
  33.6424, 25.5393, 30.4477, 17.3671, 46.2972, 7.2375, 50.1877, 22.1475,
  40.1165, 37.0945, 46.0811, 6.0761, 62.0793, 43.0903, 50.1091, 19.1353,
  70.1688, 19.2093, 74.2568, 37.311, 46.3717, 55.4389, 82.5122, 9.59174,
  79.6772, 46.7685, 70.8655, 47.9681, 95.0762, 32.1896, 83.3082, 38.432,
  81.5609, 74.6946, 107.833, 2.97668 ]
```

However we cannot answer the question whether there is a positive integer n such that $\sigma(n) > B(n)$ – in fact, Jeffrey C. Lagarias [Lag02] has shown that this question is equivalent to the Riemann hypothesis!

2.13 Pell's Equation (Solution)

This is a suggested solution for Exercise 1.13.

We can write the function as follows:

GAP code

```
FundamentalSolutionOfPellsEquation := function ( n )

  local  x, periodlength, approx;

  if n = RootInt(n,2)^2 then return fail; fi;
  x := Indeterminate(Integers);
  periodlength := Length(ContinuedFractionExpansionOfRoot(x^2-n,0)) - 1;
  if   periodlength mod 2 = 0
  then approx := ContinuedFractionApproximationOfRoot(x^2-n,  periodlength);
  else approx := ContinuedFractionApproximationOfRoot(x^2-n,2*periodlength);
  fi;
  return [ NumeratorRat(approx), DenominatorRat(approx) ];
end;
```

First we determine the length of the period of the continued fraction expansion of the square root of `n`. Then we determine the fundamental solution of Pell's equation by taking the numerator and the denominator of a suitable continued fraction approximation of that square root (note the dependency on the parity of the period length!). Examples are

GAP session log

```
gap> FundamentalSolutionOfPellsEquation(2);
[ 3, 2 ]
gap> FundamentalSolutionOfPellsEquation(5);
[ 9, 4 ]
gap> FundamentalSolutionOfPellsEquation(13);
[ 649, 180 ]
gap> FundamentalSolutionOfPellsEquation(15);
[ 4, 1 ]
gap> FundamentalSolutionOfPellsEquation(61);
[ 1766319049, 226153980 ]
```

... and for $n = 421$ we obtain

GAP session log

```
gap> FundamentalSolutionOfPellsEquation(421);
```

```
[ 3879474045914926879468217167061449, 189073995951839020880499780706260 ]
```

2.14 Automorphism Groups of Odd Order (Solution)

This is a suggested solution for Exercise 1.14.

It suffices to look at groups of odd order: Assume that G is a group of even order, and let G_2 be its Sylow 2-subgroup. If G_2 is not a subgroup of the centre of G, then already the inner automorphism group has even order. If it is a subgroup of the centre of G, then we have $G = G_2 \times G_{2'}$, and therefore $\mathrm{Aut}(G) = \mathrm{Aut}(G_2) \times \mathrm{Aut}(G_{2'})$. In this case, already the automorphism group of $G_{2'}$ has odd order, and G is not the smallest group having this property.

We perform a *brute-force* search:

GAP session log

```
gap> n := 3;;
gap> repeat
>     Print(n,"\n");
>     G := First(AllGroups(n),G->Size(AutomorphismGroup(G)) mod 2 = 1);
>     n := n + 2;
>   until G <> fail;
```

This loop runs for quite a while. We obtain the following:

GAP session log

```
gap> G;
<pc group of size 729 with 6 generators>
gap> IdGroup(G);
[ 729, 31 ]
gap> AutomorphismGroup(G);
<group of size 19683 with 9 generators>
gap> StructureDescription(DerivedSubgroup(G));
"C9"
gap> StructureDescription(G/DerivedSubgroup(G));
"C9 x C9"
```

The result that 3^6 is the smallest order of a group of more than 2 elements with an odd order automorphism group has first been obtained in [MS95].

2.15 Composite Sums (Solution)

This is a suggested solution for Exercise 1.15.

First we look for odd integers n such that any sum $n+2^k$ for "small" k has a "small" prime divisor:

GAP session log

```
gap> m := Product(Primes);;
gap> Filtered([1,3..99999],n->First([0..500],k->Gcd(n+2^k,m)=1)=fail);
```

```
[ 78557 ]
```

Now our goal is to find out whether in fact *all* sums $78557+2^k$ are composite.

For this, we first determine the smallest prime factors of the numbers $78557+2^k$ for "small" values of k. Then we find out for which k these primes p_i divide $78557+2^k$. The set of such k is the set of positive integers in a residue class modulo the order of 2 (mod p_i). Finally we form the union of the residue classes we obtain, and check whether it equals $\mathbb{Z}$ (this needs the ResClasses package [Koh07c]):

GAP session log

```
gap> primes := Set([1..100],k->Minimum(Factors(Gcd(78557+2^k,m))));
[ 3, 5, 7, 13, 19, 37, 73 ]
gap> m_i := List(primes,p->OrderMod(2,p));
[ 2, 4, 3, 12, 18, 36, 9 ]
gap> r_i := List([1..Length(primes)],
>                i->First([0..m_i[i]-1],k->(78557+2^k) mod primes[i] = 0));
[ 0, 3, 2, 1, 3, 9, 6 ]
gap> residueclasses := List(TransposedMat([r_i,m_i]),ResidueClass);
[ 0(2), 3(4), 2(3), 1(12), 3(18), 9(36), 6(9) ]
gap> Union(residueclasses);
Integers
```

Now we know that all sums $78557+2^k$ for positive integers k are composite. But is 78557 indeed the smallest odd integer such that $n+2^k$ is composite for all positive integers k? – Likely yes, but answering this question is computationally difficult:

GAP session log

```
gap> k_s := List([1,3..78557],
>                n->First([1..1000],k->IsProbablyPrimeInt(n+2^k)));;
gap> k_s{[1..100]}; # small n are not a problem ...
[ 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 3, 2, 1, 1, 4, 2, 1, 2, 1, 1, 2, 1, 5, 2,
  1, 3, 2, 1, 1, 8, 2, 1, 2, 1, 1, 4, 2, 1, 2, 1, 7, 2, 1, 3, 4, 2, 1, 2, 1,
  1, 2, 1, 1, 2, 1, 7, 4, 5, 3, 4, 2, 1, 2, 1, 3, 2, 1, 1, 10, 3, 3, 2, 1, 1,
  4, 2, 1, 4, 2, 1, 2, 1, 5, 2, 1, 3, 2, 1, 1, 4, 3, 3, 2, 1, 1, 2, 1, 1, 6 ]
gap> 2*Positions(k_s,fail)-1; # ... but larger n are:
[ 2131, 2491, 4471, 5101, 6379, 6887, 7013, 8447, 8543, 9833, 10711, 14033,
  14551, 14573, 14717, 15623, 16519, 17659, 18527, 19081, 19249, 20209,
  20273, 21143, 21661, 22193, 23147, 23221, 23971, 24953, 26213, 26491,
  28433, 29333, 29777, 30197, 31111, 31369, 31951, 32449, 32513, 34429,
  35461, 36083, 36721, 37217, 37967, 38387, 39079, 40291, 40351, 40613,
  41453, 41693, 43579, 47269, 48091, 48331, 48527, 48859, 48961, 49279,
  49577, 50839, 52339, 53119, 53359, 56717, 57083, 59071, 60443, 60451,
  60947, 60961, 62029, 63691, 64133, 64643, 65033, 65089, 65719, 67607,
  69593, 69709, 70321, 72679, 73373, 73583, 75353, 75841, 77041, 77783,
  77899, 78557 ]
```

Using a bound larger than 1000 for k, it is possible to eliminate many of the above values of n, but eliminating all of them except of 78557 seems hard.

See also the related term Sierpinski number, and the corresponding distributed computing project Seventeen or bust.

The interested reader might have a look at the similar problem with Fibonacci numbers instead of powers of 2.

2.16 Rational Points on the Unit Sphere (Solution)

This is a suggested solution for Exercise 1.16.

The function can be written as follows:

GAP code
```
RationalPointsOnUnitSphere := function ( max_abc, size, filename )

  local  picture, solutions, a, b, c, d, sum,
         pixelcoords, mirror, zero, one, i, j;

  zero := Zero(GF(2)); one := One(GF(2)); # create a white picture:
  picture := NullMat(size,size,GF(2));
  for i in [1..size] do for j in [1..size] do picture[i][j] := one; od; od;

  solutions := 0;
  for a in [1..max_abc] do
    Print("a = ",a,", #solutions = ",solutions,"\n");
    for b in [1..a] do
      for c in [1..b] do
        sum := a^2 + b^2 + c^2;
        d   := RootInt(sum);
        if d^2 = sum then
          pixelcoords := List( size * [a,b,c]/d, Int ) + 1;
          for mirror in Arrangements([1..3],2) do
            picture[pixelcoords[mirror[1]]][pixelcoords[mirror[2]]] := zero;
          od;
          solutions := solutions + 1;
        fi;
      od;
    od;
  od;
  SaveAsBitmapPicture(picture,filename);
end;
```

With some patience, you can use this function to produce a picture like this.

2.17 Aliquot Sequences (Solution)

This is a suggested solution for Exercise 1.17.

We can write the following GAP function:

GAP code
```
AliquotSequence := function ( n )
```

```
  local  a, i;

  a := [n]; i := 1;
  while a[i] > 1 and not a[i] in a{[1..i-1]} do
    a[i+1] := Sigma(a[i]) - a[i]; i := i + 1;
  od;
  return a;
end;
```

This yields the following:

```
GAP session log
gap> List([1..40],AliquotSequence);
[ [ 1 ], [ 2, 1 ], [ 3, 1 ], [ 4, 3, 1 ], [ 5, 1 ], [ 6, 6 ], [ 7, 1 ],
  [ 8, 7, 1 ], [ 9, 4, 3, 1 ], [ 10, 8, 7, 1 ], [ 11, 1 ],
  [ 12, 16, 15, 9, 4, 3, 1 ], [ 13, 1 ], [ 14, 10, 8, 7, 1 ],
  [ 15, 9, 4, 3, 1 ], [ 16, 15, 9, 4, 3, 1 ], [ 17, 1 ], [ 18, 21, 11, 1 ],
  [ 19, 1 ], [ 20, 22, 14, 10, 8, 7, 1 ], [ 21, 11, 1 ],
  [ 22, 14, 10, 8, 7, 1 ], [ 23, 1 ], [ 24, 36, 55, 17, 1 ], [ 25, 6, 6 ],
  [ 26, 16, 15, 9, 4, 3, 1 ], [ 27, 13, 1 ], [ 28, 28 ], [ 29, 1 ],
  [ 30, 42, 54, 66, 78, 90, 144, 259, 45, 33, 15, 9, 4, 3, 1 ], [ 31, 1 ],
  [ 32, 31, 1 ], [ 33, 15, 9, 4, 3, 1 ], [ 34, 20, 22, 14, 10, 8, 7, 1 ],
  [ 35, 13, 1 ], [ 36, 55, 17, 1 ], [ 37, 1 ], [ 38, 22, 14, 10, 8, 7, 1 ],
  [ 39, 17, 1 ], [ 40, 50, 43, 1 ] ]
gap> List([1..275],n->Length(AliquotSequence(n)));
[ 1, 2, 2, 3, 2, 2, 2, 3, 4, 4, 2, 7, 2, 5, 5, 6, 2, 4, 2, 7, 3, 6, 2, 5, 3,
  7, 3, 2, 2, 15, 2, 3, 6, 8, 3, 4, 2, 7, 3, 4, 2, 14, 2, 5, 7, 8, 2, 6, 4,
  3, 4, 9, 2, 13, 3, 5, 3, 4, 2, 11, 2, 9, 3, 4, 3, 12, 2, 5, 4, 6, 2, 9, 2,
  5, 5, 5, 3, 11, 2, 7, 5, 6, 2, 6, 3, 9, 7, 7, 2, 10, 4, 6, 4, 4, 4, 9, 2,
  3, 4, 5, 2, 18, 2, 7, 8, 6, 2, 10, 2, 7, 3, 9, 2, 17, 3, 5, 4, 10, 4, 12,
  8, 5, 8, 6, 3, 16, 2, 3, 3, 6, 2, 11, 4, 7, 9, 8, 2, 178, 2, 5, 5, 6, 4, 9,
  4, 6, 6, 11, 2, 177, 2, 12, 6, 8, 3, 8, 2, 7, 4, 11, 3, 4, 2, 7, 9, 10, 2,
  175, 6, 9, 3, 9, 2, 16, 3, 5, 4, 7, 2, 52, 2, 9, 4, 6, 3, 15, 3, 12, 3, 10,
  2, 13, 2, 6, 6, 4, 2, 14, 2, 4, 3, 8, 3, 10, 3, 7, 9, 7, 3, 52, 2, 11, 6,
  8, 5, 10, 4, 10, 4, 3, 3, 176, 2, 17, 8, 6, 2, 8, 2, 9, 7, 11, 2, 175, 3,
  7, 3, 7, 2, 11, 2, 3, 9, 11, 3, 15, 7, 12, 8, 11, 2, 17, 4, 7, 5, 6, 2, 14,
  8, 11, 4, 8, 2, 31, 3, 9, 5, 8, 2, 13, 2, 12, 4, 6, 3 ]
gap> Maximum(last);
178
gap> Position(last2,last);
138
gap> AliquotSequence(138);
[ 138, 150, 222, 234, 312, 528, 960, 2088, 3762, 5598, 6570, 10746, 13254,
  13830, 19434, 20886, 21606, 25098, 26742, 26754, 40446, 63234, 77406,
  110754, 171486, 253458, 295740, 647748, 1077612, 1467588, 1956812, 2109796,
  1889486, 953914, 668966, 353578, 176792, 254128, 308832, 502104, 753216,
  1240176, 2422288, 2697920, 3727264, 3655076, 2760844, 2100740, 2310856,
  2455544, 3212776, 3751064, 3282196, 2723020, 3035684, 2299240, 2988440,
  5297320, 8325080, 11222920, 15359480, 19199440, 28875608, 25266172,
  19406148, 26552604, 40541052, 54202884, 72270540, 147793668, 228408732,
  348957876, 508132204, 404465636, 303708376, 290504024, 312058216,
294959384, 290622016, 286081174, 151737434, 75868720, 108199856, 101437396,
76247552, 76099654, 42387146, 21679318, 12752594, 7278382, 3660794,
1855066, 927536, 932464, 1013592, 1546008, 2425752, 5084088, 8436192,
13709064, 20563656, 33082104, 57142536, 99483384, 245978376, 487384824,
745600776, 1118401224, 1677601896, 2538372504, 4119772776, 8030724504,
14097017496, 21148436904, 40381357656, 60572036544, 100039354704,
179931895322, 94685963278, 51399021218, 28358080762, 18046051430,
17396081338, 8698040672, 8426226964, 6319670230, 5422685354, 3217383766,
1739126474, 996366646, 636221402, 318217798, 195756362, 101900794,
54202694, 49799866, 24930374, 17971642, 11130830, 8904682, 4913018,
3126502, 1574810, 1473382, 736694, 541162, 312470, 249994, 127286, 69898,
34952, 34708, 26038, 13994, 7000, 11720, 14740, 19532, 16588, 18692, 14026,
7016, 6154, 3674, 2374, 1190, 1402, 704, 820, 944, 916, 694, 350, 394, 200,
265, 59, 1 ]
```

Thus we see that indeed all Aliquot sequences starting at integers $n \leq 275$ either stop or run into cycles. So far, everything runs quickly and there are no performance problems of any kind. However, $n = 276$ causes severe problems – the sequence grows, and factoring becomes a serious problem. As suggested in the hints, we install the following modified GAP Library method for the operation `Sigma` to ensure that FactInt [Koh07a] is used for factoring integers:

GAP code

```
InstallMethod( Sigma, "use FactInt", true, [ IsPosInt ], SUM_FLAGS,

  function( n )

    local  sigma, p, q, k;

    # make <n> it nonnegative, handle trivial cases
    if n < 0 then n := -n; fi;
    if n = 0 then Error("Sigma: <n> must not be 0"); fi;
    if n <= Length(DivisorsIntCache) then
      return Sum(DivisorsIntCache[n]);
    fi;

    # loop over all prime factors p of n
    sigma := 1;
    for p in Set(Factors(n)) do

      # compute p^e and k = 1+p+p^2+..p^e
      q := p;  k := 1 + p;
      while n mod (q * p) = 0 do q := q * p; k := k + q; od;

      # combine with the value found so far
      sigma := sigma * k;

    od;

    return sigma;
  end );
```

Further we insert a `Print` statement into our `AliquotSequence` function, and we switch on FactInt's `Info`'s whenever large integers are to be factored:

GAP code
```
AliquotSequence := function ( n )

  local  a, i;

  a := [n]; i := 1;
  while a[i] > 1 and not a[i] in a{[1..i-1]} do
    if a[i] > 10^40 then FactIntInfo(3); fi;
    Factors(a[i]);
    FactIntInfo(0);
    Print(String(i,6)," : ",a[i]," = ");
    PrintFactorsInt(a[i]); Print("\n");
    a[i+1] := Sigma(a[i]) - a[i];
    i := i + 1;
  od;
  return a;
end;
```

Now try out yourself how far you can get with computing the Aliquot sequence starting at 276 ... !

2.18 The Q Sequence (Solution)

This is a suggested solution for Exercise 1.18.

ad a)

We can write either the following recursive

GAP code
```
Q := function ( n )
  if   n in [1,2]
  then return 1;
  else return Q(n-Q(n-1)) + Q(n-Q(n-2)); fi;
end;
QSequence := l -> List([1..l],Q);
```

or the following iterative code;

GAP code
```
QSequence := function ( l )

  local  Q, n;

  Q := [1,1];
  for n in [3..l] do
    Q[n] := Q[n-Q[n-1]] + Q[n-Q[n-2]];
  od;
  return Q;
end;
```

With the recursive approach, we get something like the following timings:

GAP session log
```
gap> QSequence(10);
[ 1, 1, 2, 3, 3, 4, 5, 5, 6, 6 ]
gap> time;
0
gap> QSequence(20);
[ 1, 1, 2, 3, 3, 4, 5, 5, 6, 6, 6, 8, 8, 8, 10, 9, 10, 11, 11, 12 ]
gap> time;
220
gap> QSequence(30);
[ 1, 1, 2, 3, 3, 4, 5, 5, 6, 6, 6, 8, 8, 8, 10, 9, 10, 11, 11, 12, 12, 12,
  12, 16, 14, 14, 16, 16, 16, 16 ]
gap> time;
26640
```

Thus we observe a horrible increase in runtime. With the iterative approach, this looks quite different:

GAP session log
```
gap> QSequence(200);
[ 1, 1, 2, 3, 3, 4, 5, 5, 6, 6, 6, 8, 8, 8, 10, 9, 10, 11, 11, 12, 12, 12,
  12, 16, 14, 14, 16, 16, 16, 16, 20, 17, 17, 20, 21, 19, 20, 22, 21, 22, 23,
  23, 24, 24, 24, 24, 24, 32, 24, 25, 30, 28, 26, 30, 30, 28, 32, 30, 32, 32,
  32, 32, 40, 33, 31, 38, 35, 33, 39, 40, 37, 38, 40, 39, 40, 39, 42, 40, 41,
  43, 44, 43, 43, 46, 44, 45, 47, 47, 46, 48, 48, 48, 48, 48, 48, 64, 41, 52,
  54, 56, 48, 54, 54, 50, 60, 52, 54, 58, 60, 53, 60, 60, 52, 62, 66, 55, 62,
  68, 62, 58, 72, 58, 61, 78, 57, 71, 68, 64, 63, 73, 63, 71, 72, 72, 80, 61,
  71, 77, 65, 80, 71, 69, 77, 75, 73, 77, 79, 76, 80, 79, 75, 82, 77, 80, 80,
  78, 83, 83, 78, 85, 82, 85, 84, 84, 88, 83, 87, 88, 87, 86, 90, 88, 87, 92,
  90, 91, 92, 92, 94, 92, 93, 94, 94, 96, 94, 96, 96, 96, 96, 96, 96, 128,
  72, 96, 115, 100, 84, 114, 110, 93 ]
gap> time;
0
```

Let's have a look how often `Q` calls itself in the recursive version:

GAP code
```
QCallCounts := function ( n )

  local  Q, sequence;

  Q := function ( n )
    sequence[n] := sequence[n] + 1;
    if   n in [1,2]
    then return 1;
    else return Q(n-Q(n-1)) + Q(n-Q(n-2)); fi;
  end;

  sequence := ListWithIdenticalEntries(n,0);
  Q(n);
  return sequence;
end;
```

We obtain

GAP session log

```
gap> QCallCounts(30);
[ 1477421, 5444369, 1477421, 415342, 181422, 98727, 55073, 31366, 18752,
  11250, 6884, 4238, 2604, 1601, 987, 610, 377, 233, 144, 89, 55, 34, 21, 13,
  8, 5, 3, 2, 1, 1 ]
```

This clearly explains the poor performance. (By the way: The end of the sequence seems to be the reversed beginning of the sequence of Fibonacci numbers – can you find out something about this?)

ad b)

We can write the following GAP function:

GAP code

```
PlotQSequence := function ( l, filename )

  local  Q, graph, h, n, i, j, zero, one;

  Q := QSequence(l);
  h := Maximum(Q); # use the maximum as the height of the picture
  graph := NullMat(h,l);
  zero := Zero(GF(2)); one := One(GF(2));
  for i in [1..h] do for j in [1..l] do graph[i][j] := one; od; od;
  for n in [1..l] do graph[Q[n]][n] := zero; od;
  SaveAsBitmapPicture(graph,filename);
end;
```

References

[BEO07] H. U. Besche, B. Eick, and E. O'Brien. *The SmallGroups Library*, 2007. GAP package, `http://www.gap-system.org/Packages/sgl.html`. 8, 9, 14

[Dok03] T. Dokchitser. LLL & ABC, 2003. `http://arxiv.org/abs/math.NT/0307322`. 19

[Koh07a] S. Kohl. *FactInt – Advanced Methods for Factoring Integers (Version 1.5.2)*, 2007. GAP package, `http://www.gap-system.org/Packages/factint.html`. 11, 31

[Koh07b] S. Kohl. *RCWA – Residue-Class-Wise Affine Groups (Version 2.5.4)*, 2007. GAP package, `http://www.gap-system.org/Packages/rcwa.html`. 5, 10, 11, 12

[Koh07c] S. Kohl. *ResClasses – Set-Theoretic Computations with Residue Classes (Version 2.5.3)*, 2007. GAP package, `http://www.gap-system.org/Packages/resclasses.html`. 10, 28

[Lag02] J. C. Lagarias. An elementary problem equivalent to the Riemann hypothesis. *Amer. Math. Monthly*, 109:534–543, 2002. 26

[Lag07] J. C. Lagarias. 3x+1 problem annotated bibliography, 2007. Part I: `http://arxiv.org/abs/math.NT/0309224`, Part II: `http://arxiv.org/abs/math.NT/0608208`. 12

[LN06] F. Lübeck and M. Neunhöffer. *GAPDoc (Version 1.0)*. RWTH Aachen, 2006. GAP package, `http://www.gap-system.org/Packages/gapdoc.html`. 8

[MS95] D. MacHale and R. Sheehy. Finite groups with odd order automorphism groups. *Proc. Roy. Irish Acad. Sect. A*, 95:113–116, 1995. 27

[Paz80] G. Pazderski. The orders of which only belong metabelian groups. *Math. Nachr.*, 95:7–16, 1980. 25

[Soi02] L. Soicher. *GRAPE – GRaph Algorithms using PErmutation groups (Version 4.1)*. Queen Mary, University of London, 2002. GAP package, `http://www.gap-system.org/Packages/grape.html`. 22

Index