A History of Efficiency Problems in Maple

Michael Monagan

November 22, 2025

Abstract

The Maple project began in 1980 at the University of Waterloo. The most important design goal was that Maple be powerful, that is, Maple could solve a wide range of algebraic problems in a reasonable time. The early releases of Maple were not particularly efficient. This was not due to poor algebraic algorithms; rather, it had to do with the choice of the data representation for numbers and polynomials, the use of interpreted code when compiled code was needed, and other design choices that resulted in a loss of efficiency. This paper presents eight efficiency problems that were identified in Maple's history and what was done to fix them.

1 Introduction

Development of Maple began in December 1980 at the University of Waterloo when a group of four professors, Bruce Char, Keith Geddes, Morven Gentleman and Gaston Gonnet, decided to build their own computer algebra system. They chose to implement a kernel in the C language rather than in Lisp. The kernel supported a high level programming language suitable for implementing algebraic algorithms. The programming language was called Maple. It had to be efficient enough so that the algebraic algorithms such as factoring polynomials and computing antiderivatives, could easily be implemented in it.

The first paper on Maple [3] was presented at the EUROCAL '83 conference in London, England in March 1983. The title of that first paper was "The Design of Maple: a compact, portable, and powerful computer algebra system."

Two conference papers were published and presented in 1984, both in July. The first paper "GCDHEU: Heuristic Polynomial GCD Algorithm Based on Integer GCD Computation" was presented at the EUROSAM 1984 meeting in Cambridge, England. It addressed what was, at the time, the most pressing efficiency problem, namely, how to compute a greatest common divisor of two polynomials in one or more variables. The second paper "On the Design and Performance of the Maple System" was presented by Keith Geddes at the 1984 Macsyma Users conference in Schenectady, New York.

I had joined the Maple group as a PhD student in the summer of 1983 having taken the Symbolic Computation course with Bruce Char in winter 1982 and completing my Masters degree under Gaston Gonnet in fall 1982. The 1984 Macsyma Users conference was my first conference. The Maple group drove from Waterloo, Ontario to the conference in Schenectady, New York to show off what Maple could do. At that time we regarded Macsyma as the system to beat. We showed that Maple was a lot faster than Macsyma and Reduce, at least on our benchmarks.

The design of Maple was very successful in one aspect. Almost everyone, both students and faculty, who was working on the Maple project at Waterloo in the 1980s was implementing algorithms in the Maple language. This resulted in a much faster software development than would be possible if we had to implement algorithms in a systems language like C or Lisp. However, in

1984, although some operations in Maple were very efficient, overall, Maple was not that efficient. In this paper I have identified eight efficiency problems in Maple and what was done to address them. They are presented in chronological order. I describe them briefly here in the introduction. The reader can then read them in any order.

Problem 1: The first efficiency problem is the problem of computing the greatest common divisor (GCD) of two polynomials in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$, that is, in n variables with integer coefficients. Early computer algebra systems would often grind to a halt or run out of memory when computing GCDs. The first major new algorithm contributed by the Maple group was the GCDHEU algorithm of Char, Geddes and Gonnet [5] from 1984. The idea is both clever and practical; convert a polynomial GCD computation into an integer GCD computation! This result gave Maple an early advantage over other computer algebra systems on a critical operation.

Problem 2: The second efficiency problem has to do with the cost of evaluating a formula by using assignment. The reason I present this problem is that of all the efficiency improvements that were made to Maple, this one change achieved the greatest overall speedup on Maple's test suite. The problem also highlights two competing design goals; efficiency and usability.

Problem 3: Maple is unique among computer algebra systems in that it stores each subexpression once in memory. This is accomplished using hashing. The third efficiency problem is a horrible efficiency bug in the hash function. It's a candidate for the worst efficiency bug in Maple.

Problem 4: The fourth efficiency problem is a general problem in all computer algebra systems. We have $O(n^2)$ algorithms that should be $O(n)$ or $O(n \log n)$ in our systems. This leads to very slow running times for users. Users of computer algebra systems are not computer scientists. They would not know how to test for a quadratic algorithm. They are unlikely to realize that the computer algebra system is using a quadratic algorithm and complain about it. So slow quadratic algorithms remain in our computer algebra systems for decades!

Problem 5: The fifth efficiency problem is how to represent integers, in particular small integers, which appear often in formulas, for example, the exponents in the monomial $x^2y^3z^2$. Early versions of Lisp [27] started encoding small integers in pointers to avoid allocating memory when doing arithmetic with small integers. Because Macsyma and Reduce were implemented in Lisp, they automatically benefited from this idea. Gaston Gonnet added this to Maple in the 1990s. This resulted in another significant overall speedup on Maple's test suite.

Problem 6: Maple's library for numerical linear algebra was very slow. Partly because of the dominance of Matlab in the engineering market, which is where the money was, Maplesoft redesigned Maple's linear algebra library from scratch for Maple 6 to compete with Matlab. The new library includes compiled codes for hardware floating point precision. The new design also included a new user interface for vectors and matrices. David Hare of Maplesoft led the design effort.

Problem 7: The seventh problem is an inefficiency in polynomial division. The problem was solved by Johnson [13] in 1974 who used a heap to efficiently sort monomials when dividing polynomials. But, since Johnson didn't benchmark his algorithm against computer algebra systems, I think everyone who worked on computer algebra systems missed the implication of Johnson's work. We were not embarrassed into fixing the problem! Roman Pearce [25] rediscovered Johnson's algorithm and integrated heap based polynomial multiplication and division algorithms into Maple 14 in 2010.

Problem 8: What is the best way to represent polynomials in n variables on the computer? The choice matters because it affects the efficiency of polynomial arithmetic and that plays a significant role in a computer algebra system's overall efficiency, for example in polynomial factorization. In 2003, Richard Fateman [8] found that Pari's recursive dense representation was the fastest representation, faster than the sparse representations that other computer algebra systems used. Fateman also found that Maple, Macsyma, and Mathematica were the slowest systems because they use a very general representation for polynomials. To address this, Monagan and Pearce in [20, 21] designed a new representation for polynomials for Maple 17 in 2013. I believe this is the biggest efficiency improvement made to Maple since Maple 6.

I end with a timing benchmark for factoring determinants of matrices of polynomials. Polynomial factorization is a success story in the history of Computer Algebra. We have good algorithms. But it's a huge amount of work to implement them. In a conversation with Tony Hearn, the principal author of Reduce, Tony told me that when they implemented polynomial factorization for Reduce, the "number of pages of code for Reduce doubled"! How good are today's computer algebra systems at polynomial factorization compared with older systems like Macsyma and Reduce? Fortunately we still have access to Maxima and Reduce so we can make such a comparison. I compare polynomial factorization in Maxima [16] with Maple, Magma [1] and Singular [6].

2 Problem 1: Polynomial GCD Computation

The most pressing problem for computer algebra systems in the 1960s and 1970s was how to compute the greatest common divisor (GCD) of two polynomials, A and B, in n variables with integer coefficients. For example $\gcd(x^4 - y^4, 2x^3 - 2y^3) = x - y$.

If $n = 1$ we can use use the Euclidean algorithm. If we initialize $R_0 = A$ and $R_1 = B$, the Euclidean algorithm computes R_2, the remainder of $R_0 \div R_1$ and it uses the fact that $\gcd(R_0, R_1) = \gcd(R_1, R_2)$. Then, if $R_2 \neq 0$, we do another division; we compute the remainder R_3 of $R_1 \div R_2$. If you do this you will notice that a growth occurs in the size of the fractions that appear in the remainders. We can minimize the growth by clearing fractions from the remainders and then removing any integer common factor from their coefficients. For example, if $R_2 = \frac{4}{5}x^2 - \frac{6}{5}$ we set $R_2 = 5R_2 = 4x^2 - 6$ then set $R_2 = R_2/2 = 2x^2 - 3$. The resulting sequence of remainders is called the primitive remainder sequence. An example is shown in Table 1.

$R_0 = A$	$-21x^6 + 31x^5 - 55x^4 - 557x^3 - 209x^2 + 267x - 280$
$R_1 = B$	$-186x^5 - 19x^4 + 266x^3 - 377x^2 - 269x - 415$
R_2	$339879x^4 + 1795280x^3 + 944895x^2 - 1022193x + 1360595$
R_3	$14378212479x^3 + 12070474049x^2 - 11263467788x + 14264257620$
R_4	$101033342348478x^2 + 315914813035081x + 245876515201585$
$R_5 = G$	$3x + 5$
R_6	0

Table 1: A primitive remainder sequence showing the computation of $\gcd(A, B)$ in $\mathbb{Z}[x]$.

The remainder $R_6 = 0$ means $R_5 = 3x + 5$ is the GCD of A and B. Notice that although the degree of the remainders decreases at each step, the size of their integer coefficients increased rapidly. The input polynomials A and B have 3 digit coefficients, their GCD G has 1 digit coefficients but R_4 has 15 digit coefficients! The more division steps there are, the bigger the coefficients get.

Suppose A and B are polynomials in two variables x and y and we run the Euclidean algorithm treating x as the main variable. This time there is also a growth in the degree of the coefficients in y. Table 2 shows an example so that the reader can see this growth. In the example $G = \gcd(A, B) = 3x + 5y - 7$ and the input polynomials A and B are given in Appendix A.

	$\deg(R_i, x)$	$\deg(R_i, y)$	#digits	#terms R_i
$R_0 = A$	9	9	4	55
$R_1 = B$	8	8	4	45
R_2	7	9	7	52
R_3	6	12	11	70
R_4	5	17	16	93
R_5	4	24	21	115
R_6	3	33	25	130
R_7	2	44	31	132
$R_8 = G$	1	1	1	3

Table 2: A primitive remainder sequence showing the computation of $\gcd(A, B)$ in $\mathbb{Z}[y][x]$.

Table 2 shows that the degree of the remainders in y increases from 9 to 44, the number of digits of their integer coefficients increases from 4 to 31, and then we get $R_8 = G$ then $R_9 = 0$ and the algorithm stops. Thus we have a two dimensional growth. If we compute a GCD in n variables an n dimensional growth occurs and the Euclidean algorithm blows up. What does this mean for the user? If a computer algebra system uses any variation of the Euclidean algorithm it will grind to a halt when the remainders explode in size; it will either never come back with an answer or it will run out of memory. This was a very serious problem for early computer algebra systems in the 1960s. It limited what they could compute.

Brown's modular GCD algorithm [2] in 1971 was the first algorithm to avoid the blowup. The GCDHEU algorithm of Char, Geddes and Gonnet [5] from 1984 also avoids the blowup. I want to show you how GCDHEU works for A and B in $\mathbb{Z}[x]$. GCDHEU chooses a positive integer α and computes $\gcd(A(\alpha), B(\alpha)$, an integer GCD. On the example in Table 1, using $\alpha = 1000$, we get

$$\begin{aligned} a = A(1000) &= -20969055557208733280 \\ b = B(1000) &= -186018734377269415 \\ g = \gcd(a, b) &= 3005 \end{aligned}$$

The reader can see that $g = 3005$ is just the GCD $3x+5$ evaluated at $x = 1000$. Converting 3005 to $3x+5$ is simply writing 3005 in base $\alpha = 1000$. GCDHEU reduced a polynomial GCD computation to computing the GCD of two relatively small integers. The computation is almost trivial! This idea extends in a natural way to polynomials in more than one variable. I show how to do this in Appendix A for the example in Table 2. The GCDHEU algorithm gave Maple an advantage on a critical computation in 1984.

There are difficulties that need to be overcome. One difficulty is how to recover a GCD which has negative integer coefficients. For $\alpha = 1000$ one can modify the base conversion algorithm to recover coefficients in the range $[-500, 500)$ instead of $[0, 1000)$. A second difficulty is how to choose α. If α is too small then the algorithm will not recover the GCD.

But why didn't the Maple team at Waterloo use Brown's algorithm? The Maple professors knew about Brown's algorithm. They showed it to us graduate students in the Symbolic Computation course in 1982. So why didn't they just code Brown's algorithm in Maple?

Brown's algorithm reduces a GCD computation in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ to many GCD computations in $\mathbb{Z}_p[x]$ for many primes p. To implement Brown's algorithm one needs subroutines for polynomial multiplication, division, GCD and interpolation for $\mathbb{Z}_p[x]$. The Maple kernel in 1984 had no C code support for these operations thus Brown's algorithm, when implemented in Maple, was slow. Kernel support for $\mathbb{Z}_p[x]$ would not come until Maple 4.3 in 1989 [17]. On the other hand, the Maple kernel did have C code support for the long integer arithmetic needed by the GCDHEU algorithm. GCDHEU exploited that.

This is not the end of the story on polynomial GCD computation, however. If G is a sparse polynomial there are algorithms which are much faster than GCDHEU and Brown's algorithm. Keith Geddes implemented Paul Wang's EEZGCD algorithm [30] in Maple in 1984. Keith's EEZGCD code is still available today via the Maple command `` `gcd/gcdeh`(A,B) ``. In 2005 it was superceded by an implementation of Zippel's sparse GCD algorithm from [31]. The implementation [14] was done by my PhD student Allan Wittkopf. More recent research with my PhD student Lucas Hu [11] and joint work with Qiao-Long Huang [12] has seen new more efficient GCD algorithms developed that we are integrating into Maple.

3 Problem 2: Evaluation

Maple, Maxima, Derive and Mathematica use expression trees for representing formulas like $x^3 - 3x^2y - 3y^2 + 5$ and $3\sin(2x) - 2\cos(3x)$. To evaluate a formula in Maple and in Maxima one may use assignment. Here is Maple code to evaluate $x + 2y + x\ln x$ at $x = 1$ using assignment.

```
> f := x+2*y+x*ln(x);
                f := x + 2 y + x ln(x)
> x := 1;
                        x := 1
> f;
                       1 + 2 y
```

In the above Maple code, the statement `x := 1;` assigns 1 to x. The statement `f;` evaluates the programming variable f. Maple walks through the expression tree for f to pick up any values of assigned variables to obtain the result $1 + 2y$. The cost of this evaluation is linear in the length of the expression tree for f which is efficient.

In the Maxima code below the statement `x : 1;` assigns 1 to x. The statement `f;` also evaluates the programming variable f. But Maxima evaluates f to $x + 2y + \log(x)$ and no further. One must use the `ev(...)` command in Maxima to do what Maple does.

```
(%i1) f : x+2*y+x*log(x);
(%o1)             2 y + x log(x) + x
(%i2) x : 1;
(%o2)                       1
(%i3) f;
(%o3)             2 y + x log(x) + x
(%i4) ev(f);
(%o4)                   2 y + 1
```

The evaluation model that Maple used was advertised as a very desirable feature. It seemed natural to the Maple designers. So why doesn't Maxima do what Maple does? The reason comes down to a choice between efficiency and usabilty.

Consider the following Maple procedure which goes through the terms of a sum f and counts how many terms have x in them. In the code, g is a local variable, `nops(g)` is the number of elements of the list g which is the number of terms of f.

```
>   NumTermsInx := proc(f,x) local g,c,n,i;
>      # put the terms of f into a list g for processing
>      if type(f,`+`) then g := [op(f)]; else g := [f]; end if;
>      c := 0;
>      n := nops(g);
>      for i to n do
>          if has(g[i],x) then c := c+1; end if;
>      end do;
>      c;
>    end:
> f := x^3-3*x^2*y-3*y^2+5;
```

$$f := x^3 - 3x^2y - 3y^2 + 5$$

```
> NumTermsInx(f,y);
```

$$2$$

If f is a polynomial in m variables with n terms (the example has $m = 2$ and $n = 4$), because `g` is a local variable, the cost of computing `g[i]` is $O(mn)$ in the full evaluation model that Maple used in versions 3.3 and earlier. Thus the total evaluation cost of the for loop is $O(mn^2)$ which is quadratic in n. This Maple code will become very slow for large n. The equivalent code in Maxima would cost $O(mn)$, a factor of n less.

In Maple's model of evaluation, parameters like f always evaluated one level. I changed the model of evaluation in Maple version 4.0 in 1986 so that local variables in Maple procedures also used one level evaluation like Maxima but global variables continued to use full evaluation for backwards compatibility. This single change reduced the time it took to run Maple's entire test suite by over 10% ! This was a surprise for we had been trying for some time to speed up Maple but were previously unsuccessful.

There was concern that this change would break user's Maple code. There was no way to know in advance whether this would be the case. We made the change and waited for complaints from users. None ensued. This model of evaluation, full evaluation for global variables and one level evaluation for parameters and local variables, has not changed since Maple 4.0.

4 Problem 3: Maple's Unique Representation

I was working on Maple's polynomial factorization code in 1985 and testing Maple on factoring the determinant of V_n, the n by n Vandermonde matrix in $x_1, x_2, \ldots, x_n$. For example here is V_3, $\det(V_3)$ and its factorization.

$$V_3 = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix} \qquad \begin{aligned} \det(V_3) &= -x_1^2x_2 + x_1^2x_3 + x_1x_2^2 - x_1x_3^2 - x_2^2x_3 + x_2x_3^2 \\ &= (x_3 - x_2)(x_3 - x_1)(x_2 - x_1) \end{aligned}$$

In general $\det(V_n)$ has $n!$ terms. At $n = 8$ I noticed that computing $\det(V_8)$ in Maple 3.3 seemed slower than it should be. I tracked it down to a horrible bug in Maple's primary hash function.

Consider the monomials $m_1 = xy^2z^3$, $m_2 = z^3y^2x$ and $m_3 = x^3y^2z$. Notice that $m_1 = m_2$ and $m_1 \neq m_3$. If we input the three monomials into Maple using

```
> m1 := x*y^2*z^3;
> m2 := z^3*y^2*x;
> m3 := x^3*y^2*z;
```

Maple creates the monomials in arrays where the first word is used to encode type of the object and also its length. Maple creates

PROD	x	1	y	2	z	3
PROD	z	3	y	2	x	1
PROD	x	3	y	2	z	1

Next Maple simplifies the monomials. Maple is unique among the computer algebra systems in that during simplification, Maple automatically identifies repeated common subexpressions so that each subexpression is stored only once in memory.

After simplification, only one of xy^2z^3 and z^3y^2x is kept, namely, the first one. Maple does this for several reasons. Firstly, to save space. Secondly to speed up simplification. Thirdly so that we can test for equality of two expressions by comparing pointers, that is, using one machine instruction. How is this done?

Maple maintains an internal hash table of pointers to all live expressions in the Maple session. This hash table is called the `simpl` table. It is an array of buckets where each bucket is an array of hash values and pointers to objects. Maple uses hashing to test equality of formulas. If two products are equal up to ordering of their factors, for example, z^3y^2x and xy^2z^3, they must have the same hash value. Thus the hash function must be commutative on the factors x, y^2, z^3 of a product. If they are not equal, the hash function should be different. The horrible bug is that Maple's hash function was commutative on $x, 1, y, 2, z, 3$ so that $x^1y^2z^3$ and $x^3y^2z^1$ had the same hash value! Okay, so it's a bug. Why am I calling it a *horrible* bug?

Consider V_4 the 4 by 4 Vandermonde matrix and its determinant D_4. In Table 4 I've grouped the 24 terms of D_4 into four groups, those which don't have x_1, x_2, x_3 and x_4.

Group 1: missing x_1	$-x_2^3x_3^2x_4,\ x_2^3x_3x_4^2,\ x_2^2x_3^3x_4,\ -x_2^2x_3x_4^3,\ -x_2x_3^3x_4^2,\ x_2x_3^2x_4^3$
Group 2: missing x_2	$x_1^3x_3^2x_4,\ -x_1^3x_3x_4^2,\ -x_1^2x_3^3x_4,\ x_1^2x_3x_4^3,\ x_1x_3^3x_4^2,\ -x_1x_3^2x_4^3$
Group 3: missing x_3	$-x_1^3x_2^2x_4,\ x_1^3x_2x_4^2,\ x_1^2x_2^3x_4,\ -x_1^2x_2x_4^3,\ -x_1x_2^3x_4^2,\ x_1x_2^2x_4^3$
Group 4: missing x_4	$x_1^3x_2^2x_3,\ -x_1^3x_2x_3^2,\ -x_1^2x_2^3x_3,\ x_1^2x_2x_3^3,\ x_1x_2^3x_3^2,\ -x_1x_2^2x_3^3$

Table 3: Terms in $\det(V_4)$, the 4×4 Vandermonde matrix

Observe that each group has $n!/n = (n-1)!$ terms and in each group each monomial is in the same variables and the exponents are a permutation of $1, 2, 3$. Thus every monomial in each group in Table 4 has the same hash value! This means searching Maple's `simpl` table for one of these monomials is $O((n-1)!)$ instead of $O(1)$!!

It was not difficult to fix the problem once I'd located it. The fix was included in Maple 4.0. What I learned from this is that even though one may be an expert in algorithms, as the author of Maple's hash function was, it is easy to have blind spot and miss something. After implementing an idea, and getting the tests to pass, it is tempting to just move on to the next programming task. Instead, one should take time to stare at one's code, and ask oneself, am I missing something? Have I done something stupid?

Another lesson I learned is the following. Maple 3.1 was supposed to be "the efficiency version". The Maple developers generated tables of counts for each function in the Maple kernel for the Maple test suite. The idea was to determine which kernel functions were used most often and try to speed them up. Seems reasonable. But this search for efficiency was a flop. No significant improvement in efficiency was made. One reason for this is that test suites test primarily for algorithm correctness and not algorithm efficiency. The Maple test suite no large tests because large tests take a long time to run and the Maple test suite was run daily.

5 Problem 4: Quadratic Algorithms

I have coined a phrase. I say a programmer of an algorithm which is supposed to be $O(n)$ or $O(n \log n)$ commits an *asymptotic blunder* if their implementation is $O(n^2)$ or worse.[1]

Here is an example. I ask the students who take my Computer Algebra course at Simon Fraser University to program an FFT based multiplication algorithm that multiplies two polynomials $a \times b$ to get the product c of degree d. The FFT multiplication algorithm has complexity $O(d \log d)$. The whole point of using the FFT is to reduce the cost from $O(d^2)$ to $O(d \log d)$. If the student's implementation is $O(d^2)$ then the student has committed an asymptotic blunder; they have ruined the implementation! How could they do that? Here is one way. Suppose after applying the FFT algorithm, the coefficients of the product c are stored in an array C, that is C_i is the coefficient of x^i in c. To convert from the array C to the polynomial c my students would code

```
> c := 0;
> for i from 0 to d do
>    c := c + C[i]*x^i;
> od;
> c;
```

This code is $O(d^2)$ because it constructs and simplifies all intermediate polynomials, namely

$$C_0$$
$$C_0 + C_1 x$$
$$C_0 + C_1 x + C_2 x^2$$
$$\ldots$$
$$C_0 + C_1 x + C_2 x^2 + \cdots + C_{d-1} x^{d-1}.$$

This complexity problem is not unique to Maple. This code block if implemented in Magma, Mathematica, Maxima or Singular will also be quadratic in d. We can forgive the students for not realizing what they do; the cost of this for loop depends on how the sum is computed which is hidden from view. Note, the Maple programmer can construct $c(x)$ from C in $O(d)$ by using the Maple command `c := add( C[i]*x^i, i=0..d )` .

A question. Do todays computer algebra systems have $O(n^2)$ algorithms which should be $O(n)$ or $O(n \log n)$? They do! Here is one instance. Suppose we want to read a polynomial from a text file into the computer algebra system. For example, the text file may contain

```
f := 3*x^3-2*w*y*z-5*x*y^2*z+w^3-2*x*y*z-5*y*z^2*x+z^3-2*w*x*y;
```

[1] A more general definition would replace $O(n)$, $O(n \log n)$ and $O(n^2)$ with $O(n^c)$, $\tilde{O}(n^c)$ and $O(n^{c+1})$ respectively.

If the polynomial is in m variables and has t terms, including the cost for sorting terms, I think reading should take $O(mt \log t)$ but in Maxima and Magma it is $O(mt^2)$, that is, quadratic in t.

Maple and Singular used to have the same problem! I noticed the problem in Maple 3.3 and fixed it for Maple 4.0. I noticed the problem in Singular 3.1. I wanted to time Singular's polynomial factorization code for factoring determinants of matrices of polynomials but Singular's determinant algorithm took too long. So I computed the determinants in Maple but then reading them in to Singular also took too long! Table 5 gives the time it takes for Maxima 5.45.0, Magma V2.28-19, Maple 2024, and Singular 3.4.1 to read in a polynomial f with t terms in 8 variables with 2 digit integer coefficients and $\deg(f) \le 20$.

t	Maxima	(space)	Magma	Maple	(.m format)	Singular
1000	0.27		0.03	0.005		0.003
2000	0.77		0.06	0.010		0.006
4000	2.93		0.12	0.019		0.012
8000	11.92	(10.8gb)	0.43	0.038	(0.003)	0.024
16000	47.72	(43.4gb)	1.47	0.075	(0.009)	0.049
32000	199.27	(171gb)	Seg fault	0.149	(0.012)	0.099
64000	NA	NA	Seg fault	0.319	(0.019)	0.200
128000	NA	NA		0.770	(0.034)	0.455
256000	NA	NA		1.799	(0.070)	0.924

Table 4: Time (in CPU seconds) for reading in a polynomial f from a file in text format with t terms. NA = Not Attempted.

The timings in Table 5 show that Maxima and Magma take quadratic time to read a polynomial with t terms but Maple and Singular appear to take linear time. In Table 5 column (.m format) is the time to load the polynomial in Maple's internal format which is linear time as the terms are already sorted. As the reader can see, reading a polynomial with 256,000 terms is not a difficult problem! It takes Singular 3.4.1 under one second.

6 Problem 3: Representation of Small Integers

Computer algebra systems support integers of arbitrary length. Before GMP (the Gnu Multiprecision Package) existed, each computer algebra system had its own representation for long integers and its own set of arithmetic routines for them. Long integers were stored in arrays. For example the array $\boxed{a_0 \mid a_1 \mid a_2 \mid a_3}$ stores the integer $a_0 + a_1B + a_2B^2 + a_3B^3$ where B is the base of the integer and $0 \le a_i < B$. Maple used the decimal base $B = 10^4$ on a 32 bit computer but most computer algebra systems used $B = 2^{32}$. But how are small integers stored? Prior to Maple 6, Maple used the same representation for small and large integers. For example, Maple's 4.0's representation for the integers 123456789 and -6 was $\boxed{\text{INTPOS} \mid 6789 \mid 2345 \mid 1}$ and $\boxed{\text{INTNEG} \mid 6}$. The first word, the header word, encodes the type and length of the object.

Suppose we want to multiply the monomials x^2y^3 and x^2y^4 to get x^4y^7. Prior to Maple 6, Maple stored the monomials like this.

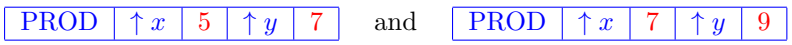
$$\boxed{\text{PROD} \mid \uparrow x \mid \uparrow 2 \mid \uparrow y \mid \uparrow 3} \quad \text{and} \quad \boxed{\text{PROD} \mid \uparrow x \mid \uparrow 3 \mid \uparrow y \mid \uparrow 4}.$$

Here $\uparrow 3$ means what is stored is a pointer to the INTPOS object 3. To add two exponents Maple created an INTPOS object for their sum. After creating the INTPOS object Maple searches

the `simpl` table to see if it already exists. This makes monomial multiplication *very* slow, and as a consequence, polynomial multiplication and division are slow.

The solution is "immediate integers". Gaston Gonnet added immediate integers to Maple in the 1990s. This lead to a significant speedup of Maple. But what are immediate integers?

On a 64 bit computer, words have 8 bytes so if a pointer is word aligned, it's least significant 3 bits are zero bits. If an integer x satisfies $-2^{62} < x < 2^{62}$ Maple 2024 encodes it as $2x+1$ so that it is odd. Otherwise Maple 2024 stores a pointer (which is even) to an INTPOS or INTNEG object. In the two PROD objects above, what is now stored in the exponent fields is 5, 7 and 9 for the exponents 2, 3, and 4, respectively, like this.



Maple 2024's representation for the monomials x^2y^3 and y^3z^4.

Maple 2024 can distinguish between a small integer, and a pointer to a large integer, by inspecting the least significant bit. To add the encodings $X = 2x+1$ and $Y = 2y+1$ of two integers x and y we just compute $X + Y - 1$ and check for overflow. This change means that no space is allocated when adding exponents in monomial multiplication, and all integers in the range $-2^{62} < x < 2^{62}$ are no longer stored in Maple's `simpl` table.

Immediate integers are first mentioned in the release notes for Maple 6 which was released in late 1999. See `?updates,Maple6,language`. In [27], Juho Snellman traces the immediate integer idea back to early versions of Lisp. He concludes and I quote him here.

> By 1964, the M 460 LISP implementation had arrived at the general solution of using pointers to invalid parts of the address [odd word pointers for example] space for storing immediate data, but user-accessible integers were still boxed; the only use for the unboxed integers was as an internal building block. In 1966 PDP-6 LISP applied the idea of tagged immediate data to tiny positive integers [12 bits only], and the PDP-1 based BBN LISP took the idea to the logical conclusion, and allowed immediate storage of integers of almost the full machine word.

So immediate integers predate Maple by 15 years. Why didn't we put them into Maple earlier? We were C programmers. We had little knowledge of Lisp.

Immediate floats were added to Maple 2025 by Paul Demarco. Maple 2025 stores a 61 bit signed integer x as $4x+3$ so the least significant two bits are 11. Maple 2025 stores a double precision float x with $-2^{256} < x < 2^{254}$ with 01 as the least significant two bits.

7 Problem 6: Numerical Linear Algebra

Maple, like all early computer algebra systems, was neither designed nor intended to be an engine for large numerical linear algebra computations. Prior to Maple 6, Maple was very slow at numerical linear algebra. There are three reasons for this.

1. Maple used a software representation for floating point numbers so that it could support high precision floating point computations. But software floating point arithmetic is 100 times slower than hardware floating point arithmetic.

2. Maple used a hash table to store arrays, vectors and matrices. This representation is space expensive for small vectors and matrices. It is also cache unfriendly for large matrices.

3 The numerical linear algebra routines in Maple's `linalg` package were written in the Maple programming language which is interpreted, like Python.

Maplesoft wanted to get into the engineering market. To do that Maple had to be competitive with Matlab for numerical linear algebra. So for Maple 6, Maplesoft ditched the hash table representation for arrays, vectors, and matrices and replaced it with the standard dense array representation, and replaced the slow interpreted codes with the compiled C codes from the NAG numerial library. The new array type `rtable` can store 8 byte signed integers and 8 byte floating point numbers (which are IEEE double precision floats), as well as (pointers to) other types of objects. For two dimensional arrays and matrices, both column major and row major orderings are supported. Here is an example

```
> x := evalf(Pi);
                                        x := 3.141592654

> dismantle(x);
FLOAT(3): 3.141592654
   INTPOS(2): 3141592654
   INTNEG(2): -9

> A := Matrix( [[3,x],[2/3,Pi]], datatype=float[8] );
```

$$A := \begin{bmatrix} 3. & 3.14159265400000 \\ 0.666666666666667 & 3.14159265358979 \end{bmatrix}$$

```
> op(3,A); # storage options
      datatype = float[8], storage = rectangular, order = Fortran_order, shape = []

> y := A[1,2];
                                      y := 3.14159265400000

> dismantle(y);
HFLOAT(2): 3.141592654

> z := 2*x+y;
                                      z := 9.42477796200000

> dismantle(z);
HFLOAT(2): 9.424777962
```

The example shows that x is a software float and y is a hardware float. Hardware floats were also new in Maple 6. Thus Maple 6 has two representations for floats, hardware floats and software floats. Notice that hardware floats are contagious, that is, z, a result of arithmetic between exact rationals, software floats and hardware floats is a hardware float.

Table 7 shows timings comparing the old linear algebra package `linalg` that uses software floats and is coded in Maple verses the new linear algebra package `LinearAlgebra` that uses hardware floats and compiled code from the NAG numerical library. In the benchmark, A and B are n by n matrices and b is a vector of dimension n for $n = 100$ and $n = 200$. The entries of A, B and b are randomly generated 10 digit floats on $[0, 1)$.

The first row of times is for reducing the augmented matrix $[A|b]$ to reduced row Echelon form. I don't know why the LinearAlgebra library is not fast at this operation. The fifth row is for computing the (complex) eigenvalues of A. As the reader can see, the new design is between 100 and 1000 times faster on this benchmark. Interestingly Macsyma supported both hardware floats and software floats but Macsyma did not use a compiled numerical linear algebra library.

	linalg		LinearAlgebra	
	$n = 100$	$n = 200$	$n = 100$	$n = 200$
rref $[A\|b]$	1.27	12.85	0.746	4.05
solve $Ax = b$	0.75	8.35	0.0005	0.0042
multiply AB	1.44	15.18	0.0003	0.0142
singularvalues of A	1.62	18.06	0.0043	0.0150
eigenvalues of A	0.86	6.81	0.0182	0.0748

Table 5: CPU timings (in second) for linear algebra operations in Maple

8 Problem 7: Polynomial Division Algorithms

Multiplying and dividing polynomials in more than one variable are core operations in a computer algebra system. Their speed will have an impact on the overall efficiency of the system. Suppose we have two polynomials f and q in n variables $x_1, x_2, ..., x_n$ with integer coefficients. Suppose and we want to compute $h = f \times q$ in expanded form.

If $n > 1$ Maple, Magma, Maxima, and Singular all use the classical (high school) multiplication algorithm. They multiply all pairs of monomials and coefficients and then sort the terms in the product to add coefficients of like terms. For example

$$\begin{aligned}(x_1x_2 + 2x_1 + x_2)(x_1 + 2x_2) &= x_1^2x_2 + 2x_1^2 + x_1x_2 + 2x_1x_2^2 + 4x_1x_2 + 2x_2^2 \\ &= x_1^2x_2 + 2x_1^2 + 2x_1x_2^2 + 5x_1x_2 + 2x_2^2.\end{aligned}$$

Let $\#f$ denote the number of terms of the polynomial f. This multiplication does $\#f\#q$ monomial multiplications and $\#f\#q$ coefficient multiplications. The sorting cost depends on the multiplication algorithm. If we multiply all $\#f\#q$ terms out and sort them the sorting cost will be $O(\#f\#q(\log \#f + \log \#q))$ monomial comparisons. A divide and conquer multiplication yields a sorting cost of $O(\#f\#q \log \min(\#f, \#q))$ monomial comparisons.

Now suppose we want to divide h by f to get the quotient $q = h/f$. The classical division algorithm for $h \div f$ will compute $h - fq$ where q is the quotient. Most of the work is computing fq. Let $q = q_1 + q_2 + q_3 + \dots$ be the terms of q. The classical division algorithm initializes $r = h$ then computes q_1 then sets $r = r - q_1f$. Then it computes q_2 then sets $r = r - q_2f$. It repeats this until either the division fails or $r = 0$. So it computes

$$(((h - q_1f) - q_2f) - q_3f) - \dots.$$

Each subtraction $r - q_if$ can be done efficiently using a merge with complexity $O(\#r + \#f)$ monomial comparisons. The problem is that if q is large compared with f, each subtraction $r - q_if$ goes through all the terms of r and the sorting cost is $O(\#f\#q^2)$ monomial comparisons. In [13] Johnson used a binary heap to efficiently sort the terms in the product fq using $O(\#f\#q \log \#q)$ monomial comparisons. In [18], Monagan and Pearce reduced this to $O(\#f\#q \min(\log \#f, \log \#q))$ monomial comparisons.

Let $f = 1 + w^2 + x^2 + y^2 + z^2$ and $p = f^n$. Tables 6 and 7 time Maxima, Magma, Maple and Singular on polynomial multiplication $r = p(p + 1)$ and two divisions, for different n. The division $r \div f^2$ should be much faster than $r \div p$ since $\#f^2 \ll \#p$. For the Maxima timings I used `ratexpand(p*(p+1))` instead of `expand(p*(p+1))` because `expand` in Maxima is slow; it's over 10 times slower than `ratexpand`.

			Maxima 5.45.0			Magma 2.28-19		
n	$\#p$	$\#r$	$r = p(p+1)$	$r \div p$	$r \div f^2$	$r = p(p+1)$	$r \div p$	$r \div f^2$
15	3876	46376	6.562	4.658	0.548	0.46	0.91	1.17
17	5985	73815	13.802	10.628	0.904	1.13	2.27	2.86
19	8855	111930	28.729	32.442	1.759	3.00	5.41	6.83
21	12650	163185	58.847	54.513	2.207	6.98	11.91	14.28
23	17550	230300	166.48	186.15	4.275	14.12	23.97	28.60
25	23751	316251	297.17	373.00	6.407	28.37	44.42	51.84

Table 6: Timings (in CPU seconds) for polynomial arithmetic

	Maple 2024					Singular 3.4.1		
n	$r = p(p+1)$	1 core	$r \div p$	1 core	$r \div f^2$	$r = p(p+1)$	$r \div p$	$r \div f^2$
15	0.019	0.009	0.040	0.098	0.003	0.019	0.110	0.009
17	0.035	0.218	0.082	0.242	0.007	0.246	0.268	0.015
19	0.064	0.501	0.064	0.551	0.023	0.548	0.598	0.027
21	0.129	1.16	0.129	1.16	0.036	1.127	1.234	0.048
23	0.204	2.06	0.204	2.28	0.057	2.209	2.442	0.072
25	0.318	4.21	0.318	4.21	0.099	4.069	4.444	0.102

Table 7: Timings (in CPU seconds) for polynomial arithmetic

Notice that unlike Maxima, the Magma timings for $r \div f^2$ are slower than the timings for $r \div p$ even though f^2 is a relatively small polynomial.

Maple is using parallel algorithms from [19, 26]. The timings for Maple are real times. The timings in columns $r = p(p+1)$ and $r \div p$ are parallel timings for a 24 core CPU and the timings for 1 core follow. The Maple parallel timings for $r = p(p+1)$ and $r \div p$ are about 10 times faster than the Maple 1 core timings, which are comparable to the Singular timings which are about 5 to 10 times faster than Magma which is 10 times faster than Maxima.

9 Problem 8: Which Polynomial Representation is Best?

At the 1984 Macsyma User's Conference in Schenectady New York, David Stoutemyer gave a talk entitled "Which polynomial representation is best?". Let me here first show three different polynomial representations, namely, sum-of-terms representations, recursive sparse representations and the recursive dense representation. Then I will return to Stoutemyer's question.

9.1 Distributed Representations

The sum-of-terms representation is used by Magma, Maple, Mathematica, Maxima, and Singular. Singular's sum-of-terms representation for the polynomial $f = 9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$ is depicted in Figure 1. Singular uses a linked list of terms. Here the terms are ordered in decreasing graded lexicographical order. Singular knows f is a polynomial in the ring $\mathbb{Z}[x, y, z]$ so the variables do not appear explicitly in the representation.

Maple also uses a sum-of-terms representation. Maple's sum-of-terms representation for the polynomial $f = 9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$ is depicted in Figure 2. In Maple the terms are

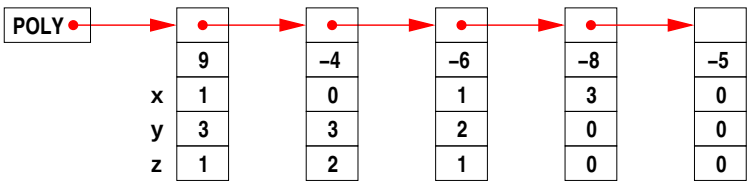


Figure 1: Singular's sum-of-products representation for $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$

not sorted ordered in a monomial ordering. They are sorted by the address of the PROD objects (the monomials). Mathematica's representation is very similar to Maple's. The main difference is that Mathematica sorts terms by lexicographical order.

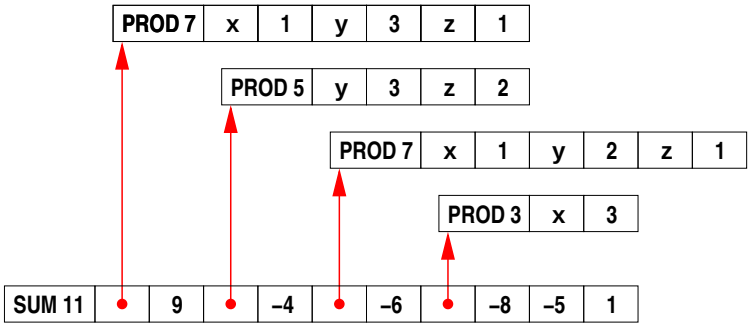


Figure 2: Maple's sum-of-terms representation for $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$

There is a subtle difference between the sum-of-terms representation used by Magma and Singular and the sum-of-terms representation used by Maple, Mathematica, and Maxima, namely, the latter use a sparse representation for the monomials, that is, no zero exponents appear in their representations. In Singular the term $-8x^3$ is stored as $\boxed{-8 \mid 3 \mid 0 \mid 0}$ but in Maple it's stored as $\boxed{\text{PROD} \mid x \mid 3}$. If f is a linear polynomial in n variables where n is large, then Maple uses $O(n)$ space to store f but Singular needs $O(n^2)$ space. Thus a linear system of n linear polynomials in n unknowns requires $O(n^2)$ space in Maple but $O(n^3)$ space in Singular.

9.2 Recursive Representations

The recursive representation is used by Derive, Fermat, Pari, Reduce and Trip. Trip's recursive sparse representation for the polynomial $f = 9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$ is depicted in Figure 3. To understand the representation it is helpful to think of f as a polynomial in the ring $\mathbb{Z}[z][y][x]$, that is, $f = (-8)x^3 + ((9z)y^3 + (-6z)y^2)x + ((-4z^2)y^3 + (-5))$.

Algorithms for $+, -, \times, \div$ in the recursive representation are different from those in the sum-of-terms representation. In the sum-of-terms representation there is a significant cost for processing monomials. Storage must be allocated for each monomial. In the recursive representation there are no monomials. Algorithms are simpler because they are univariate.

Pari uses a recursive dense representation for all polynomials. Pari's recursive dense representation for the polynomial $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$ is depicted in Figure 4.

At the 1984 Macsyma User's Conference, Stoutemyer observed that the recursive dense representation was the fastest for polynomial multiplication and division which was a surprise. Stoutemyer

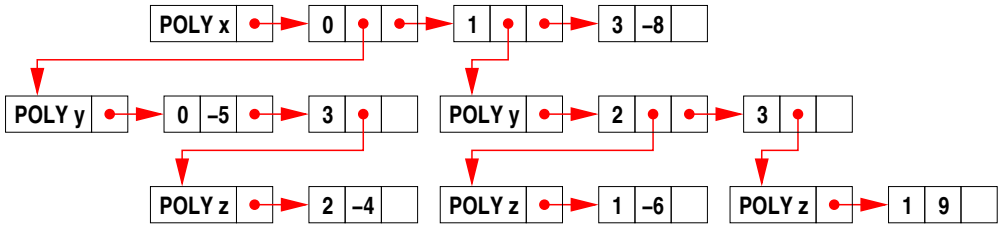


Figure 3: Trip's sparse recursive representation for $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$

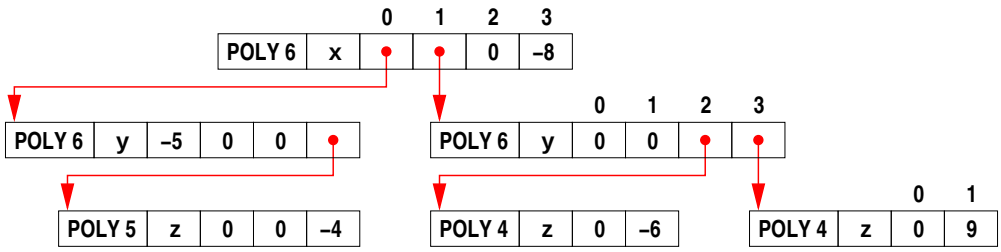


Figure 4: Pari's recursive dense representation for $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$

also observed that recursive sparse was faster than the sum-of-terms representation. These observations were later confirmed by Richard Fateman in 2003. Fateman observed that the recursive representation used by Pari was fastest on his benchmarks and amongst the computer algebra systems that use the sum-of-terms representation, Maple, Mathematica and Macsyma were slower than Singular and Magma.

In 2022 I asked Henri Cohen, the author of Pari, why he chose the recursive dense representation. Henri told me it was the first representation for multivariate polynomials that he thought of trying and when it seemed to work well, he kept it.

9.3 Speeding up Sum-Of-Terms Representations

Fateman's benchmark revealed that the sum-of-terms representation used by Maple, Mathematica, and Maxima is the slowest representation, over 5 to 13 times slower than Singular's sum-of-terms representation which was 2.6 times slower than Pari. Why is that? Look again at Maple's general representation shown in Figure 2. Suppose we want to multiply two monomials wxy^2 times x^2yz^2 in Maple. The two monomials would be stored as PROD arrays of length 7 words. They look like

PROD	w	1	x	1	y	2

and

PROD	x	2	y	1	z	2

.

To multiply them Maple concatenates them to form the product

PROD	w	1	x	1	y	2	x	2	y	1	z	2

,

which requires a storage allocation. Maple then sorts on the variables to get

PROD	w	1	x	1	x	1	y	2	y	2	z	2

,

adds up like exponents and shrinks the array to be

PROD	w	1	x	2	y	4	z	2

. Then Maple hashes the resulting monomial and looks up the `simpl` table to test if it already exists in Maple. No wonder Maple is slow at multiplying polynomials! How can we speed Maple up?

ALTRAN [10] is one of the oldest computer algebra systems. It was developed at Bell Labs in the 1960s to do polynomial and rational function arithmetic. ALTRAN uses a sum-of-terms representation for polynomials but it packs monomials into words. One reason Altran did this is because in the 1960s, memory was scarce. But packing monomials also speeds up monomial arithmetic. In the 1980s and 1990s, as the amount of RAM on computers grew from kilobytes to megabytes, packing monomials did not seem necessary anymore.

In [20, 21] Roman Pearce and I designed a new polynomial representation for Maple for polynomials in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$. We pack monomials into 64 bit words using a graded monomial ordering. Figure 5 depicts the POLY representation for the polynomial $f = 9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$.

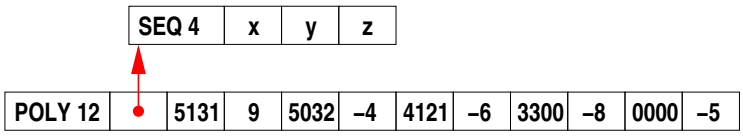


Figure 5: Maple's POLY representation for $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$

If f is a polynomial in $n > 1$ variables with integer coefficients, Maple uses $b = \lfloor 64/(n+1) \rfloor$ bits for each exponent. In the remaining $64 - nb$ bits Maple stores the total degree of the monomial. For example consider the first monomial $x^1y^3z^1$ in our polynomial f which is shown as 5131 in Figure 9.3. It is encoded as the unsigned 64 bit integer $5B^3 + 1B^2 + 3B + 1$ where $B = 2^{16}$. If f is a polynomial in one variable with integer coefficients, Maple uses all 64 bits for the degree.

If all monomials in a polynomial $f \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$ can be packed into 64 bits using this encoding, and $\deg(f) > 1$, and f has at least two terms, then Maple uses the POLY representation, otherwise it uses the sum-of-terms representation as depicted in Figure 2. Conversions between representations when necessary are automatic and invisible to the Maple user. The Maple user can find out which representation is being used using the `dismantle` command. For example

```
> f := 3*x^2-5*x*y;
```

$$f := 3x^2 - 5xy$$

```
> dismantle(f);
POLY(6)
   EXPSEQ(3)
      NAME(4): x
      NAME(4): y
   DEGREES(HW): ^2 ^2 ^0
   INTPOS(2): 3
   DEGREES(HW): ^2 ^1 ^1
   INTNEG(2): -5
> g := 3*x-5*y+4*z;
```

$$g := 3x - 5y + 4z$$

```
> dismantle(g);
SUM(7)
   NAME(4): x
   INTPOS(2): 3
   NAME(4): y
   INTNEG(2): -5
   NAME(4): z
   INTPOS(2): 4
```

9.4 Advantages of the POLY representation

A first advantage is that the POLY representation reduces the space needed to store a polynomial. For the polynomial $f = 9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$ the space needed by POLY is 16 words (see Figure 5). Compare this with 33 words in Maple's sum-of-terms representation (see Figure 2).

A second advantage is that the POLY representation eliminates memory management for the monomials. They are not stored in Maple's `simpl` table (See Section 3).

A third advantage is that to sort terms in the POLY representation, we can compare monomials using a 64 bit unsigned integer comparison, that is, with one machine instruction. This is the fastest it could be.

A fourth advantage is that monomial multiplication becomes a 64 bit unsigned integer addition, that is, one machine instruction. Again, this is the fastest it could be. Of course overflow could occur. However, observe if we want to multiply $f \times g$, no overflow can occur if $\deg(f)+\deg(g) < 2^b$, and, because the total degree of f in the POLY representation is stored in the first term, this test is $O(1)$.

The reader may wonder why POLY doesn't use pure lexicographical order so that we don't have to store the total degree of each monomial which would mean we have more bits available for the exponents. The reason is because of polynomial division. If one uses lexicographical order to divide $f \div g$, and $\deg(g, x_i) \le \deg(f, x_i)$ for $1 \le i \le n$, intermediate degrees can be greater than those of f and one would have to check for overflow. In graded lexicographical order, this cannot happen. Thus if $\deg f < 2^b$ then the entire division algorithm can execute with no test for overflow.

How many polynomials can be packed in the POLY representation? The first row in Table 7 shows the number of bits $b = \lfloor (n+1)/64 \rfloor$ for exponents, for different n. The second row shows $64 - nb$, the number of bits available for the total degree.

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
exponents	21	16	12	10	9	8	7	6	5	5	4	4	4	4	3	3	3	3	3
degree	22	16	16	14	10	8	8	10	14	9	16	12	8	4	16	13	10	7	4

Table 8: Number of bits used for exponents and the total degree in POLY for n variables.

The 64 bit word length of today's computes is big enough so that POLY handles a wide range of polynomials in practice. 32 bits would not be big enough. We did consider using 128 bits but did not implement it. Currently POLY is also limited to integer coefficients. An experiment we did try was to support rational coefficients and algebraic number coefficients but the coding effort was deemed too great for the speedup that we would have achieved.

The POLY representation first appeared in Maple 17 in 2013 and it was first announced at the Asian Symposium on computer mathematics in Beijing, October 2012. At that time the total degree of f was also restricted to fit in $b = \lfloor 64/(n+1) \rfloor$ bits. Joris van der Hoeven asked me why we did not use all $64 - nb$ bits for the total degree so that we could store more polynomials in the POLY representation. The answer I gave was that it would complicate the code. I changed my mind when I was once again computing determinants of Vandermonde matrices.

If V_n is the $n \times n$ Vandermonde matrix in $x_1, x_2, \ldots, x_n$ then the total degree $\deg(V_n) = \sum_{i=1}^{n-1} i = n(n-1)/2$. If we use only $b = \lfloor 64/(n+1) \rfloor$ bits for the total degree, V_9 fits in POLY but V_{10} does not because for $n = 9$ we have $b = 6$ bits and $\deg(V_9) = 45$ but for $n = 10$ we have $b = 5$ bits and $\deg(V_{10}) = 55$. By using all $64 - nb$ bits for the total degree, $V_{10}, V_{11}, V_{12}, V_{13}$ and V_{14} all fit in POLY, a large increase in the range of POLY. Using all $64 - nb$ bits for the total degree in POLY was added for Maple 18.

9.5 Benchmark for POLY

The timings in Table 9 compare the speed of Maple on multiplying three polynomials $f_1 \times f_2 \times f_3$ in n variables and factoring their products. The polynomials f_1, f_2 and f_3 have 100 terms in n variables and 6 digit coefficients. The code for generating the input polynomials and timings is given in Appendix B.

Column Maple 16 is before POLY was added to Maple. Column Maple 17 is for after POLY was added to Maple. Column Maple 18 is after further improvements to POLY. For Maple 2019 the polynomial factorization algorithm was changed. Previously Maple used Paul Wang's multivariate Hensel lifting from [29, 7]. Since Maple 2019, Maple uses Monagan and Tuncer's algorithm from [22, 23, 24] which uses sparse polynomial interpolation.

	Maple 16		Maple 17		Maple 18		Maple 2019	
n	expand	factor	expand	factor	expand	factor	expand	factor
5	0.158s	25.15s	0.029s	8.61s	0.032s	8.87s	0.034s	0.80s
6	0.504s	86.30s	0.058s	29.29s	0.048s	34.53s	0.052s	1.69s
7	0.719s	4.35m	0.050s	1.45m	0.055s	69.38s	0.063s	3.02s
8	1.07s	8.06m	0.049s	3.90m	0.058s	2.52m	0.066s	4.16s
9	0.96s	10.47m	0.049s	4.91m	0.058s	3.83m	0.068s	5.05s
10	1.13s	16.10m	0.049s	12.26m	0.058s	3.61m	0.070s	5.58s
11	1.25s	21.47m	0.048s	9.07m	0.058s	19.96m	0.069s	6.70s
12	1.35s	37.54m	1.17s	38.84m	0.98s	21.01m	1.99s	1.33m
13	1.41s	76.87m	1.22s	109.79m	1.11s	25.56m	2.47s	1.57m
14	1.20s	98.50m	1.30s	3.17h	1.10s	30.17m	4.95s	2.29m
15	1.24s	58.59m	1.32s	112.4m	1.27s	47.24m	1.68s	2.52m

Table 9: CPU timings for multiplying and factoring polynomials in n variables

Polynomial multiplication is about 20 times faster for $5 \le n \le 11$ in Maple 17 than in Maple 16. This is due entirely to POLY. Factorization is 2 to 3 times faster for $5 \le n \le 11$. This is also due to POLY. The product $f_1 \times f_2 \times f_3$ at $n = 12$ variables overflows the 64 bit monomials so Maple uses the old sum-of-products representation so there is no gain for $12 \le n \le 15$.

The factorization timings for Maple 16, 17, and 18 are increasing exponentially as a function of n. Paul Wang's multivariate Hensel lifting is, in general, exponential in n. In contrast the factorization timings for Maple 2019 are linear in n (the jump at $n = 12$ is because POLY overflows). The Monagan and Tuncer polynomial factorization algorithm is random polynomial time.

10 Factoring Determinants of Polynomial Matrices.

We consider three families of matrices of polynomials, V_n the n by n Vandermonde matrix in $x_1, x_2, \ldots, x_n$, T_n the symmetric Toeplitz matrix in $x_1, x_2, \ldots, x_n$ and C_n the cyclic matrix in $x_1, x_2, ..., x_n$. For each matrix we compute and factor it's determinant. Note that $\det(V_n)$, $\det(T_n)$ and $\det(C_n)$ are all polynomials in $\mathbb{Z}[x_1, x_2, \ldots, x_n$.

We time Maxima version 5.45.0, Maple 2024, Magma V2.28-19 and Singular 3.4.1. Timings were obtained on a 24 core Intel Xeon Gold 6342 CPU running at 2.8/3.5 GHz base/turbo.

10.1 Vandermonde matrices

The n by n Vandermonde matrix Vn is defined by $V_{nij} = x_i^{j-1}$ for $1 \le i \le n$, $1 \le j \le n$. The factorization of $\det(V_n)$ is $\prod_{1 \le i < j \le n}(x_j - x_i)$. For example

$$V_3 = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix} \qquad \begin{aligned} \det(V_3) &= -x_1^2x_2 + x_1^2x_3 + x_1x_2^2 - x_1x_3^2 - x_2^2x_3 + x_2x_3^2 \\ &= (x_3 - x_2)(x_3 - x_1)(-x_1 + x_2). \end{aligned}$$

To factor $\det(V_3)$ Maple choose a main variable, say x_1, writes $D_3 = \det(V_3)$ as a polynomial in $\mathbb{Z}[x_2, x_3][x_1]$ to get

$$D_3 = (x_3 - x_2)x_1^2 + (x_2^2 - x_3^2)x_1 + (x_2x_3^2 - x_2^2x_3)x^0,$$

computes the GCD of the coefficients in x_1, that is, computes

$$c = \gcd(x_3 - x_2, x_2^2 - x_3^2, x_2x_3^2 - x_2^2x_3) = x_3 - x_2.$$

If $c \neq 1$ Maple factors c and $p = D_3/c = x_1^2 - x_1x_2 - x_1x_3 + x_2x_3$ recursively. To factor p, Maple computes the GCD of the coefficients in each variable to try to identify a factor cheaply. Here for x_1, we have $\gcd(1, -x_2 - x_3, x_2x_3) = 1$ but for x_2 we get $\gcd(x_3 - x_1, x_1^2 - x_1x_3) = x_3 - x_1$ and obtain the factorization $p = (x_3 - x_1)(x_2 - x_1)$. The entire factorization of $\det(V_n)$ is done with a sequence of GCD computations. Table 10.1 shows the time it takes Maple, Magma, Singular and Maxima to compute and then factor $\det(V_n)$. We conjecture that the reason Magma is so slow is because of its poor polynomial division algorithm (See Section 8).

		Maple 2024		Magma 2.29-19		Singular 3.4.1		Maxima 5.45.0	
n	#det	det	factor	det	factor	det	factor	det	factor
7	5040	0.004	0.006	0.01	0.04	0.002	0.018	1.153	1.99
8	40320	0.024	0.036	0.02	0.50	0.029	0.180	17.56	44.99
9	362880	0.144	0.553	0.18	8.90	0.538	2.163	255.66	875.83
10	3628800	1.59	11.18	3.20	518.78	10.053	34.405	OM	NA
11	39916800	19.39	252.60	34.26	22,739.0	131.202	8,851.09	NA	NA
12	479001600	253.80	5334.6	NA	NA	NA	NA	NA	NA

Table 10: Timings (in CPU seconds) to compute and factor $\det(V_n)$. OM = Out of Memory. NA = Not Attempted.

Maple ran out of memory when trying to compute $\det(V_{13})$ because the determinant is too big. The POLY representation (see Figure 5) needs 2 words of memory per term to store $\det(V_{13})$ for a total of $8 \cdot 2 \cdot 13! = 99.6$ gigabytes. Maple's sum-of-products representation (see Figure 2) would need 25 words per monomial for a total of $8 \cdot 25 \cdot 13! = 1.245$ terabytes.

10.2 Symmetric Toeplitz matrices

Let T_n be the n by n symmetric Toeplitz matrix with $T_{nij} = x_{|i-j|+1}$ for $1 \le i \le n$, $1 \le j \le n$. For $n \ge 2$, $\det(T_n)$ has two irreducible factors. For example

$$T_3 = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_2 & x_1 & x_2 \\ x_3 & x_2 & x_1 \end{bmatrix} \qquad \begin{aligned} \det(T_3) &= x_1^3 - 2x_1x_2^2 - x_1x_3^2 + 2x_2^2x_3 \\ &= (x_1 - x_3)(x_1^2 + x_1x_3 - 2x_2^2). \end{aligned}$$

Table 10.2 below has timings for Magma, Maple, Singular and Maxima to compute and then factor $\det(T_n)$. The column numterms is the number of terms of the two factors. The polynomial $\det(T_n)$ and its factors are dense. All four computer algebra systems use multivariate Hensel lifting to recover the factors. They first pick a main variable, say x_1, then pick an evaluation point $(\alpha_2, \alpha_3, \ldots, \alpha_n) \in \mathbb{Z}^{n-1}$ then factor the univariate polynomial $\det(Tn)(x_1, \alpha_2, \alpha_3, \ldots, \alpha_n)$ over $\mathbb{Z}$. They then apply Hensel lifting to recover the variables in the multivariate factors one variable at a time. The four computer algebra systems have different Hensel lifting algorithms.

Maxima uses Paul Wang's Hensel lifting from [29]. Since Maple 2019, Maple uses Monagan and Tuncer's sparse Hensel lifting from [22, 23, 24]. Prior to Maple 2019, Maple used Keith Geddes' implementation of Wang's algorithm from the [7] text. Allan Steel implemented the Hensel lifting in Magma. Allan also used the description of Wang's algorithm from [7]. Martin Lee implemented the Hensel lifting for Singular. His implementation is described in his PhD thesis [15].

		Magma 2.28-19		Maple 2024		Singular 3.4.1		Maxima 5.45.0	
n	$\#f_1, \#f_2$	det	factor	det	factor	det	factor	det	factor
8	167,167	0.01	0.09	.008	.089	0.003	0.018	.840	40.44
9	294,153	0.08	0.26	.026	.218	0.019	0.150	7.93	896.7
10	931,931	0.64	1.50	.382	3.83	0.112	2.406	64.18	22,013.1
11	1730,849	5.09	4.55	1.52	9.71	0.695	29.249	373.2	NA
12	5579,5579	32.93	94.89	6.71	21.92	4.526	405.785	NA	NA
13	10611,4983	215.14	365.3	36.41	55.16	36.915	1,689.11	NA	NA
14	34937,34937	1204.37	5,484.3	169.2	388.80	130.86	96,242.9	NA	NA

Table 11: Timings (in CPU seconds) to compute and factor $\det(T_n)$. NA = Not Attempted.

10.3 Circulant matrices

Let C_n be the n by n circulant matrix with $C_{nij} = x_{(i+j-2) \mod n+1}$ for $1 \le i \le n$, $1 \le j \le n$. For $n \ge 1$, $\det(C_n)$ has $\phi(n)$ irreducible factors where ϕ is Euler's totient function. One of the factors is $x_1 + x_2 + \cdots + x_n$. For example

$$C_3 = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_1 \\ x_3 & x_1 & x_2 \end{bmatrix} \qquad \begin{aligned} \det(C_3) &= -x_1^3 + 3x_1x_2x_3 - x_2^3 - x_3^3 \\ &= (x_1 + x_2 + x_3)(-x_1^2 + x_1x_2 + x_1x_3 - x_2^2 + x_2x_3 - x_3^2). \end{aligned}$$

Notice that the second factor of $\det(C_3)$ has 6 terms but $\det(C_3)$ has 4 terms. If n is prime then $\det(C_n)$ has one linear factor $x_1 + x + 2 + \cdots + x_n$ and one large non-linear factor which has more terms than $\det(C_n)$. In Table 10.3 below, column 2 shows the number of terms in $\det(C_n)$ and the number of terms in the largest factor of $\det(C_n)$.

In Hensel lifting, the linear factor will be found first. For an odd prime n, the non-linear factor can be found by a single division. But this division has a quotient with many more terms than the divisor which has n terms which will cause a slowdown if the division algorithm is poor. If division is not used then the Hensel lifting must recover the large factor. Notice what happens to Magma and Singular for $n = 11$ and $n = 13$ in Table 10.3.

		Magma 2.28-19		Maple 2024		Singular 3.4.1		Maxima 5.45.0	
n	#det,#fmax	det	factor	det	factor	det	factor	det	factor
7	246, 924	0.00	0.01	.002	.020	0.001	0.012	.090	0.15
8	810, 86	0.01	0.05	.007	.084	0.003	0.018	0.580	0.361
9	2704, 1005	0.03	0.53	.027	.273	0.011	0.137	3.80	0.635
10	7492, 715	0.15	5.70	.135	2.18	0.056	0.340	30.39	4.037
11	32066,184756	0.95	104.52	.931	0.983	0.310	133.167	2922.1	35.42
12	86500, 621	7.02	2019.27	3.22	4.07	2.359	2.814	OM	113.97
13	400024,2704156	61.72	43,519.1	17.59	11.23	12.673	39,838.9	NA	NA
14	1366500,27132	427.74	$> 6days$	160.8	508.2	54.865	296.051	NA	NA

Table 12: Timings (in CPU seconds) to compute and factor $\det(C_n)$. OM = Out of Memory, NA = Not Attempted.

References

[1] Wibe Bosma, John Cannon and Catherine Playoust. The Magma algebra system. I. The user language, *J. Symb. Cmpt.*, **24**:235–265, Elsevier, 1997. DOI = 10.1006/jsco.1996.0125,

[2] Steven W. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors, *J. ACM* **18**:478–504, ACM, 1971.

[3] Bruce W. Char, Keith O. Geddes, W. Morven Gentleman, Gaston H. Gonnet. The Design of Maple: a Compact, Portable, and Powerful Computer Algebra System. Proceedings of EUROCAL '83, pp. 101–115, Springer, March 1983.

[4] Bruce Char, Gregory J. Fee, Keith O. Geddes, Gaston H. Gonnet, Michael B. Monagan, Stephen M. Watt. On the Design and Performance of the Maple System. Proceedings of the 1984 Macsyma User's Conference, pp. 189–220, 1984.

[5] Bruce Char, Keith Geddes, Gaston Gonnet. GCDHEU: Heuristic Polynomial GCD Algorithm Based on Integer GCD Computation. Proceedings of EUROSAM 84, LNCS **174**:285–296, Springer, 1984.

[6] Decker, W.; Greuel, G.-M.; Pfister, G.; Schönemann, H.: `Singular` 4-4-0 — A computer algebra system for polynomial computations. https://www.singular.uni-kl.de, 2024.

[7] K.O. Geddes, S. Czapor, and G. Labahn. *Algorithms for Computer Algebra.* Kluwer, 1992.

[8] Richard Fateman. Comparing the speed of programs for sparse polynomial multiplication. ACM SIGSAM Bulletin **37**(1):4–15, 2003.

[9] Keith Geddes. Numerical Integration using Symbolic Analysis. Maple Technical Newsletter **6**:8–17, Waterloo Maple Software, Fall 1991.

[10] Hall, A.D. The ALTRAN System for Rational Function Manipulation — A Survey. Communications of the ACM, **14**(8):517–521, August 1971.

[11] Jiaxiong Hu and Michael Monagan. A fast parallel sparse polynomial GCD algorithm. In *Proceedings of ISSAC 2016*, pp. 271–278, ACM, 2016.

[12] Qiao-Long Huang and Michael Monagan. A New Sparse Polynomial GCD by Separating Terms. Proceedings of ISSAC 2024, pp. 134-142, ACM Digital Library, 2024.

[13] Johnson, S.C. Sparse polynomial arithmetic. ACM SIGSAM Bulletin **8**(3):63–71, 1974.

[14] J. de Kleine, M. Monagan, A. Wittkopf. Algorithms for the Non-monic case of the Sparse Modular GCD Algorithm. *Proceedings of ISSAC '2005*, ACM Press, pp. 124–131, 2005.

[15] Martin M. Lee. Factorization of multivariate polynomials. Ph.D. Thesis, 2013. See `https://d-nb.info/1036637972/34`

[16] The Maxima Group. Maxima, a Computer Algebra System. Version 5.45.0, May 24, 2021. http://maxima.sourceforge.net/

[17] M. B. Monagan. In-place arithmetic for polynomials over $\mathbf{Z}_n$. *Proceedings of DISCO '92*, LNCS **721**:22–34, Springer, 1993.

[18] Michael Monagan and Roman Pearce. Sparse Polynomial Division Using a Heap. *J. Symb. Cmpt.* **46**(7):807–822, 2011.

[19] Michael Monagan and Roman Pearce. Parallel Sparse Polynomial Multiplication using Heaps. *Proceedings of ISSAC '09*, pp. 263-269, ACM, 2009.

[20] Michael Monagan and Roman Pearce. POLY: A new polynomial data structure for Maple. In *Computer Mathematics*, Springer Verlag, pp. 325–348, October 2014.

[21] Michael Monagan and Roman Pearce. The design of Maple's sum-of-products and POLY data structures for representing mathematical objects. *Communications of Computer Algebra*, **48** (4), pp. 166–186, December 2014.

[22] Michael Monagan and Baris Tuncer. Using Sparse Interpolation in Hensel Lifting. *Proceedings of CASC 2016*, LNCS **9890**:381–400, Springer, 2016.

[23] Michael Monagan and Baris Tuncer. Polynomial Factorization in Maple 2019. *Proceedings of the 2019 Maple conference*, In *Maple in Mathematics Education and Research*. Communications in Computer and Information Science, **1125**:341–345, Springer, 2020.

[24] Michael Monagan and Baris Tuncer. The complexity of sparse Hensel lifting and sparse polynomial factorization. *J. Symb. Cmpt.* **99**: 189–230, Springer, 2020.

[25] Michael Monagan and Roman Pearce. Sparse Polynomial Division Using a Heap. *J. Symb. Cmpt.* **46** (7) 807–822, 2011.

[26] Roman Pearce and Michael Monagan. Parallel Sparse Polynomial Division using Heaps. Proceedings of PASCO '2010, pp. 105–111, ACM, 2010.

[27] Juho Snellman's Weblog. Numbers and tagged pointers in early Lisp implementations. https://www.snellman.net/blog/archive/2017-09-04-lisp-numbers/ Posted 2017.

[28] David Stoutemyer. Which polynomial representation is best? Surprises Abound! Proceedings of the 1984 Macsyma User's Conference, pp. 221–243, May 1984.
Accessible at https://udspace.udel.edu/items/55e9feab-ba45-46a1-9dc0-db2adbfb7157

[29] Wang, P.S. An improved Multivariate Polynomial Factoring Algorithm. *Mathematics of Computation*, **32**, 1978.

[30] Paul Wang. The EEZ-GCD algorithm. *ACM SIGSAM Bulletin*, **14**(2): 50–60, 1980.

[31] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. of EUROSAM '79*, pp. 216–226. Springer-Verlag, 1979.

11 Appendix A

Example of the GCDHEU algorithm from [5] for computing a bivariate GCD in $\mathbb{Z}[x, y]$.

```
> A := -21*x^9+31*x^8*y-172*x^7*y^2-470*x^6*y^3-12*x^5*y^4+220*x^4*y^5+379*x^3*y^6
-17*x^2*y^7+324*x*y^8+490*y^9-116*x^8-168*x^7*y+907*x^6*y^2-559*x^5*y^3-519*x^4*y^4
-900*x^3*y^5+71*x^2*y^6+259*x*y^7-801*y^8+217*x^7-598*x^6*y+889*x^5*y^2+744*x^4
*y^3+789*x^3*y^4+706*x^2*y^5-128*x*y^6+211*y^7+173*x^6-858*x^5*y+329*x^4*y^2-
732*x^3*y^3-433*x^2*y^4-502*x*y^5-375*y^6+265*x^5-1069*x^4*y-187*x^3*y^2+396*x^2
*y^3-398*x*y^4+387*y^5+544*x^4+700*x^3*y+475*x^2*y^2+509*x*y^3-89*y^4-206*x^3-
853*x^2*y-104*x*y^2+678*y^3+731*x^2-130*x*y-610*y^2-350*x-322*y+343:

> B := -141*x^8-115*x^7*y+473*x^6*y^2+548*x^5*y^3+158*x^4*y^4+95*x^3*y^5-111*x^2*y^6
-579*x*y^7-240*y^8+86*x^7-481*x^6*y-450*x^5*y^2-469*x^4*y^3-83*x^3*y^4-204*x^2*y
^5+1003*x*y^6+601*y^7+537*x^6-295*x^5*y+907*x^4*y^2+223*x^3*y^3-156*x^2*y^4-863
*x*y^5-511*y^6+355*x^5-148*x^4*y-702*x^3*y^2-61*x^2*y^3+860*x*y^4+221*y^5-617*x
^4-12*x^3*y+209*x^2*y^2+198*x*y^3+30*y^4+104*x^3+759*x^2*y-1129*x*y^2-141*y^3
-768*x^2-49*x*y-340*y^2+829*x+929*y-497:

> a := subs(x=1000,y=10^9,A); # a := A(1000,10^9)
a := 490000323198983259590071090725692979705008174087316669871500044262412 41\
    3472730650343

> b := subs(x=1000,y=10^9,B); # b := B(1000,10^9)
b := -240000578399109997416204704861607608274510011324228190813886664967767171497

> g := igcd(a,b);
                                g := 5000002993

> g := genpoly(g,10^9,y);
                                g := 2993 + 5 y

> G := genpoly(g,10^3,x);
                               G := 3 x + 5 y - 7
```

12 Appendix B

Maple code for generating the polynomials f_1, f_2, f_3 for the benchmark in Section 9.5.

```
kernelopts(numcpus=1);
d := 10;
for n from 5 to 15 do
  X := [seq( x||i, i=1..n )];
  f := randpoly(X,coeffs=rand(-10^6..10^6),terms=100,degree=d);
  g := randpoly(X,coeffs=rand(-10^6..10^6),terms=100,degree=d);
  h := randpoly(X,coeffs=rand(-10^6..10^6),terms=100,degree=d);
  a := CodeTools[Usage](expand(f*g*h));
  h := CodeTools[Usage](factor(a)); # Uses MTSHL in Maple 2019
od:
```