ON SOME COMPUTATIONAL PROBLEMS
IN FINITE ABELIAN GROUPS

JOHANNES BUCHMANN, MICHAEL J. JACOBSON, JR., AND EDLYN TESKE

ABSTRACT. We present new algorithms for computing orders of elements, discrete logarithms, and structures of finite abelian groups. We estimate the computational complexity and storage requirements, and we explicitly determine the O-constants and Ω-constants. We implemented the algorithms for class groups of imaginary quadratic orders and present a selection of our experimental results.

Our algorithms are based on a modification of Shanks' baby-step giant-step strategy, and have the advantage that their computational complexity and storage requirements are relative to the actual order, discrete logarithm, or size of the group, rather than relative to an upper bound on the group order.

1. INTRODUCTION

Let G be a finite abelian group, written multiplicatively, in which we assume that the following are possible:

- for $a, b \in G$ we can compute $c = a * b$
- for $a \in G$ we can compute a^{-1}
- for $a, b \in G$ we can test whether $a = b$

We call these the *group operations*. Note that from every group element a we can determine the neutral element $1 = a * a^{-1}$. As an example, we will consider class groups of imaginary quadratic fields. Another example is the group of points on an elliptic curve over a finite field.

For any subset S of G, denote by $\langle S \rangle$ the subgroup of G generated by S. If $\langle S \rangle = G$, then S is called a generating set of G. If $S = \{g\}$, then write $\langle g \rangle$ instead of $\langle S \rangle$.

Three common computational problems in such groups are:

- Given $g \in G$ compute $|\langle g \rangle|$, the *order* of g in G, i.e., the least positive integer x such that $g^x = 1$.
- Given $g, d \in G$ decide whether d belongs to the cyclic subgroup $\langle g \rangle$ of G generated by g. If $d \in \langle g \rangle$, find $\log_g d$, the *discrete logarithm* of d to the base g, i.e., the least non-negative integer x such that $g^x = d$.
- Given a generating set of G compute the *structure* of G. By computing the structure of G we mean computing positive integers $m_1, \ldots, m_k$ with $m_1 > 1$,

$m_i | m_{i+1}$, $1 \leq i < k$, and an isomorphism $\phi : G \to \mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_k\mathbb{Z}$. This isomorphism is given in terms of the images of the generators. The integers m_i are the uniquely determined *invariants* of G.

In this paper we present improved versions of Shanks' algorithms for solving these problems. As in Shanks' original method [10] (see also [4], [8]), operations in G and table look-ups are used in our algorithms. The table entries are pairs (S^z, z), where S is a set of group elements, z belongs to the set $\mathbb{Z}^S$ of all maps $S \longrightarrow \mathbb{Z}$, and $S^z = \prod_{g \in S} g^{z(g)}$. When we estimate the complexity of our algorithms, we count the number of group operations, the number of table look-ups, and we determine bounds on the table sizes, i.e., the number of group elements which have to be stored. We ignore the time and space for doing index calculations. If hashing on the group elements is possible, the tables of group elements are hash tables and the time for one table look-up is comparable to the time required for a group operation.

Here are our main results.

Theorem 1.1. *There is an algorithm for computing the order x of an element $g \in G$ which executes one inversion and at most*

$$4 \left\lceil \sqrt{x} \right\rceil + \left\lceil \log \frac{\sqrt{x}}{2} \right\rceil$$

multiplications in G. It uses a table of at most

$$2 \left\lceil \sqrt{x} \right\rceil$$

pairs $(g, r) \in G \times \{1, \dots, 2 \lceil \sqrt{x} \rceil\}$. The total number of table look-ups is bounded by

$$2 \left\lceil \sqrt{x} \right\rceil .$$

Theorem 1.2. *There is an algorithm that decides for $g, d \in G$, $d \neq 1$, whether $d \in \langle g \rangle$ and if so computes $\log_g d$. Let*

$$x = \begin{cases} |\langle g \rangle| & \text{if } d \notin \langle g \rangle, \\ \log_g d & \text{if } d \in \langle g \rangle. \end{cases}$$

The algorithm executes at most

$$6 \left\lceil \sqrt{x} \right\rceil + \left\lceil \log \sqrt{x} \right\rceil$$

multiplications in G. It uses a table of at most

$$2 \left\lceil \sqrt{x} \right\rceil$$

pairs $(g, r) \in G \times \{1, \dots, 2 \lceil \sqrt{x} \rceil\}$. The total number of table look-ups is bounded by

$$4 \left\lceil \sqrt{x} \right\rceil .$$

Our algorithm for computing the structure of G may behave differently if the generators are input in different orders. Therefore, the algorithm receives as input a *generating sequence*, i.e., a finite sequence $S = (g_1, \dots, g_l)$ of group elements such that $\{g_1, \dots, g_l\}$ is a generating set of G. Set

(1) $$G_j = \langle g_1, \dots, g_j \rangle, \quad 0 \leq j \leq l,$$

and

(2) $$l(S) = |\{j \in \{1, \dots, l\} \; : \; G_{j-1} \neq G_j\}| .$$

In other words, if we generate G by using $g_1, g_2, g_3, \ldots,$ $l(S)$ is the number of generators which enlarge the group. Note that $l(S)$ is at least as large as the number of invariants of G. We will prove the following result.

Theorem 1.3. *There is an algorithm that computes the structure of G from a generating sequence S of G which executes $|S|$ inversions and at most*

$$2^{\frac{l(S)}{2}}(|S|+5)\sqrt{|G|} + 4\,l(S)\sqrt{|G|} + \log|G|$$

multiplications in G, with $l(S)$ from (2). It uses two tables of at most

$$2^{\frac{l(S)}{2}}\sqrt{|G|} \quad and \quad 2^{\frac{l(S)}{2}+1}\sqrt{|G|}$$

pairs $(g,\vec{q}) \in G \times \{0, \ldots, 2\sqrt{|G|}\}^{|S|}$. The total number of table look-ups is bounded by

$$2^{\frac{l(S)}{2}}(|S|+l(S)+2)\sqrt{|G|}.$$

The upper bound for the complexity of the group structure algorithm is exponential in the number $l(S)$ of generators that are really used to determine the group structure. If that number is fixed, the complexity of the algorithm is $O(|S|\sqrt{|G|})$. On the other hand, our analysis shows that if

$$G = (\mathbb{Z}/2\mathbb{Z})^l$$

for some positive integer l, the complexity of our algorithm is $\Omega(|G|)$, where the symbol $\Omega(f(n))$ stands for the set of all functions g such that there exists a constant M with $|g(n)| \geq M|f(n)|$ for all large n. This lower bound also holds for Shanks' original algorithm and its variations. Hence, for finite abelian groups with a large number of small invariants our algorithm is not appropriate.

The basic idea of this paper is to use baby-step giant-step algorithms with some initial step-width $v \in 2\mathbb{N}$ and to double that step-width as long as the result of the computation has not been found. A similar idea has been used in [2] but the results obtained there are weaker than ours. For $v = 2$ we obtain the above theorems. If v is chosen such that $L = v^2$ is an upper bound on the group order, we obtain Shanks' original algorithms. Our analysis shows that for $v = \sqrt{L}$ the number of multiplications and the table size in the order and discrete logarithm algorithms is $\Omega(\sqrt{L})$. Also, for that choice of v the number of multiplications and the table size in the group structure algorithm is $\Omega(2^{l(S)/2}|S|\sqrt{|L|})$. Thus, if the upper bound L is much larger than the actual order, discrete logarithm or group order, the algorithm wastes a lot of time and space.

We implemented our algorithms for class groups of imaginary quadratic orders using the computer algebra system LiDIA [7]. We present experimental results which yield good choices for the initial step-width v for these groups.

The paper is organized as follows. In Section 2 we describe and analyze the order algorithm. That section also contains the basic idea of the paper. In Section 3 we discuss the algorithm for computing discrete logarithms, and Section 4 is devoted to the group structure algorithm.

2. Computing the order of an element

Given an element $g \in G$ we wish to compute $x = |\langle g \rangle|$. Our improved algorithm, a modification of Shanks' baby-step giant-step method, is based on the following statement:

Lemma 2.1. *Let v be an even positive integer. For every positive integer x there are uniquely determined integers k, q and r with $k \geq 0$, $\lfloor 4^{k-1} \rfloor v^2 \leq 2^k v q < 4^k v^2$ and $1 \leq r \leq 2^k v$ such that $x = y + r$, $y = 2^k v q$.*

Proof. Let $x \in \mathbb{N}$. We first show the existence of such k, q and r. We choose k such that $\lfloor 4^{k-1} \rfloor v^2 < x \leq 4^k v^2$, and write $x = 2^k v q + r$ with $1 \leq r \leq 2^k v$. Then $\lfloor 4^{k-1} \rfloor v^2 - 2^k v < 2^k v q \leq 4^k v^2 - 1$, which implies that $2^k v q < 4^k v^2$. Moreover, if $k = 0$, we have $-v < vq$, so $0 \leq vq$. If $k \geq 1$, we have $4^{k-1} v^2 - 2^k v < 2^k v q$; hence $2^{k-1}(v/2) - 1 < q$. Since $v/2$ is integral, this implies that $2^{k-1}(v/2) \leq q$, so that $4^{k-1} v^2 \leq 2^k v q$.

To show the uniqueness of this representation, let $x = 2^k v q + r$ with k, q and r as stated above. Then $\lfloor 4^{k-1} \rfloor v^2 \leq 2^k v q < 4^k v^2$, which implies $q \leq 2^k v - 1$, so that $x = 2^k v q + r \leq 2^k v (2^k v - 1) + 2^k v = 4^k v^2$. Moreover, we have $\lfloor 4^{k-1} \rfloor v^2 < 2^k v q + r = x$. These inequalities determine k uniquely. The uniqueness of q and r is due to the uniqueness statement for division with remainder. □

We explain the method for computing $x = |\langle g \rangle|$. We select an even positive integer v which is used as the initial step-width in the algorithm. Then there is a unique non-negative integer k such that x belongs to the interval

$$I_k = \{ \lfloor 4^{k-1} \rfloor v^2 + 1, \ldots, 4^k v^2 \}.$$

We search those intervals for $k = 0, 1, 2, \ldots$ until x is found. By Lemma 2.1, each number in I_k can be written as $y + r$ with $y = 2^k v q$ and r, q as stated in Lemma 2.1. Also, each integer that can be written in this way belongs to the interval I_k. To check whether x is in I_k we test whether $g^{y+r} = 1$ with $y = 2^k v q$ and r and q as stated in Lemma 2.1. This means that we test whether

$$g^y = g^{-r}, \qquad 1 \leq r \leq 2^k v, \quad y = 2^k v q, \quad \lfloor 4^{k-1} \rfloor v^2 \leq y < 4^k v^2.$$

For this purpose, we compute the set

$$R_k = \{ (g^{-r}, r) : 1 \leq r \leq 2^k v \},$$

and for all values of q such that $\lfloor 4^{k-1} \rfloor v^2 \leq y < 4^k v^2$ we check whether there exists $(g^y, r) \in R_k$ for some r. If so, $|\langle g \rangle| = y + r$. Otherwise, we increase k by 1. If $x \leq v$, the set R_0 contains at least one pair $(1, r)$, and $|\langle g \rangle| = r$ for the smallest such r. Therefore, before adding a pair (g^{-r}, r), $1 \leq r \leq v$, to R_0 in the course of the computation of R_0, we always check whether $g^{-r} = 1$, and we break if the answer is "yes", since then $|\langle g \rangle|$ is already found.

The efficiency of the algorithm can be improved if we know a lower bound B of $|\langle g \rangle|$. Writing $C = B - 1$, we then work with the set $R_k = \{ (g^{-(r+C)}, r) : 1 \leq r \leq 2^k v \}$, and if we find $(g^y, r) \in R_k$, $|\langle g \rangle| = y + r + C$. If no lower bound for $|\langle g \rangle|$ is known, we set $C = 0$.

We now present the algorithm.

Algorithm 2.2.

This algorithm computes the order of the group element g in G.

INPUT: $g \in G$, lower bound $C + 1$ for $|\langle g \rangle|$, initial step-width v $(v \in 2\mathbb{N})$
OUTPUT: $x = |\langle g \rangle|$

(1)	$x = 0$
(2)	$s = 1$; $y = v$; $u = v$
(3)	$h = g^{-1}$
(4)	$a = h^C$; $b = g^v$; $c = b$ /* $a = g^{-C-r}$;
(5)	$R = \emptyset$ $b = g^y$; $c = g^u$ */
(6)	**while** $(x == 0)$ **do**
(7)	**for** $(r = s, s + 1 \dots, u)$ **do** /* new baby steps */
(8)	$a = a * h$
(9)	**if** $(s == 1)$ **then** /* check if $1 \le x \le v$ */
(10)	**if** $(a == 1)$ **then**
(11)	$x = r + C$
(12)	**return** (x)
(13)	**break** while
(14)	**else**
(15)	$R = R \cup \{(a, r)\}$
(16)	**fi**
(17)	**else**
(18)	$R = R \cup \{(a, r)\}$
(19)	**fi**
(20)	**od**
(21)	**while** $(x == 0 \text{ and } y < u^2)$ **do** /* giant steps */
(22)	**if** (there is a number r such that $(b, r) \in R$) **then**
(23)	$x = y + r + C$
(24)	**return** (x)
(25)	**else**
(26)	$y = y + u$
(27)	$b = b * c$
(28)	**fi**
(29)	**od**
(30)	$s = u + 1$; $u = 2u$ /* double
(31)	$c = c^2$ step-width */
(32)	**od**

Theorem 2.3. *Let $C = 0$. Let $x = |\langle g \rangle|$. For every choice of v, Algorithm 2.2 executes one inversion and at most $2\lfloor \log v \rfloor + 1$ multiplications in G and requires space for three group elements. On further group multiplications, space required, and table look-ups, we have the following estimates.*

1. *If $x \leq v$, Algorithm 2.2 executes x additional multiplications in G. It uses a table of $x - 1$ pairs $(g, r) \in G \times \{1, \ldots, x - 1\}$ and it performs x equality checks.*

2. *If $\sqrt{x} \leq v < x$, the number M of additional multiplications in G satisfies*

$$v \leq M \leq \lceil \sqrt{x} \rceil + v - 2.$$

The algorithm uses a table of v pairs $(g, r) \in G \times \{1, \ldots, v\}$, and it performs v equality checks. The total number TL of table look-ups satisfies

$$1 \leq TL \leq \lceil \sqrt{x} \rceil - 1.$$

3. *If $\sqrt{x} > v$, the number M of additional multiplications in G satisfies*

$$\frac{5}{4} \lceil \sqrt{x} \rceil + \left\lceil \log \frac{\sqrt{x}}{v} \right\rceil - 1 \leq M \leq 4 \lceil \sqrt{x} \rceil - \frac{v}{2} + \left\lceil \log \frac{\sqrt{x}}{v} \right\rceil - 5.$$

It performs v equality checks. It uses a table of at least $\lceil \sqrt{x} \rceil$ and at most $2 \lceil \sqrt{x} \rceil - 2$ pairs $(g, r) \in G \times \{1, \ldots, 2 \lceil \sqrt{x} \rceil\}$. The total number TL of table look-ups satisfies

$$\frac{\lceil \sqrt{x} \rceil}{4} \leq TL \leq 2 \lceil \sqrt{x} \rceil - \frac{v}{2} - 2.$$

Proof. For the initialization, Algorithm 2.2 requires one inversion and at most $2 \lfloor \log v \rfloor + 1$ multiplications to compute g^{-1} and $b = c = g^v$. It must store g^{-1}, b and c.

If $x \leq v$, we find $x = |\langle g \rangle|$ during the first iteration of the outer while loop, in the course of the computation of the set R ($= R_0$). This requires x group multiplications and x equality checks, and the set R contains $x - 1$ pairs (g^{-r}, r).

If $\sqrt{x} \leq v < x$, we also find $x = |\langle g \rangle|$ during the first iteration of the outer while loop. The set R contains v pairs (g^{-r}, r), which also means that the algorithm must perform v multiplications in G to compute R. It performs v equality checks to test whether $g^{-r} = 1$. In the iterations of the inner while loop, the algorithm checks whether $(g^{qv}, r) \in R$ while $v \leq qv \leq x - r$ ($r \in \{1, \ldots, v\}$), i.e., $1 \leq q \leq (x - 1)/v$, so we have $1 \leq q < \sqrt{x}$. It computes g^{qv} while $2v \leq qv \leq x - r$, thus $2 \leq q < \sqrt{x}$. This requires between 1 and $\lceil \sqrt{x} \rceil - 1$ table look-ups, and at most $\lceil \sqrt{x} \rceil - 2$ group multiplications. Hence, the total number of multiplications in the outer while loop is between v and $v + \lceil \sqrt{x} \rceil - 2$, if $\sqrt{x} \leq v < x$.

If $v < \sqrt{x}$, the algorithm performs k additional iterations of the outer while loop, where

$$(3) \qquad 4^{k-1} v^2 < x \leq 4^k v^2$$

i.e., $k - 1 < \log \frac{\sqrt{x}}{v} \leq k$, hence $k = \left\lceil \log \frac{\sqrt{x}}{v} \right\rceil$. After the last iteration, the set R contains $2^k v$ pairs (g^{-r}, r). Thus, to compute R the algorithm performs $2^k v$ multiplications in G, and it must store a table of $2^k v$ pairs. From (3) we see that $2^{k-1} v < \sqrt{x} \leq 2^k v$, so

$$(4) \qquad \lceil \sqrt{x} \rceil \leq 2^k v \leq 2 \lceil \sqrt{x} \rceil - 2.$$

In the first inner while loop, the algorithm computes g^{qv} for $2 \leq q \leq v$, which requires $v - 1$ multiplications. In all of the following inner while loops except the

last, it computes $g^{q2^i v}$ while

$$4^{i-1}v^2 < q2^i v \le 4^i v^2,$$

i.e.,

$$2^{i-1}\frac{v}{2} < q \le 2^i v,$$

where $1 \le i \le k-1$. This requires

$$2^i v - 2^{i-1}\frac{v}{2} = 3 \cdot 2^{i-1}\frac{v}{2}$$

multiplications in each loop, which can be summed to

$$\sum_{i=1}^{k-1} 3 \cdot 2^{i-1}\frac{v}{2} = 3 \cdot (2^{k-1}-1)\frac{v}{2}$$

multiplications. In the last loop, the algorithm computes $g^{q2^k v}$ while

$$4^{k-1}v^2 < q2^k v \le x - r, \quad r \in \{1,\dots,2^k v\},$$

so $2^{k-1}\frac{v}{2} < q \le \frac{x-1}{2^k v}$. Since $\sqrt{x} \le 2^k v$, we have $\frac{x-1}{2^k v} \le \frac{x-1}{\sqrt{x}} < \sqrt{x}$, and this requires at most

$$\lceil \sqrt{x} \rceil - 1 - 2^{k-1}\frac{v}{2}$$

multiplications. Thus, the total number of multiplications in the inner while loops is at least

$$v - 1 + 3 \cdot (2^{k-1}-1)\frac{v}{2} \quad = \quad 3 \cdot 2^{k-1}\frac{v}{2} - \frac{v}{2} - 1$$

$$\overset{(4)}{\ge} \quad \frac{3}{4}\lceil \sqrt{x}\rceil - \frac{\sqrt{x}}{2} - 1 \ge \frac{\lceil \sqrt{x}\rceil}{4} - 1,$$

and at most

$$v - 1 + 3 \cdot (2^{k-1}-1)\frac{v}{2} + \lceil \sqrt{x}\rceil - 1 - 2^{k-1}\frac{v}{2} \quad = \quad 2^{k-1}v - \frac{v}{2} + \lceil \sqrt{x}\rceil - 2$$

$$\overset{(4)}{\le} \quad 2\lceil \sqrt{x}\rceil - \frac{v}{2} - 3,$$

The total number of table look-ups is bounded below by $\lceil \sqrt{x}\rceil/4$ and above by $2\lceil \sqrt{x}\rceil - \frac{v}{2} - 2$, which is the maximum number of iterations of the inner while loops. Finally, note that in k outer while loops the algorithm performs one multiplication to compute c^2. Together with (4) we get that Algorithm 2.2 performs at least

$$\left\lceil \log \frac{\sqrt{x}}{v}\right\rceil + \frac{5}{4}\lceil \sqrt{x}\rceil - 1$$

and at most

$$\left\lceil \log \frac{\sqrt{x}}{v}\right\rceil + 4\lceil \sqrt{x}\rceil - \frac{v}{2} - 5$$

additional multiplications in G to find $|\langle g\rangle|$, if $v < \sqrt{x}$.   □

Remark 2.4. To adapt Theorem 2.3 to the case $C \ge 1$, we just have to replace each x by $x - C$ and add to the total number of group multiplications the multiplications required to compute $a = (g^{-1})^C$, i.e., at most $2\lfloor \log C\rfloor + 1$ multiplications.

In practice, the most efficient way to handle the set R is by means of a hash table. This is possible as long as the group elements are represented as sequences of integers. Then, each look-up in the table R requires just one computation of a hash value and usually one equality test for group elements.

As we see from Theorem 2.3, the efficiency of Algorithm 2.2 depends largely on the appropriate choice of the initial step-width v. As noted by Shanks [10], the optimal choice of v is $v = \sqrt{|\langle g \rangle|}$. This results in about $2\sqrt{|\langle g \rangle|}$ group multiplications in our algorithm. If v is chosen too large (in comparison with $\sqrt{|\langle g \rangle|}$), we waste space and time because the set R is too big. If v is chosen too small, we waste time because of superfluous iterations of the outer while loop.

In order to test our algorithm, we implemented it using the LiDIA system [7] to compute orders of elements in ideal class groups of imaginary quadratic orders. For three discriminants of sizes ten, fifteen, and twenty decimal digits, we computed the orders of the ideal classes of four prime ideals that we knew from previous computations had different orders. In Tables 1, 2, and 3 we show the actual numbers of group multiplications and table look-ups, denoted by GM and TL respectively, that were required to compute the order of each prime ideal class, together with the lower and upper bounds predicted by Theorem 2.3. Δ denotes the discriminant of the quadratic order and I_p denotes the ideal class of which the prime ideal lying over the prime p is the reduced representative. We compute the order of each prime ideal class three times, using a different value of v each time. The simplest version of our algorithm uses $v = 2$, $v = \Delta^{1/4}$ is equivalent to Shanks' original algorithm [10], and $v = \Delta^{1/4}/2$ is half-way between the other two and has been shown to yield the best overall run times in our tests. Table 4 gives the run times for these computations on a SPARCstation 20.

TABLE 1. Order algorithm — group multiplications and table look-ups ($v = 2$)

			Lower		Computed		Upper	
Δ	p	$|\langle I_p \rangle|$	GM	TL	GM	TL	GM	TL
$-4(10^{10} + 1)$	5	4033	85	16	164	94	258	125
	3	16132	166	32	324	189	515	253
	13	24198	202	39	485	221	628	309
	7	48396	282	55	580	316	884	437
$-4(10^{15} + 1)$	7	2	0	0	4	0	5	0
	29	42908	267	52	558	294	836	413
	17	128724	456	89	1027	506	1441	715
	3	257448	643	127	1278	757	2037	1013
$-4(10^{20} + 1)$	13	232024638	19054	3808	38750	22352	60942	30463
	5	464049276	26941	5385	63327	30544	86179	43081
	37	928098552	38095	7616	77489	44706	121871	60927
	7	1856197104	53870	10771	126642	61090	172348	86165

For $\Delta = -4(10^{10} + 1)$, our algorithm using $v = 2$ is faster than Shanks' algorithm (i.e. $v = \Delta^{1/4}$) for I_5, the class with the smallest order. It is slower for I_3 and I_{13}, even though it executes fewer total group operations, because it performs four

TABLE 2. Order algorithm — group multiplications and table look-ups $(v = \Delta^{1/4}/2)$

Δ	p	$|\langle I_p \rangle|$	Lower		Computed		Upper	
			GM	TL	GM	TL	GM	TL
$-4(10^{10}+1)$	5	4033	224	1	251	18	297	63
	3	16132	224	1	305	72	361	127
	13	24198	224	1	341	108	389	155
	7	48396	224	1	449	216	453	219
$-4(10^{15}+1)$	7	2	0	0	19	0	19	0
	29	42908	3976	1	4002	10	4199	207
	17	128724	3976	1	4024	32	4350	358
	3	257448	3976	1	4056	64	4499	507
$-4(10^{20}+1)$	13	232024638	70711	1	74014	3281	85965	15232
	5	464049276	70711	1	77295	6562	92274	21541
	37	928098552	70711	1	83858	13125	101197	30464
	7	1856197104	70711	1	96983	26250	113816	43083

TABLE 3. Order algorithm — group multiplications and table look-ups $(v = \Delta^{1/4})$

Δ	p	$|\langle I_p \rangle|$	Lower		Computed		Upper	
			GM	TL	GM	TL	GM	TL
$-4(10^{10}+1)$	5	4033	448	1	467	9	523	63
	3	16132	448	1	494	36	587	127
	13	24198	448	1	512	54	615	155
	7	48396	448	1	566	108	679	219
$-4(10^{15}+1)$	7	2	0	0	21	0	21	0
	29	42908	7953	1	7977	5	8178	207
	17	128724	7953	1	7988	16	8329	358
	3	257448	7953	1	8004	32	8478	507
$-4(10^{20}+1)$	13	232024638	141422	1	143086	1640	156678	15232
	5	464049276	141422	1	144727	3281	162987	21541
	37	928098552	141422	1	148008	6562	171910	30464
	7	1856197104	141422	1	154571	13125	184529	43083

times as many giant steps than Shanks' algorithm, which requires additional table look-ups. In the case of I_7, the optimal value of v is $v = \sqrt{48396} \approx 220$, and the initial step-width $v = \Delta^{1/4} \approx 447$ that is used in Shanks' algorithm is sufficiently accurate that the extra giant steps taken by our algorithm cause it to execute more group operations in total. $v = \Delta^{1/4}/2 \approx 223$ is closer to the optimal value than $v = \Delta^{1/4}$, and this choice of v does result in the best overall performance for three of the four ideals.

For $\Delta = -4(10^{15}+1)$, our algorithm using $v = 2$ is the fastest for three of the four prime ideals. The orders of the ideals are sufficiently small that selecting $v = \Delta^{1/4}$ or $\Delta^{1/4}/2$ results in too many extra baby steps. For $\Delta = -4(10^{20}+1)$,

TABLE 4. Order Algorithm — run times

Δ	p	$|\langle I_p \rangle|$	$v = 2$	$v = \Delta^{1/4}/2$	$v = \Delta^{1/4}$
$-4(10^{10} + 1)$	5	4033	0.19 sec	0.16 sec	0.27 sec
	3	16132	0.31 sec	0.22 sec	0.27 sec
	13	24198	0.44 sec	0.30 sec	0.37 sec
	7	48396	0.60 sec	0.44 sec	0.41 sec
$-4(10^{15} + 1)$	7	2	0.13 sec	0.09 sec	0.10 sec
	29	42908	0.86 sec	2.83 sec	4.87 sec
	17	128724	1.20 sec	2.39 sec	4.60 sec
	3	257448	1.74 sec	1.95 sec	3.26 sec
$-4(10^{20} + 1)$	13	232024638	61.40 sec	51.80 sec	92.22 sec
	5	464049276	81.68 sec	52.72 sec	78.71 sec
	37	928098552	128.06 sec	78.76 sec	116.60 sec
	7	1856197104	172.86 sec	102.08 sec	111.14 sec

our algorithm using $v = 2$ is faster than Shanks' algorithm for one of the four ideals. In this case, $\Delta^{1/4}$ is closer to the optimal value of v than for $\Delta = -4(10^{15} + 1)$. Using $v = \Delta^{1/4}/2$, as in the case of $\Delta = -4(10^{10} + 1)$, actually results in the best overall performance.

Our results suggest that Algorithm 2.2 has two main advantages over Shanks' original algorithm. The first and most obvious advantage is that it is faster when the order of the element is much smaller than the order of the group. In these cases, Shanks' algorithm executes too many baby steps, and although our algorithm using $v = 2$ will execute some unnecessary giant steps, it will still execute fewer group operations overall. The second advantage is when the upper bound on the order of the group is too large. Shanks' algorithm will execute too many baby steps in this case as well. Our algorithm allows one to select an initial step-width that is much smaller than the estimated order of the group, in the hope of attaining a better approximation of $\sqrt{|\langle g \rangle|}$. Using Shanks' algorithm, an initial step-width that is too small results in far too many giant steps, but our algorithm will detect if the initial step-width is too small and enlarge it if necessary.

3. COMPUTING DISCRETE LOGARITHMS

Given $g, d \in G$, we wish to decide whether d belongs to the group $\langle g \rangle$ generated by g. If the answer is "yes," we want to compute $x = \log_g d$. We use the following modification of Algorithm 2.2 to solve this problem. We compute the order of g in G as in Algorithm 2.2, i.e., we try to find integers y and r such that $g^{y+r} = 1$. However, for each y, before checking whether $g^{y+r} = 1$, we check whether $g^{y+r} = d$. For this, we work with the same set R as in Algorithm 2.2. We first check whether $(d^{-1} * g^y, r) \in R$, with R as in Algorithm 2.2. If this is the case, $\log_g d = y + r$. Otherwise, we check whether $(g^y, r) \in R$. As soon as we have found $|\langle g \rangle|$, we know that there is no discrete logarithm of d for base g, since $\log_g d < |\langle g \rangle|$. Just as in Algorithm 2.2, during the computation of $R_0 = \{(g^{-r}, r) : 1 \leq r \leq v\}$ we always check whether $\log_g d$ or $|\langle g \rangle|$ is already found before we include a pair (g^{-r}, r).

Algorithm 3.1.

This algorithm computes the discrete logarithm of d $(d \neq 1)$ for base g.

INPUT: $d, g \in G$ $(d \neq 1)$, initial step-width v $(v \in 2\mathbb{N})$
OUTPUT: $t = 1$ and $x = \log_g d$ if $d \in \langle g \rangle$,
$\quad\quad\quad t = 0$ and $x = |\langle g \rangle|$ if $d \notin \langle g \rangle$

(1) $t = 2; x = 0$
(2) $s = 1; y = v; u = v$
(3) $h = g^{-1}$
(4) $a = 1; b = g^v; c = b$ /* $a = g^{-r}$, $b = g^y$,
(5) $R = \emptyset$ $c = g^u$ */
(6) $f = d^{-1}$
(7)   **while** $(t == 2)$ **do**
(8)   **for** $(r = s, s + 1 \ldots, u)$ **do**   /* new baby steps */
(9) $a = a * h$
(10)   **if** $(s == 1)$ **then**   /* check if $1 \leq x \leq v$ */
(11)   **if** $(a == f)$ **then**
(12) $x = r; t = 1$
(13)   **break** while
(14)   **else**
(15)   **if** $(a == 1)$ **then**
(16) $x = r; t = 0$
(17)   **break** while
(18)   **else**
(19) $R = R \cup \{(a, r)\}$
(20)   **fi**
(21)   **fi**
(22)   **else**
(23) $R = R \cup \{(a, r)\}$
(24)   **fi**
(25)   **od**

```
(26)        while (t == 2 and y < u²)  do      /* giant steps */
(27)            if (there is a number r such    /* checking for
(28)               that (f * b, r) ∈ R)  then      discrete log. */
(29)               x = y + r; t = 1
(30)            else
(31)               if (there is a number r such  /* checking for
(32)                  that (b, r) ∈ R) then        the order of g */
(33)                  x = y + r; t = 0
(34)               else
(35)                  y = y + u
(36)                  b = b * c
(37)               fi
(38)            fi
(39)         od
(40)         s = u + 1; u = 2u                  /* double
(41)         c = c²                                step-width */
(42)      od
(43)      return (t, x)
```

To discuss the complexity of Algorithm 3.1 we use the same arguments as in Theorem 2.3. We only have to observe that there are twice as many equality checks during the computation of R_0. Moreover, in each inner while loop, we have one additional multiplication to compute $f * b$ and at most one additional table look-up to check whether $(f * b, r) \in R$. With $x = \log_g d$ if $d \in \langle g \rangle$ and $x = |\langle g \rangle|$ if $d \notin \langle g \rangle$, this causes at least 1 additional multiplication and at most $\lceil \sqrt{x} \rceil - 1$ additional multiplications and table look-ups if $\sqrt{x} \le v < x$. If $\sqrt{x} > v$, this causes at least $\lceil \sqrt{x} \rceil / 4 - 1$ and at most $2 \lceil \sqrt{x} \rceil - \frac{v}{2} - 2$ additional multiplications and table look-ups. Therefore, we have the following result:

Theorem 3.2. *Let $d \neq 1$. Let*

$$x = \begin{cases} |\langle g \rangle| & \text{if } d \notin \langle g \rangle, \\ \log_g d & \text{if } d \in \langle g \rangle. \end{cases}$$

For every choice of v, Algorithm 3.1 executes two inversions and at most $2\lfloor \log v \rfloor + 1$ multiplications in G and requires space for five group elements. On further group multiplications, space required, and table look-ups, we have the following estimates.

1. *If $x \le v$, Algorithm 3.1 executes x further multiplications in G. It uses a table of $x - 1$ pairs $(g, r) \in G \times \{1, \dots, x - 1\}$, and it performs at most $2x$ equality checks.*
2. *If $\sqrt{x} \le v < x$, the number M of further multiplications in G satisfies*

$$v + 1 \le M \le 2 \lceil \sqrt{x} \rceil + v - 3.$$

The algorithm uses a table of v pairs $(g, r) \in G \times \{1, \dots, v\}$, and it performs 2v equality checks. The total number TL of table look-ups satisfies

$$1 \leq TL \leq 2 \lceil \sqrt{x} \rceil - 2.$$

3. *If $\sqrt{x} > v$, the number M of further multiplications in G satisfies*

$$\frac{3}{2} \lceil \sqrt{x} \rceil + \left\lceil \log \frac{\sqrt{x}}{v} \right\rceil - 2 \leq M \leq 6 \lceil \sqrt{x} \rceil - v + \left\lceil \log \frac{\sqrt{x}}{v} \right\rceil - 7.$$

It performs at most 2v equality checks. It uses a table of at least $\lceil \sqrt{x} \rceil$ and at most $2 \lceil \sqrt{x} \rceil - 2$ pairs $(g, r) \in G \times \{1, \dots, 2 \lceil \sqrt{x} \rceil\}$. The total number TL of table look-ups satisfies

$$\frac{\lceil \sqrt{x} \rceil}{2} - 1 \leq TL \leq 4 \lceil \sqrt{x} \rceil - v - 4.$$

Algorithm 3.1 was implemented using the LiDIA system [7] to compute discrete logarithms in ideal class groups of imaginary quadratic orders. For the same three discriminants used in the previous section, we select the prime ideal class I_p with the largest order and use our algorithm to compute $\log_{I_p} I_p^x$ in the class group, for 5 different values of x with varying sizes. Again, we perform the discrete logarithm computation three times using the same three values of v as before. Table 5 shows the number of group multiplications and table look-ups, denoted by GM and TL, required for each computation and Table 6 gives the run times for each computation.

TABLE 5. DL algorithm — group multiplications and table look-ups

Δ	p	x	$v = 2$		$v = \Delta^{1/4}/2$		$v = \Delta^{1/4}$	
			GM	TL	GM	TL	GM	TL
$-4(10^{10} + 1)$	7	9679	411	275	319	85	500	41
		19358	668	403	405	171	544	85
		29037	744	479	491	257	586	127
		38716	820	555	577	343	630	171
		48395	896	631	665	431	674	215
$-4(10^{15} + 1)$	3	51491	920	655	4016	23	7984	11
		102980	1433	911	4042	49	7996	23
		154469	1633	1111	4068	75	8010	37
		205958	1835	1313	4094	101	8022	49
		257447	2035	1513	4120	127	8036	63
$-4(10^{20} + 1)$	7	371239423	88207	55423	81233	10499	146696	5249
		742478843	110865	78081	91733	20999	151946	10499
		1113718263	165072	99519	102233	31499	157196	15749
		1484957683	176402	110849	112733	41999	162446	20999
		1856197103	187732	122179	123233	52499	167696	26249

TABLE 6. DL algorithm — run times

Δ	p	x	$v = 2$	$v = \Delta^{1/4}/2$	$v = \Delta^{1/4}$
$-4(10^{10} + 1)$	7	9679	0.34 sec	0.20 sec	0.24 sec
		19358	0.53 sec	0.28 sec	0.28 sec
		29037	0.58 sec	0.36 sec	0.32 sec
		38716	0.66 sec	0.45 sec	0.37 sec
		48395	0.56 sec	0.40 sec	0.34 sec
$-4(10^{15} + 1)$	3	51491	1.28 sec	1.55 sec	3.13 sec
		102980	1.80 sec	1.51 sec	3.09 sec
		154469	2.05 sec	1.67 sec	2.73 sec
		205958	2.20 sec	1.44 sec	2.63 sec
		257447	1.53 sec	1.40 sec	3.22 sec
$-4(10^{20} + 1)$	7	371239423	130.04 sec	55.17 sec	80.03 sec
		742478843	179.09 sec	80.61 sec	90.70 sec
		1113718263	238.34 sec	101.31 sec	106.19 sec
		1484957683	264.40 sec	127.83 sec	113.06 sec
		1856197103	188.05 sec	105.26 sec	106.34 sec

In the DL Algorithm, the advantages of our method in comparison with Shanks' method become even clearer. This is due to the fact that for $\log_g d$ all values between 0 and $|\langle g \rangle|$ are possible, and they are equally probable (this is, however, not the case in cryptographic circumstances). Assuming that $|\langle g \rangle|$ is not known a priori, Shanks' original algorithm still uses the upper bound of the group order, i.e., $v = \Delta^{1/4}$, which causes the algorithm to perform far too many baby steps.

For $\Delta = -4(10^{10} + 1)$ and the prime ideal lying over 7, in which case the bound $v = \Delta^{1/4}$ is quite accurate, Shanks' algorithm is faster than our algorithm using $v = \Delta^{1/4}/2$ for about half of the computed logarithms in our experiments. For all DL's smaller than half of the order of I_7, our algorithm using $v = \Delta^{1/4}/2$ works better. The comparatively bad results for our algorithm using $v = 2$ are due to the fact that in the DL Algorithm each giant step causes two group multiplications and two table look-ups, so that too many giant steps cause twice as much unnecessary work as in the Order Algorithm. Note that computing discrete logarithms of elements generated by I_7 represents the best case for Shanks' algorithm and the worst case for our algorithm because I_7 is the largest prime ideal.

For $\Delta = -4(10^{15} + 1)$ and $\Delta = -4(10^{20} + 1)$, where the upper bounds of the group order are much larger than the orders of the largest prime ideals, our algorithm using $v = \Delta^{1/4}/2$ works better for almost all possible logarithms.

Thus, our experiments lead to similar conclusions to those related to Algorithm 2.2, namely that our algorithm works better when $\log_g d$ is small compared to $|\langle g \rangle|$ and when the upper bound on the order of the group is too large.

4. COMPUTING THE STRUCTURE OF A SUBGROUP

Given a generating system, i.e., a finite sequence $S = (g_1, \ldots, g_l)$, of a finite abelian group G that is given as described in the introduction, we want to find the

structure of G. By this we mean finding positive integers $m_1, \ldots, m_k$ with $m_1 > 1$, $m_j | m_{j+1}$, $1 \leq j < k$, and an isomorphism

$$(5) \qquad\qquad \phi : G \longrightarrow \mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_k\mathbb{Z}.$$

This isomorphism will be given in terms of the images of the elements of the generating system S. The integers m_i are the *invariants* of G.

We describe our method. For $\vec{z} = (z_1, \ldots, z_l) \in \mathbb{Z}^l$ we write

$$S^{\vec{z}} = \prod_{i=1}^{l} g_i^{z_i}.$$

A *relation* on S is a vector $\vec{z} \in \mathbb{Z}^l$ such that $S^{\vec{z}} = 1$. The set $L(S)$ of all relations on S is a lattice in $\mathbb{Z}^l$ of dimension l, since it is the kernel of the surjective homomorphism

$$\mathbb{Z}^l \longrightarrow G, \quad \vec{z} \mapsto S^{\vec{z}}.$$

Our approach will be to compute a basis $B = (\vec{b}_1, \ldots, \vec{b}_l)$ of $L(S)$. That basis is identified with a matrix of column vectors $\vec{b}_j = (b_{1j}, \ldots, b_{lj})^t$. Then the order of G is $|\det B|$. Using standard techniques, e.g. [6], [4], we compute the Smith normal form N of B and a matrix $U \in \mathbb{Z}^{l \times l}$ with $\det U \not\equiv 0 \mod |G|$ such that there exists $V \in \mathbb{Z}^{l \times l}$ with $\det V \not\equiv 0 \mod |G|$ and $N \equiv UBV \mod |G|$. This means that we do not need to find two unimodular transformations for the Smith normal form but only a left unimodular transformation $\mod G$. Therefore, modular techniques for finding the Smith normal form can be applied.

Let N be the diagonal matrix $\mathrm{diag}(1, \ldots, 1, m_1, \ldots, m_k)$ where $m_1 > 1$. Then $m_1, \ldots, m_k$ are the invariants of G. To define the map (5) delete in U all but the last k rows. Let $\vec{u}_1, \ldots, \vec{u}_l$ be the column vectors of that new matrix, $\vec{u}_j = (u_{1j}, \ldots, u_{kj})^t$. Set

$$\phi(g_j) = (u_{1j} \bmod m_1, \ldots, u_{kj} \bmod m_k), \qquad 1 \leq j \leq l.$$

Then ϕ is the isomorphism we were looking for.

We describe the computation of the basis B of the relation lattice $L(S)$. The matrix B will be an upper triangular matrix with positive entries on the diagonal.

Suppose that we have computed $\vec{b}_1, \ldots, \vec{b}_{j-1}$. We describe the computation of $\vec{b}_j$. The diagonal entry b_{jj} is the smallest positive integer such that $g_j^{b_{jj}}$ belongs to the subgroup of G generated by $\{g_1, \ldots, g_{j-1}\}$. The problem of finding $\vec{b}_j$ is therefore similar to a discrete logarithm problem. Indeed, our solution is an extension of Algorithm 3.1. Instead of a single initial step-width v we now use a vector $\vec{v} = (v_1, \ldots, v_l) \in 2\mathbb{N}^l$ of step-widths. Since there is a basis B with $0 \leq b_{ij} < b_{ii}$ for $1 \leq i < j \leq l$ we can write

$$b_{ij} = q_i B_i + r_i,$$

where $B_i = \lceil \sqrt{b_{ii}} \rceil$, $0 \le q_i < B_i$, and $0 \le r_i < B_i$. We also write

$$b_{jj} = y + r_j,$$

where $y = 2^k v_j q$ and $1 \le r_j \le 2^k v_j$ with k and q as in Lemma 2.1. Set

$$S_i = (g_1, \dots, g_i), \qquad 1 \le i \le l.$$

As in Algorithms 2.2 and 3.1 we proceed to find the smallest value of y such that

$$(6) \qquad\qquad S_{j-1}{}^{\vec{q}\vec{B}} g_j{}^y = S_j{}^{-\vec{r}},$$

where

$$(7) \qquad \vec{q} = (q_1, \dots, q_{j-1}), \qquad 0 \le q_i < B_i,\ q_i B_i < b_{ii},\ 1 \le i < j,$$

$\vec{B} = (B_1, \dots, B_{j-1})$, $\vec{q}\vec{B}$ denotes the componentwise multiplication, and

$$(8) \qquad \vec{r} = (r_1, \dots, r_j), \qquad 0 \le r_i < B_i, 1 \le i < j,\ \text{and } 1 \le r_j \le 2^k v_j.$$

Once y is found we have

$$\vec{b}_j = \vec{q}\vec{B} \circ (y) \circ \underbrace{(0, \dots, 0)}_{l-j} + \vec{r} \circ \underbrace{(0, \dots, 0)}_{l-j}$$

where $\circ$ denotes the concatenation of vectors.

To be able to check quickly whether (6) holds for a given y we use the sets

$$Q = \left\{ (S_{j-1}{}^{\vec{q}\vec{B}}, \vec{q}) \ : \ \vec{q} \text{ as in (7)} \right\}$$

and

$$R = \left\{ (S_j{}^{\vec{r}}, \vec{r}) \ : \ \vec{r} \text{ as in (8)} \right\}.$$

Just as in the order algorithm and the discrete logarithm algorithm we check whether $b_{jj} \le v_j$ already during the first computation of R, i.e., when $k = 0$ in (8). For this we use the set

$$R' = \left\{ (S_{j-1}{}^{\vec{r}} \ : \ 0 \le r_i < B_i,\ 1 \le i < j \right\}.$$

Moreover, before computing any element of R we check if $b_{jj} = 1$. This is done separately, because in many cases the algorithm will compute the entire group structure with only a few generators, and all the others can be handled by this special case. For this check we use the set Q instead of R', since in general Q is considerably smaller than R'.

Here is the algorithm which determines the HNF-basis B for the relation lattice $L(S)$.

Algorithm 4.1.

This algorithm computes the HNF-basis for the lattice of relations on a generating system for a finite abelian group.

INPUT: A generating system $S = (g_1, \ldots, g_l)$ of G,
             initial step-width $\vec{v} = (v_1, \ldots, v_l) \in 2\mathbb{N}^l$.
OUTPUT: A basis $\mathcal{B} = (\vec{b}_1, \ldots, \vec{b}_l)$ of the lattice of relations
             on S in upper triangular form.

(1) $R' = \{(1, (\ ))\}$, $Q = \{(1, (\ ))\}$,   /* initialization */
        $\vec{B} = (\ )$
(2)   **for** $(j = 1, \ldots, l)$ **do**
(3) $s = 1$, $y = v_j$, $u = v_j$
(4) $h = g_j^{-1}$ /* $a = g_j^{-i}$, $b = g_j^{y}$,
(5) $a = 1$, $b = g_j^{v_j}$, $c = b$ $c = g_j^{u}$ */
(6) $R = \emptyset$
(7)   **for** (all $(e, \vec{q}) \in Q$) **do**   /* check whether g_j is
(8) $d = e * g_j$ contained in current
(9)   **if** (there is $\vec{r}$ such that $(d, \vec{r}) \in R'$ subgroup */
                    and $q_i + r_i < b_{ii} \forall i = 1, \ldots, j-1)$
                  then
(10) $\vec{b}_j = \vec{q}\vec{B} \circ (0) \circ (0, \ldots, 0) + \vec{r} \circ$
                    $(1) \circ (0, \ldots, 0)$
(11)   **break** for
(12)   **fi**
(13)   **od**
(14)   **while** $(b_{jj} == 0)$ **do**
(15)   **for** $(i = s, \ldots, u)$ **do**   /* new baby steps */
(16) $a = a * h$
(17)   **if** ($s == 1$ and $i > 1$) **then**   /* check whether
(18)   **for** (all $(d, \vec{r}) \in R'$) **do** $1 < b_{jj} \le v_j$ */
(19) $e = d * a$
(20)   **if** (there is $\vec{q}$ such that $(e, \vec{q}) \in Q$
                            and $q_i + r_i < b_{ii} \forall i = 1, \ldots, j-1$
) **then**
(21) $\vec{b}_j = \vec{q}\vec{B} \circ (0) \circ (0, \ldots, 0) + \vec{r} \circ$
                            $(i) \circ (0, \ldots, 0)$
(22)   **break** while
(23)   **else**
(24) $R = R \cup \{(e, \vec{r} \circ (i))\}$
(25)   **fi**
(26)   **od**
(27)   **else**
(28) $R = R \cup \{(d*a, \vec{r} \circ (i)) : (d, \vec{r}) \in$
                    $R'\}$
(29)   **fi**
(30)   **od**

```
(31)        while ((b_jj == 0) ∧ (y < u²))  do     /* giant steps */
(32)          for (all (e, q⃗) ∈ Q)  do
(33)            d = e * b
(34)            if ( there is r⃗ such that (d, r⃗) ∈ R
                     and q_i + r_i < b_ii ∀i = 1, ... , j−1
                   ) then
(35)              b⃗_j = q⃗B⃗ ∘ (y) ∘ (0, ... , 0) + r⃗ ∘
                  (0, ... , 0)
(36)              break while
(37)            fi
(38)          od
(39)          y = y + u
(40)          b = b * c
(41)        od
(42)        s = u + 1                             /* double step-
                                                     width */
(43)        u = 2u
(44)        c = c²
(45)      od
(46)      if (j ≠ l)  then
(47)        B_j = ⌈√(b_jj)⌉
(48)        B⃗ = B⃗ ∘ (B_j)
(49)        R' = {(d, r⃗∘(0)) : (d, r⃗) ∈ R'}∪     /* compute new
               {(d, r⃗) ∈ R : 1 ≤ r_j < B_j}         R' and Q */
(50)        Q = {(e, q⃗ ∘ (0)) : (e, q⃗) ∈ Q} ∪
               {(e * g_j^{q_j B_j}, q⃗ ∘ (q_j)) : (e, q⃗) ∈
               Q, 1 ≤ q_j < B_j, q_j B_j < b_jj}
(51)      fi
(52)    od
(53)    B = (b⃗_1, ... , b⃗_l)
(54)    return (B)
```

We implemented our algorithm using the LiDIA system [7] to compute, once again, in ideal class groups of imaginary quadratic orders. During the course of the implementation, we found that the sets R', R, and Q are most efficiently stored as indexed hash tables, since the algorithm requires traversing the tables in addition to fast searching. Using this data structure also allows one to maintain R' and R in one table. In the interest of saving storage, the exponent vectors in these tables are encoded into single integers.

For simplicity, we analyze the complexity of Algorithm 4.1 only with initial step-width $\vec{v} = (2, \ldots, 2)$. We need the following lemma.

Lemma 4.2. *Let $m_1, \ldots, m_l \in \mathbb{N}$ and $e = |\{m_j | m_j > 1\}|$. Then we have*

$$\prod_{j=1}^{l} \lceil \sqrt{m_j} \rceil \leq 2^{\frac{e}{2}} \cdot \prod_{j=1}^{l} \sqrt{m_j}.$$

Equality holds if and only if $m_j = 2$ for all m_j with $m_j > 1$.

Proof. Let $m \in \mathbb{N}$. If $m = 1$, $\lceil \sqrt{m} \rceil = \sqrt{m}$. If $m = 2$, $\lceil \sqrt{m} \rceil = \sqrt{2}\sqrt{m}$. If $3 \leq m \leq 5$, $\lceil \sqrt{m} \rceil < \sqrt{2}\sqrt{m}$. If $m \geq 6$, $1 < \sqrt{6}(\sqrt{2} - 1) \leq \sqrt{m}(\sqrt{2} - 1)$, thus $\lceil \sqrt{m} \rceil < \sqrt{m} + 1 < \sqrt{2}\sqrt{m}$. Hence, $\lceil \sqrt{m} \rceil \leq \sqrt{2}\sqrt{m}$ for all $m \geq 2$, with equality if and only if $m = 2$. From this our claim follows.   □

Theorem 4.3. *Let $\vec{v} = (v_1, \dots, v_l)$ with $v_1 = \cdots = v_l = 2$. Algorithm 4.1 executes l inversions and at most*

$$2^{\frac{l(S)}{2}}(l + 5)\sqrt{|G|} + 4l(S)\sqrt{|G|} + \log|G|$$

multiplications in G, where l denotes the number of generators and $l(S)$ denotes the number of the b_{jj} that are larger than 1. It uses two tables of at most

$$2^{\frac{l(S)}{2}}\sqrt{|G|} \quad \text{and} \quad 2^{\frac{l(S)}{2}+1}\sqrt{|G|}$$

pairs $(e, \vec{q}) \in G \times \{0, \dots, |G|\}^l$. The total number of table look-ups is bounded by $2^{\frac{l(S)}{2}}(l + l(S) + 2)\sqrt{|G|}$.

Proof. Algorithm 4.1 performs l outer loops to compute the column vectors $\vec{b}_1, \dots, \vec{b}_l$. We first estimate the sizes of the sets Q, R and R' for each of these outer loops, and how many group operations are needed to build up these sets. Let Q_j, R_j and R'_j denote the sets Q, R and R' after the j-th loop. We have

$$|Q_0| = 1 \quad \text{and} \quad |Q_j| \leq |Q_{j-1}| \cdot B_j,$$

where $B_j = \lceil \sqrt{b_{jj}} \rceil$. Thus $|Q_j| \leq \prod_{i=1}^{j} B_i$. Writing

$$(9) \qquad\qquad T = \prod_{i=1}^{l} B_i,$$

we get that $|Q_j| \leq T$ for all j. It takes at most

$$|Q_{j-1}| \cdot (B_j - 1) \leq (B_j - 1) \cdot \prod_{i=1}^{j-1} B_i$$

multiplications to compute Q_j given Q_{j-1}. The maximum number M_Q of multiplications to compute all the sets $Q_1, \dots, Q_{l-1}$ can be estimated as follows:

$$\begin{aligned} M_Q &\leq \sum_{j=1}^{l-1}(B_j - 1)\prod_{i=1}^{j-1} B_i = \sum_{j=1}^{l-1}\left(\prod_{i=1}^{j} B_i - \prod_{i=1}^{j-1} B_i\right) \\ &= \prod_{j=1}^{l-1} B_j - 1 \leq T - 1. \end{aligned}$$

In considering R_j, we use the arguments of the proof of Theorem 2.3, with initial step-width $v = 2$. At the end of the j-th loop, the set R ($= R_j$) consists of (at most) $p_j|R'_{j-1}|$ pairs $(d * g_j{}^i, \vec{r})$ with $d \in R'_{j-1}$ and $1 \leq i \leq p_j$, where $p_j = 0$ if $b_{jj} = 1$, $p_j = 1$ if $b_{jj} = 2$, $p_j = 2$ if $2 < b_{jj} \leq 4$, and $p_j \leq 2\lceil \sqrt{b_{jj}} \rceil - 2$ if $2 < \sqrt{b_{jj}}$. So we have $p_j \leq 2B_j - 2$ in all cases. Hence

$$|R_j| \leq (2B_j - 2)|R'_{j-1}|.$$

Moreover, the algorithm performs at most $2B_j - 2$ multiplications $a = a * h$ (step (16)), so that the maximum number of multiplications needed to compute R_j given

R'_{j-1} can be estimated by $(2B_j - 2)|R'_{j-1}| + 2B_j - 2$. From the way in which $|R'|$ is built up, we see that

$$|R'_j| = |R'_{j-1}| + (B_j - 1)|R'_{j-1}| = B_j|R'_{j-1}|,$$

where $|R'_0| = 1$. Thus,

$$(10) \qquad |R'_j| = \prod_{i=1}^{j} B_i, \qquad j = 1, \dots, l-1,$$

and

$$|R_j| \leq 2(B_j - 1) \prod_{i=1}^{j-1} B_i, \qquad j = 1, \dots, l,$$

so that we always have $|R'| \leq T$ and $|R| \leq 2T$. Therefore, the algorithm requires space for at most $4T$ pairs $(e, \vec{q}) \in G \times \{0, \dots, |G|\}^l$ to store Q, R and R'. Let

$$\begin{aligned} E &= \{g_j \,:\, j \in \{1, \dots, l\},\, b_{jj} \geq 2\} \\ &= \{g_j \,:\, j \in \{1, \dots, l\},\, G_{j-1} \neq G_j\}, \end{aligned}$$

where $G_j = \langle g_1, \dots, g_j \rangle$, $0 \leq j \leq l$. Let $l(S) = |E|$. Then the maximum number M_R of multiplications required to compute all the sets R_j satisfies

$$\begin{aligned} M_R &\leq 2 \left[\sum_{j=1}^{l} (B_j - 1) \prod_{i=1}^{j-1} B_i + \sum_{j=1}^{l} (B_j - 1) \right] \\ &= 2 \left[\prod_{j=1}^{l} B_j - 1 + \sum_{j=1}^{l} (B_j - 1) \right] \\ &\leq 2T - 2 + 2 \sum_{\substack{j=1 \\ g_j \in E}}^{l} b_{jj} \leq 2T - 2 + 2l(S) \cdot \sqrt{|G|}. \end{aligned}$$

Next, let us consider the remaining group multiplications and table look-ups in the l outer loops. It takes l inversions and l multiplications to compute g_j^{-1} and g_j^2 at the beginning of each outer loop. To check whether $b_{jj} = 1$, i.e., whether g_j is contained in the subgroup generated by $g_1, \dots, g_{j-1}$, the algorithm checks for at most $|Q|$ elements e whether $(e * g_j, \vec{r}) \in R'$ for some $\vec{r}$. In total, this requires at most

$$l \cdot |Q| \leq l \cdot T$$

multiplications and table look-ups. We only have iterations of the outer while loops ((14) - (45)) if $g_j \in E$. To check whether $1 < b_{jj} \leq 2 \,(= v_j)$, the algorithm checks for at most $|R'|$ elements d whether $(d * g_j^{-b_{jj}}, \vec{q}) \in Q$ for some $\vec{q}$. This requires at most $|R'| \leq (b_{jj} - 1)T = T$ table look-ups for each $g_j \in E$, so altogether at most

$$l(S) \cdot T$$

table look-ups (the multiplications have already been considered above). We only have further computations in the j-th outer loop if $2 < b_{jj}$. In this case, we have to distinguish between the cases $\sqrt{b_{jj}} \leq 2$ and $2 < \sqrt{b_{jj}}$. If $\sqrt{b_{jj}} \leq 2$, i.e., $b_{jj} = 3$ or 4, then b_{jj} is found during the first iteration of the outer while loop. We conclude from the proof of Theorem 2.3 that it takes at most $(B_j - 1)|Q_{j-1}| = |Q_{j-1}|$ table

look-ups and multiplications to check whether some element $(e * b, \vec{r})$ is in R (steps (33)/(34)), and it requires at most $B_j - 2 (= 0)$ multiplications $b = b * c$ (step (40)), plus one additional multiplication in step (44). If $2 < \sqrt{b_{jj}}$, we have in total

$$k_j = \left\lceil \log \frac{b_{jj}}{2} \right\rceil + 1 = \lceil \log b_{jj} \rceil$$

iterations of the outer while loop, which means in all $\lceil \log b_{jj} \rceil$ multiplications for squaring c in step (44). We use the results of the proof of Theorem 2.3 to see that in the inner while loops ((31) - (41)), the algorithm performs at most $(2B_j - 3)|Q_{j-1}|$ table look-ups and multiplications in steps (33)/(34) and at most $2B_j - 4$ multiplications in step (40). Combining the results of the cases $\sqrt{b_{jj}} \leq 2$ and $2 \leq \sqrt{b_{jj}}$, we get that for each $g_j \in E$ the algorithm performs at most

$$(2B_j - 3)|Q_{j-1}|$$

table look-ups in the inner while loops and at most

$$(2B_j - 3)|Q_{j-1}| + 2B_j - 4 + \lceil \log b_{jj} \rceil$$

group multiplications in the inner while loops and step (44). The maximum total number of table look-ups in the inner while loops, TL_I, can be estimated by

$$
\begin{aligned}
TL_I \quad &\leq \quad \sum_{j=1}^{l}(2B_j - 3)|Q_{j-1}| \leq 2\sum_{j=1}^{l}(B_j - 1)|Q_{j-1}| \\
&\leq \quad 2\sum_{j=1}^{l}(B_j - 1)\prod_{i=1}^{j-1} B_i \leq 2\left(\prod_{j=1}^{l} B_j - 1\right) \leq 2T - 2.
\end{aligned}
$$

For the maximum total number of multiplications in the inner while loops, M_I, we get

$$
\begin{aligned}
M_I \quad &\leq \quad 2T - 2 + 2\sum_{j=1}^{l}(B_j - 1) - 2l + \sum_{j=1}^{l}\lceil \log b_{jj} \rceil \\
&\leq \quad 2T + 2l(S) \cdot \sqrt{|G|} + \log|G| + l(S) - 2l - 2,
\end{aligned}
$$

where we use that $\sum_{j=1}^{l}\lceil \log b_{jj} \rceil \leq \log \prod_{j=1}^{l} b_{jj} + l(S) = \log|G| + l(S)$.

Summing up, Algorithm 4.1 performs l inversions, at most $(l+5)T + 4l(S)\sqrt{|G|} + \log|G|$ group multiplications, and at most $(l + l(S) + 2)T - 2$ table look-ups. It requires space for at most $4T$ pairs $(e, \vec{q}) \in G \times \{0, \dots, |G|\}^l$. Since $T \leq 2^{\frac{l(S)}{2}}\sqrt{|G|}$ (see Lemma 4.2), this completes the proof. $\qquad\square$

To get reasonable lower bounds for the total number of group multiplications and table look-ups we should treat many different distributions and orders of $\{b_{11}, \dots, b_{ll}\}$ separately. So we just give lower bounds for the sizes of Q, R and R', and the number of group multiplications required to compute these sets. This is done in the proof of the following theorem.

Theorem 4.4. *Let $\vec{v} = (2, \dots, 2)$ in Algorithm 4.1. Then the number of group multiplications and the table sizes are*

$$\Omega \left(\prod_{j=1}^{l} \left\lceil \sqrt{b_{jj}} \right\rceil \right),$$

where $b_{11}, \dots, b_{ll}$ denote the diagonal elements of the basis $\mathcal{B}$ computed by the algorithm.

Proof. Just as in the previous proof, let Q_j, R_j and R'_j denote the sets Q, R and R' after the j-th outer loop.

From the way Q is built up we get that $|Q_j| = |Q_{j-1}|$ if $b_{jj} = 1$, and $|Q_j| \geq |Q_{j-1}|(B_j - 1)$ if $B_j > 1$ (i.e., if $g_j \in E$). Thus $|Q_j| \geq \prod_{\substack{i=1 \\ g_i \in E}}^{j} (B_i - 1)$, so at the end of the algorithm we have

$$|Q| \geq \prod_{\substack{j=1 \\ g_j \in E}}^{l-1} (B_j - 1) \quad =: \quad L,$$

and it takes at least $L - 1$ multiplications to compute all the sets $Q_1, \dots, Q_{l-1}$. This lower bound is sharp: If $b_{jj} \leq 2$ for all j, then $Q = \{(1, \vec{0})\}$ at the end of the algorithm.

From the way R is built up and from Theorem 2.3 we see that $|R_j| = 0$ if $b_{jj} = 1$, $|R_j| \geq |R'_{j-1}|$ if $b_{jj} = 2$, and $|R_j| \geq B_j |R'_{j-1}|$ if $b_{jj} \geq 3$. With (10) it follows that

$$|R_j| \geq \frac{1}{2} \prod_{\substack{i=1 \\ g_j \in E}}^{j} B_i.$$

Let $p = \max\{j \in \{1, \dots, l\} \; : \; g_j \in E\}$. Then $|R_p| \geq 1/2 \cdot T$, and it takes at least $1/2 \cdot T$ multiplications to compute R_p, with T as in (9). It also follows from (10) that at the end of the algorithm we have that $|R'| = T/B_l$. (Especially, $|R'| = T$ if $g_l \notin E$.) This proves the theorem. □

Corollary 4.5. *Let $G = (\mathbb{Z}/2\mathbb{Z})^k$. Then Algorithm 4.1 is of complexity $\Omega(|G|)$.*

Proof. Let S be a generating sequence of G. Let $l = |S|$. Then for the diagonal elements b_{jj} computed by the algorithm we necessarily have k times $b_{jj} = 2$ and $l - k$ times $b_{jj} = 1$. Since $k = \log |G|$, the assertion follows by Lemma 4.2 together with Theorem 4.4. □

Let us further comment on the factor $2^{\frac{l(S)}{2}}$, which appears in Theorem 4.3. We conclude from Theorem 4.4 that the more cyclic subgroups G has and the smaller they are, the larger R and R' (and Q) are, and thus the storage required and the number of group multiplications increases. However, this effect only depends on the structure of the type of groups we are dealing with. For example, in the case of groups of points on elliptic curves over finite fields, which are either cyclic or isomorphic to a product of two cyclic groups, this phenomenon is not relevant. In the case of ideal class groups of imaginary quadratic fields, where we expect small ranks [5], we can say that the worst case does not occur very frequently, especially for large discriminants.

Theoretically there is another possibility to have a disturbingly large exponent $l(S)$ even if G is cyclic or consists of very large cyclic subgroups, namely if many generators are needed to build up each cyclic factor.

To estimate the damage caused by these effects, we did the following experiments. For $n = 3, 4, \ldots, 10$ we took the first 1000 discriminants smaller than 10^n. For each of these discriminants, we took the ten prime ideal classes of smallest norm in the corresponding imaginary quadratic order and used our algorithm to compute the subgroup generated by these classes. We measured the sizes of Q and R' and counted the number $l(S)$ of prime ideal classes actually used in the algorithm to compute the subgroup. In Table 7 we compare $|Q|$ and $|R'|$ with $|G|$ and $\sqrt{|G|}$, and in Table 8 we compare $l(S)$ with the number of cyclic factors of $|G|$, which is the minimum number of generators needed to compute $|G|$.

TABLE 7. Subgroup algorithm — size of R' and Q relative to $|G|$ and $|G|^{1/2}$

	$|Q|/|G|$		$|R'|/|G|$		$|Q|/\sqrt{|G|}$		$|R'|/\sqrt{|G|}$	
n	max	ave	max	ave	max	ave	max	ave
3	0.50000	0.18271	1.00000	0.42068	1.30931	0.74079	5.65685	1.67457
4	0.44444	0.11417	1.00000	0.28418	1.33333	0.71759	5.65685	1.74131
5	0.19048	0.06389	0.66667	0.16840	1.25988	0.69884	5.36656	1.80439
6	0.13187	0.03534	0.66667	0.09723	1.35710	0.68240	9.23760	1.83296
7	0.06028	0.01932	0.26667	0.05590	1.34739	0.66498	7.15542	1.89770
8	0.03896	0.01055	0.16000	0.03205	1.26179	0.64524	9.05097	1.93323
9	0.01755	0.00581	0.09231	0.01835	1.31979	0.63294	8.41976	1.96326
10	0.01094	0.00317	0.05424	0.01059	1.15923	0.61567	8.21715	2.02498

TABLE 8. Subgroup algorithm — worst case data

	$l(S)$ - number of cyclic factors			
n	0	1	2	3
3	549	423	28	0
4	520	422	58	0
5	505	415	77	3
6	525	385	84	6
7	505	402	86	7
8	537	380	77	6
9	552	373	66	9
10	536	384	75	5

Table 7 shows that usually the sets Q and R' do not blow up to the entire group size but contain less than $2\sqrt{|G|}$ pairs $(g, \vec{q})$ in average. Only for discriminants with absolute value smaller than 10^5 it occurred that $|R'| \geq |G|$. Due to the way Q is built up by Algorithm 4.1, the set Q always contains considerably fewer elements than R'.

Table 8 shows that in more than half of all our experiments the algorithm actually uses no more generators than theoretically necessary. Also in the remaining cases, $l(S)$ is always very small.

In Tables 9 and 10 we give some examples of subgroups computed with our algorithm. For each discriminant, we compute the group G generated by the classes of the 10 prime ideals of smallest norms in the order. As before, GM is the number

of group multiplications required and TL is the number of table look-ups. We used $\vec{v} = (2, \ldots, 2)$ as an initial step-width in all cases.

TABLE 9. Subgroup algorithm — sample run times

Δ	G	GM	TL	time
$-(10^2 + 3)$	[5]	50	14	0.03 sec
$-(10^3 + 3)$	[4]	44	13	0.04 sec
$-(10^4 + 3)$	[12]	59	17	0.05 sec
$-(10^5 + 3)$	[39]	92	31	0.06 sec
$-(10^6 + 3)$	[105]	122	51	0.08 sec
$-(10^7 + 3)$	[706]	332	149	0.20 sec
$-(10^8 + 3)$	[1702]	421	196	0.28 sec
$-(10^9 + 3)$	[1840, 2]	595	245	0.40 sec
$-(10^{10} + 3)$	[10538]	1038	369	0.73 sec
$-(10^{11} + 3)$	[31057]	2213	1067	1.60 sec
$-(10^{12} + 3)$	[62284, 2]	3223	1989	2.56 sec
$-(10^{13} + 3)$	[124264, 2, 2]	5794	2464	4.25 sec
$-(10^{14} + 3)$	[356368, 2, 2]	9233	3751	7.53 sec
$-(10^{15} + 3)$	[3929262]	23564	13182	18.41 sec
$-(10^{16} + 3)$	[12284352]	37249	16409	30.36 sec
$-(10^{17} + 3)$	[38545929]	67130	29484	1 min, 0.12 sec
$-(10^{18} + 3)$	[102764373]	103039	54913	1 min, 30.62 sec
$-(10^{19} + 3)$	[78425040, 2, 2, 2]	149197	83049	3 min, 39.47 sec
$-(10^{20} + 3)$	[721166712, 2]	343423	210837	6 min, 13.87 sec

TABLE 10. Subgroup algorithm — sample run times

Δ	G	GM	TL	time
$-4(10^2 + 1)$	[14]	63	18	0.03 sec
$-4(10^3 + 1)$	[10, 2, 2]	87	30	0.04 sec
$-4(10^4 + 1)$	[40, 4]	140	58	0.06 sec
$-4(10^5 + 1)$	[230, 2]	223	91	0.10 sec
$-4(10^6 + 1)$	[516, 2]	319	114	0.13 sec
$-4(10^7 + 1)$	[1446, 2]	598	243	0.26 sec
$-4(10^8 + 1)$	[4104, 4]	1223	640	0.71 sec
$-4(10^9 + 1)$	[2560, 2, 2, 2, 2]	1509	385	0.80 sec
$-4(10^{10} + 1)$	[48396, 2, 2]	3388	1622	1.93 sec
$-4(10^{11} + 1)$	[56772, 2, 2, 2]	4891	2660	2.54 sec
$-4(10^{12} + 1)$	[117360, 4, 2]	6680	3323	3.76 sec
$-4(10^{13} + 1)$	[742228, 2, 2]	12037	4233	7.11 sec
$-4(10^{14} + 1)$	[1159048, 4, 2, 2]	27615	11729	16.91 sec
$-4(10^{15} + 1)$	[257448, 4, 2, 2, 2, 2]	57387	22013	36.58 sec
$-4(10^{16} + 1)$	[11809616, 2, 2, 2]	120027	26425	1 min, 11.94 sec
$-4(10^{17} + 1)$	[46854696, 2, 2, 2]	134990	65584	1 min, 28.21 sec
$-4(10^{18} + 1)$	[264135076, 2, 2]	233224	94688	2 min, 40.54 sec
$-4(10^{19} + 1)$	[1649441906, 2]	572162	236334	6 min, 33.53 sec
$-4(10^{20} + 1)$	[1856197104, 2, 2, 2]	979126	380022	15 min, 39.46 sec

REFERENCES

[1] I. Biehl and J. Buchmann, *Algorithms for quadratic orders*, Proc. Sympos. Appl. Math., vol. 48, Amer. Math. Soc., Providence, RI, 1994. MR **95m**:11146

[2] J. Buchmann and S. Paulus, *Algorithms for finite abelian groups*, Extended abstract. To be published in the proceedings of NTAMCS 93.

[3] D.A. Buell, *Binary quadratic forms: classical theory and modern computations*, Springer-Verlag, New York, 1989. MR **92b**:11021

[4] H. Cohen, *A course in computational algebraic number theory*, Springer-Verlag, Berlin, 1993. MR **94i**:11105

[5] H. Cohen and H.W. Lenstra, Jr., *Heuristics on class groups of number fields*, Number Theory, Lecture Notes in Math., vol. 1068, Springer-Verlag, New York, 1984, pp. 33–62. MR **85j**:11144

[6] P.D. Domich, *Residual Hermite normal form computations*, ACM Transactions on Mathematical Software **15** (1989), no. 3, 275–286. MR **91d**:15020

[7] J. Buchmann I. Biehl and T. Papanikolaou, *LiDIA - a library for computational number theory*, The LiDIA Group, Universität des Saarlandes, Saarbrücken, Germany, 1995.

[8] A.K. Lenstra and H.W. Lenstra, Jr., *Algorithms in number theory*, Handbook of theoretical computer science (J. van Leeuwen, ed.), Elsevier Science Publishers, 1990, pp. 673–715. CMP 92:01

[9] S. Paulus, *Algorithmen für endliche abelsche Gruppen*, Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 1992.

[10] D. Shanks, *Class number, a theory of factorization and genera*, Proc. Symp. Pure Math. 20, AMS, Providence, R.I., 1971, pp. 415–440. MR **47**:4932

TECHNISCHE HOCHSCHULE DARMSTADT, INSTITUT FÜR THEORETISCHE INFORMATIK, ALEXANDERSTRASSE 10, 64283 DARMSTADT, GERMANY

E-mail address, J. Buchmann: `buchmann@cdc.informatik.th-darmstadt.de`

E-mail address, M. J. Jacobson, Jr.: `jacobs@cdc.informatik.th-darmstadt.de`

E-mail address, E. Teske: `teske@cdc.informatik.th-darmstadt.de`