MATHEMATICS OF COMPUTATION
Volume 69, Number 230, Pages 767–773
S 0025-5718(99)01141-2
Article electronically published on March 4, 1999

A MODIFICATION OF SHANKS' BABY-STEP GIANT-STEP ALGORITHM

DAVID C. TERR

ABSTRACT. I describe a modification to Shanks' baby-step giant-step algorithm for computing the order n of an element g of a group G, assuming n is finite. My method has the advantage of being able to compute n quickly, which Shanks' method fails to do when the order of G is infinite, unknown, or much larger than n. I describe the algorithm in detail. I also present the results of implementations of my algorithm, as well as those of a similar algorithm developed by Buchmann, Jacobson, and Teske, for calculating the order of various ideal classes of imaginary quadratic orders.

1. INTRODUCTION

Shanks' baby-step giant-step algorithm [1, 2] is a well-known procedure for finding the order n of an element g of a finite group G. Running it involves $2\sqrt{K}+O(1)$ group multiplications (GM), and $\sqrt{K}+O(1)$ table lookups (TL), where K is an upper bound on n (for instance, one often uses $K = |G|$). Often, however, K is unknown or much larger than n. In this case, a faster algorithm is desired. Here is my main result, to be proven later:

Theorem 1.1. *Let G be a group for which it is possible to compute the product of any two elements, to determine whether two elements are equal, and to determine whether a given element is equal to 1, the identity of G, and let g be an element of G of finite order $n > 2$. Then there exists a deterministic algorithm which determines the order n of g, using $GM = 2\lceil\sqrt{2(n-1)}\rfloor - 2$ group multiplications and $TL = \lceil\sqrt{2(n-1)}\rfloor - 1$ table lookups. (Here, $\lceil x\rfloor$ denotes the nearest integer to x.)*

2. THE ORDER ALGORITHM

My algorithm is similar to Shanks' in that one still compares powers g^{t_j} of g (giant steps) to an updated hash table of pre-computed consecutive powers g^i for $1 \le i \le v$ (baby steps). However, with my algorithm, one does not need an upper bound on n. Instead, the giant steps are not constant, but grow linearly. Specifically, the sequence $(t_j)_{j=0}^{\infty}$ is defined recursively as follows:

$$\begin{aligned} t_0 &= 2v; \\ t_{j+1} &= t_j + j + v + 1 \quad (j \ge 0). \end{aligned} \tag{1}$$

Received by the editor September 4, 1996 and, in revised form, May 30, 1998.
1991 *Mathematics Subject Classification.* Primary 68P10, 20C40.

It is straightforward to show that the solution of this recurrence is

$$t_j = (j+2)v + \tfrac{1}{2}j(j+1). \tag{2}$$

The hash table serves two purposes. First, after each step, one must use the last entry g^{j+v} in the table to compute g^{t_j} from $g^{t_{j-1}}$, since $t_j = t_{j-1} + j + v$. Second, the table of baby steps serves the same purpose as in Shanks' algorithm, i.e. every positive integer n can be expressed as the difference $t_j - i$, where t_j is the least number in the sequence (t_j) not exceeding n and $0 \le i < j + v$. The algorithm proceeds as follows. (My notation is the same as in [1].)

Algorithm 2.1. Input: $g \in G,\ v \in \mathbf{Z},\ v \ge 2$
Output: $n = |\langle g \rangle|$

(1) **if** $(g == 1)$ **then** /* *trivial case* */
(2) **return** (1)
(3) **else**
(4) $n = 0;\ i = 1;\ R = \{(1,0),\ (g,1)\};\ a = g$ /* *initialization* */
(5) **while** $(n == 0$ *and* $i < v)$ **do**
(6) $a = g * a;\ i = i + 1$ /* *baby steps* */
(7) **if** $(a == 1)$ **then**
(8) $n = i$ /* *found order among baby steps* */
(9) **return** (n)
(10) **else**
(11) $R = R \cup \{(a, i)\}$ /* *update hash table* */
(12) **fi**
(13) **od**
(14) $j = 0;\ b = a * a;\ t = 2 * v$ /* *initialize giant steps* $(t = t_j)$ */
(15) **while** $(n == 0)$ **do**
(16) **if** *(there exists a number* i *such that* $(b, i) \in R)$ **then** /* *table lookup* */

(17) $n = t - i$ /* *order of g* */

(18) **return** (n)

(19) **break while**

(20) **else**

(21) $a = g * a;\ j = j + 1;\ R = R \cup \{(a, j + v)\}$ /**baby step* */

(22) $b = a * b;\ t = t + j + v$ /* *giant step* */

(23) **fi**

(24) **od**

(25) **fi**

(26) **fi**

Theorem 2.2. *For the above algorithm, we have*

$$GM = 2\lceil\sqrt{2n + v(v-3)}\rfloor - v$$

and

$$TL = \lceil\sqrt{2n + v(v-3)}\rfloor - v + 1$$

if $n > v$. If $n \le v$, we have $GM = n - 1$ and $TL = 0$. The algorithm also requires storing a total of

$$|R| = \lceil\sqrt{2n + v(v-3)}\rfloor + 1$$

group elements in the hash table in the case $n > v$.

Proof. In the case where $1 < n \le v$, the order n of g will be found after computing the first n baby steps, which requires $n-1$ group multiplications, no table lookups, and no storage. (If $n = 1$, no work is required, other than noting that $g = 1$.) If $n > v$, the order n of g will be found after $j + v + 1$ entries of the table are computed, as well as g^{t_j}, where j is the least integer such that $n \le t_j$. We have

$$t_{j-1} = (j+1)v + \tfrac{1}{2}j(j-1) < n \le t_j = (j+2)v + \tfrac{1}{2}j(j+1).$$

Using straightforward algebra, we find that

$$j = \lceil\sqrt{2(n-1) + (v - \tfrac{3}{2})^2} - \tfrac{1}{2}\rceil - v = \lceil\sqrt{2n + v(v-3)}\rfloor - v.$$

To compute the last $j + v - 1$ entries of the table as well as g^{t_i} for $0 \le i \le j$, a total of

$$\mathrm{GM} = 2j + v = 2\lceil\sqrt{2n + v(v-3)}\rfloor - v$$

group multiplications and

$$\mathrm{TL} = j + 1 = \lceil\sqrt{2n + v(v-3)}\rfloor - v + 1$$

table lookups are required. (The first two elements of the table are not computed, just initialized.) It is also necessary to store the $|R| = j+v+1 = \lceil\sqrt{2n + v(v-3)}\rfloor + 1$ group elements g^i $(0 \le i \le j + v)$ in the table. QED

Setting $v = 2$, we find that

$$\mathrm{GM} = 2\lceil\sqrt{2(n-1)}\rfloor - 2$$

and

$$\mathrm{TL} = \lceil\sqrt{2(n-1)}\rfloor - 1,$$

if $n > 2$, proving Theorem 1.1. QED

The speed of my algorithm can be improved slightly, since g^i need not be computed twice if i belongs to the sequence (t_j) of giant steps. This will reduce GM by $O(\sqrt{j}) = O((n+v^2)^{\frac{1}{4}})$, without affecting TL. However, it also requires that one store an additional j group elements, so that as n gets large, the storage requirements are nearly doubled, whereas the computing time is only reduced by a fraction which tends to zero. Thus, this improvement is impractical for large n.

3. Comparison with algorithm of Buchmann, Jacobson, and Teske [1]

Here I present results of the implementation of my algorithm to the calculation of the orders of four ideal classes of each of three imaginary quadratic number fields with discriminants $\Delta = -4(10^8+1)$, $-4(10^{10}+1)$, and $-4(10^{12}+1)$. I wrote the program myself in Java, using Metrowerks CodeWarrior. I also wrote a version which implements the algorithm of Buchmann, Jacobson, and Teske [1], which I will refer to henceforth as the BJT algorithm. Both programs appear as Java applets on my webpage; the interested reader can run both algorithms and look at the source code [3]. For each of the twelve ideal classes I considered, I ran each algorithm with five different values of v, the initial giant step size, namely $v = 2$, $|\Delta|^{\frac{1}{4}}/8$, $|\Delta|^{\frac{1}{4}}/4$, $|\Delta|^{\frac{1}{4}}/2$, and $|\Delta|^{\frac{1}{4}}$.

In each of the following tables, I_p denotes the prime ideal lying above the rational prime p of the imaginary quadratic number field K with discriminant Δ, and (I_p) is the ideal class of the class group G=Cl(K) containing I_p. The numbers in bold indicate which algorithm involves the lesser of the quantity shown. In the case of a tie, each value is printed in italics. The numbers in brackets indicate which of the

Table 1. Order algorithm, $v = 2$

			My algorithm			BJT		
Δ	p	$\lvert\langle(I_p)\rangle\rvert$	GM	TL	$\lvert R\rvert$	GM	TL	$\lvert R\rvert$
$-4(10^8+1)$	5	228	**40**	**20**	[22]	41	21	**[16]**
	3	456	**58**	*29*	**[31]**	66	*29*	[32]
	7	1368	**102**	**51**	**[53]**	122	52	[64]
	11	4104	**180**	**90**	**[92]**	230	95	[128]
$-4(10^{10}+1)$	5	4033	178	**89**	[91]	**164**	94	**[64]**
	3	16132	358	**179**	[181]	**324**	189	**[128]**
	13	24198	**438**	**219**	**[221]**	485	221	256
	7	48396	620	**310**	[312]	**580**	316	**256**
$-4(10^{12}+1)$	11	13040	320	**160**	[162]	**299**	164	**[128]**
	59	23472	**432**	**216**	**[218]**	482	218	256
	5	29340	**482**	*241*	**[243]**	505	*241*	256
	3	117360	**966**	**483**	**[485]**	1005	484	512

TABLE 2. Order algorithm, $v = |\Delta|^{\frac{1}{4}}/8$

			My algorithm			BJT		
Δ	p	$\lvert\langle(I_p)\rangle\rvert$	GM	TL	$\lvert R\rvert$	GM	TL	$\lvert R\rvert$
$-4(10^8+1)$	5	228	[36]	**10**	28	**[35]**	12	**18**
	3	456	**[50]**	**17**	**35**	63	21	36
	7	1368	**92**	**38**	**56**	124	45	72
	11	4104	166	**75**	93	**162**	83	**72**
$-4(10^{10}+1)$	5	4033	**[154]**	**50**	106	184	64	112
	3	16132	**320**	**133**	**189**	389	156	224
	13	24198	**398**	**172**	228	425	192	**[224]**
	7	48396	576	**261**	317	**533**	300	**[224]**
$-4(10^{12}+1)$	11	13040	[299]	**62**	239	**[259]**	74	**176**
	59	23472	[381]	**103**	280	**[318]**	133	**[176]**
	5	29340	[421]	**123**	300	**[351]**	166	**[176]**
	3	117360	853	**339**	516	**783**	421	**[352]**

TABLE 3. Order algorithm, $v = |\Delta|^{\frac{1}{4}}/4$

			My algorithm			BJT		
Δ	p	$\lvert\langle(I_p)\rangle\rvert$	GM	TL	$\lvert R\rvert$	GM	TL	$\lvert R\rvert$
$-4(10^8+1)$	5	228	**45**	*6*	41	48	*6*	**36**
	3	456	55	**11**	46	**[54]**	12	**36**
	7	1368	**[89]**	**28**	**63**	115	36	72
	11	4104	159	**63**	98	**153**	74	**72**
$-4(10^{10}+1)$	5	4033	172	**31**	143	**[156]**	36	**112**
	3	16132	**[310]**	**100**	**212**	361	128	224
	13	24198	**[380]**	**135**	247	397	164	**[224]**
	7	48396	548	**219**	331	**505**	272	**[224]**
$-4(10^{12}+1)$	11	13040	422	**35**	389	**401**	36	**354**
	59	23472	474	**61**	415	**431**	66	**354**
	5	29340	502	**75**	429	**447**	82	**354**
	3	117360	[844]	**246**	600	**[696]**	331	**354**

five giant-step sizes used was optimal for the given ideal class and algorithm. Note that in the case of $\Delta = -4(10^{10}+1)$, and $v = 2$, $|\Delta|^{\frac{1}{4}}/2$, and $|\Delta|^{\frac{1}{4}}$ (middle four rows of Tables 1, 4, and 5) my results for the BJT algorithm agree with those in [1].

As can be seen from the above tables, the speed and storage of my algorithm are very close to that of Buchmann et al. In most cases, GM and TL are within 10% of each other. My method seems to have the advantage of requiring fewer table lookups; in each case, TL for my algorithm is no greater than for BJT. GM appears to comparable for the two algorithms, although mine seems to involve fewer group multiplications for small v than theirs. For larger v, their algorithm appears to work better. In the case of calculating orders of ideals of imaginary quadratic fields, my algorithm appears to be optimal for v near $|\Delta|^{\frac{1}{4}}/4$, whereas the BJT algorithm

TABLE 4. Order algorithm, $v = |\Delta|^{\frac{1}{4}}/2$

			My algorithm			BJT		
Δ	p	$\lvert\langle(I_p)\rangle\rvert$	GM	TL	$\lvert R\rvert$	GM	TL	$\lvert R\rvert$
$-4(10^8+1)$	5	228	**75**	*3*	74	81	*3*	**70**
	3	456	**81**	*6*	77	84	*6*	**70**
	7	1368	103	**17**	88	[**97**]	19	**70**
	11	4104	[157]	**44**	115	[**136**]	58	[**70**]
$-4(10^{10}+1)$	5	4033	256	**17**	241	**251**	18	**224**
	3	16132	348	**63**	287	[**305**]	72	**224**
	13	24198	402	**90**	314	[**341**]	108	[**224**]
	7	48396	[540]	**159**	383	[**449**]	216	[**224**]
$-4(10^{12}+1)$	11	13040	741	*18*	725	**738**	*18*	**708**
	59	23472	769	**32**	739	**753**	33	**708**
	5	29340	785	**40**	747	**761**	41	**708**
	3	117360	1005	**150**	857	**885**	165	**708**

TABLE 5. Order algorithm, $v = |\Delta|^{\frac{1}{4}}$

			My algorithm			BJT		
Δ	p	$\lvert\langle(I_p)\rangle\rvert$	GM	TL	$\lvert R\rvert$	GM	TL	$\lvert R\rvert$
$-4(10^8+1)$	5	228	**141**	*[1]*	*142*	153	*[1]*	*142*
	3	456	**145**	*[3]*	144	155	*[3]*	**142**
	7	1368	**157**	*[9]*	150	161	*[9]*	**142**
	11	4104	191	[**26**]	167	**180**	[28]	**142**
$-4(10^{10}+1)$	5	4033	**461**	[**8**]	455	467	[9]	**448**
	3	16132	513	[**34**]	481	**494**	[36]	**448**
	13	24198	547	[**51**]	498	**512**	[54]	**448**
	7	48396	639	[**97**]	544	**566**	[108]	**448**
$-4(10^{12}+1)$	11	13040	**1430**	*[9]*	1423	1437	*[9]*	**1414**
	59	23472	*1444*	*[16]*	1430	*1444*	*[16]*	**1414**
	5	29340	1452	*[20]*	1434	**1448**	*[20]*	**1414**
	3	117360	1572	[**80**]	1494	**1510**	[82]	**1414**

appears to work better for v near $|\Delta|^{\frac{1}{4}}/2$. Upon differentiating the equation for GM in Theorem 2.2 with respect to v (neglecting the integer rounding), we find that the optimal value of v should be asymptotic to $\sqrt{\frac{2}{3}n}$ for my algorithm. When one has no good lower bound on n, one should use $v = 2$, in which case my algorithm appears to be faster than the BJT algorithm.

4. A bound for similar methods

It would be of interest to find a lower bound on GM for all similar methods. For the purpose of this calculation, we will neglect table lookups; thus we will assume that every power of g calculated is tabulated as soon as it is compared with all previous table entries, assuming no match is found. Although this may make TL

unnecessarily high, it will ensure that GM is as small as possible. Let $(a_i)_{i=0}^{\infty}$ be the chronological sequence of powers of g calculated in determining n, where $a_0 = 0$ and $a_1 = 1$. (The sequence (a_i) must be infinite to allow for the determination of arbitrarily large n.) Since every power of g is obtained by multiplying two previously calculated powers of g, the sequence (a_i) must be an addition sequence, i.e. for every integer $k \geq 2$, we must have $a_k = a_i + a_j$ for some positive integers $i\,, j < k$, possibly with $i = j$. Also, every positive integer n must be expressible as the difference between two elements of the sequence, i.e. we must have $n = a_{i_2} - a_{i_1}$ for some i_1 and i_2. Aside from these two restrictions, we make no further restrictions on the sequence (a_i). Let $i(n) = \min\{\max(i_1, i_2) : n = a_{i_2} - a_{i_1}\}$. Then if the order of g is n, we require $i(n) - 1$ group multiplications to determine n.

Theorem 4.1. *There exist infinitely many positive integers m such that $i(m) \geq \sqrt{2m}$.*

Proof. Given a fixed positive integer n, let m be a positive integer not exceeding n such that $i(m)$ is maximal, i.e. we have $i(m) = \max\{i(k) : 1 \leq k \leq n\}$. Then every positive integer not exceeding n is the difference of two numbers in the sequence whose indices do not exceed $i(m)$. This implies

$$m \leq n \leq \binom{i(m)}{2} = \frac{i(m)(i(m)-1)}{2} \leq \frac{i(m)^2}{2},$$

the binomial coefficient being an upper bound on the number of distinct differences among the first $i(m)$ terms of the sequence. Since the right side grows at least as fast as n, infinitely many values of m are needed as n grows. Solving the above inequality for $i(m)$, we find that $i(m) \geq \sqrt{2m}$ for each of these m. QED

Theorem 4.1 may be restated as follows: The number of group multiplications required to calculate *arbitrary* n is bounded below by $C\sqrt{n}$, where

$$C := \limsup_{m \to \infty} \frac{i(m)}{\sqrt{m}} \geq \sqrt{2}.$$

In the case $v = 2$, my method yields a constant $C = 2\sqrt{2}$, twice the theoretical lower bound. Thus, any similar method would involve no less than half as many group multiplications as mine.

Acknowledgment

I would like to thank my advisor, H. W. Lenstra Jr., for inspiring me to write this paper and for providing me with some key ideas.

References

[1] J. Buchmann, M.J. Jacobson, Jr., and E. Teske, "On Some Computational Problems in Finite Abelian Groups", Math. Comp. 66 (1997), pp. 1663-1687. MR **98a:**11185

[2] D. E. Knuth, "The Art of Computer Programming", vol. 3 (1973), pp. 575-6 (prob. 17). MR **56:**4281

[3] "Dave's Cool Java Home Page", http://www.geocities.com/CapeCanaveral /Launch-Pad/5318 (1998).

2614 Warring St. #7, Berkeley, CA 94704
E-mail address: davidcterr@aol.com