LOWER BOUNDS FOR ARITHMETIC PROBLEMS

by

João Meidânis

Computer Sciences Technical Report #978

November 1990

Lower Bounds for Arithmetic Problems *

João Meidânis †

Computer Sciences Department

University of Wisconsin - Madison

joao@cs.wisc.edu

June 28, 1990

1 Introduction

In this paper we study the complexity of computing some arithmetic functions of integer inputs using the computation tree model (see [BO83], [Str83], [SY82], [PS81]). This model, which is described in some detail in Section 2, is equivalent to a random access machine (RAM) without indirect addressing, in the sense that programs for one model can be translated into programs for the other model with the same time complexity (up to constant factors). The instruction repertoire includes the four arithmetic operations $+, -, *, /$, plus the floor, div, and mod operations and conditional branching.

The running time in our model is defined as the number of arithmetic operations and comparisons executed, regardless of the complexity of the operation and size of operands, which is not very realistic. However, for the purpose of giving lower bounds, the use of a stronger model does not impair the usefulness of the results since they will hold also for any weaker model.

The main results of this paper are:

*Sponsored by the NSF (grant DCR-8552596) and by FAPESP, BRAZIL (grant 87/0197-2).

†Permanent address: DCC - IMECC - UNICAMP, Cx. Postal 6065, 13081 - Campinas - SP, BRAZIL.

- a lower bound of $\Omega(\sqrt{\log n})$ on the depth of any computation tree that computes the modular power $a^b \bmod m$, where a, b, m are n-bit integers. This bound also applies to RAMs with indirect addressing.
- a lower bound of $\Omega(\log \log n)$ on the depth of any computation tree that computes the Jacobi symbol $\left(\frac{a}{b}\right)$ where a, b are n-bit integers, with $b > 1$, b odd, and $\gcd(a, b) = 1$.

The first bound holds even for the weaker problem of deciding whether or not $2^a \bmod a = 2$ for an n-bit integer a. In our proofs we use results obtained by Mansour, Schieber, and Tiwari on computations with the floor operation (see [MST88b], [MST88a]).

Modular powering and the Jacobi symbol are largely used in cryptography and have several other number theoretical applications. Hence, there is considerable interest in determining their complexity, and this paper is a step towards that. We believe that many other number theoretical functions can be given lower bounds using the methods employed here.

Prior to our work, the best known lower bounds for these problems were as follows. In [vzG87], von zur Gathen showed that at least $\Omega(n)$ operations are needed for modular powering, but his operations were only the four basic arithmetic operations in the field Z_m, so the result is only valid for a fixed prime m, which is not part of the input. We don't know of any lower bounds for the Jacobi symbol, although any bound for modular powering implies a bound for the Legendre symbol, which is the special case of the Jacobi symbol when the second argument is prime.

The rest of this paper is organized as follows. Section 2 describes the computation tree model. Section 3 explains the way we derive lower bounds for this model. Section 4 shows how these ideas are employed in our particular case. Finally, Section 5 lists some possibilities for future work on this topic.

2 Computation trees

A *program* in the computation tree model is a rooted tree in which each node is labeled with an *instruction*. Instructions can be of two kinds: assignments and comparisons. Assignment instructions have the form

⟨result⟩ — ⟨operand1⟩ ⟨operator⟩ ⟨operand2⟩

where ⟨result⟩ is a variable name, ⟨operator⟩ is one of $+, -, *, /, \mathrm{mod}, \mathrm{div}$, and each operand is either the constant "1", a variable name, or an input name.

The mod and div operations are defined in terms of the *floor* function $\lfloor x \rfloor$ (which indicates the largest integer less than or equal to the real x) as follows.

$$\begin{aligned} a \text{ div } b &= \left\lfloor \frac{a}{|b|} \right\rfloor \\ a \bmod b &= a - |b|(a \text{ div } b) \end{aligned}$$

Notice that "/" stands for exact division: hence, even if the inputs are integers, intermediate results can be arbitrary rational numbers.

Assignment nodes have either one child or no children. Comparison nodes always have two children and bear instructions of the form

$$\langle \text{operand1} \rangle < \langle \text{operand2} \rangle$$

where ⟨operand1⟩, ⟨operand2⟩ are as above.

This completes our description of programs. Let's now specify how to compute the output given a program and values for each of its inputs.

The computation starts at the root and proceeds down until a leaf (childless node) is reached. The value computed at this leaf (which must be an assignment node) is the desired output. At each step the instruction of the current node is executed and then control passes to one of its children. For comparison nodes (the only ones where there is a choice), the left child is chosen if the comparison results "true", and the right child is selected if the result is "false."

Two things can go wrong during a computation: division by zero and attempt to use the value of an uninitialized variable. In both these cases we say that the computation *terminates abnormally*, as opposed to a normal termination that produces an output. We define the *domain* of a tree T, denoted by $\mathrm{domain}(T)$, as the set of input values that cause normal termination. For k-input trees, this is a set of k-tuples.

The *running time* of a program for a given input is the number of nodes visited during execution. The *depth* of a tree T, denoted by $\mathrm{depth}(T)$, is the number of nodes in the longest simple path from the root to a leaf. Hence, $\mathrm{depth}(T)$ is an upper bound for the running time of T over all inputs.

3 Main ideas

The way we prove the lower bounds in this paper can be viewed as a three step process. First, we find a *decision* problem that is reducible to the computation of the arithmetic function we are interested in. A decision problem is one for which there are only two possible outputs. For modular powering, we chose the question "is $2^a \bmod a = 2$?". For the Jacobi symbol, the function itself can be used since there are only two possible outputs, namely, $+1$ and -1. The reason we want decision problems is that we can assume without loss of generality that *if the executions for two inputs end up in the same leaf, they give the same output value.* This can be achieved appending a comparison node to the end of the program that tests the output and branches to compute the appropriate constant value. This observation is true for RAMs with or without indirect addressing also.

The second step is to show that for every program T it is possible to find a set S of inputs that will follow the same path down to a leaf during execution. In view of what we did in step 1, this means that all inputs in S give the same output.

The last step consists in showing that, if the depth of T is smaller than the claimed lower bound, the set S will be rich enough to contain inputs that give different outputs, contradicting the conclusion from steps 1 and 2.

4 Results

Since there are a few differences in the way we treat the problems according to the number of inputs, we will analyze each case separately, although the basic steps outlined in the last section apply to both cases.

4.1 One input problems

In this section we obtain the $\Omega(\sqrt{\log n})$ lower bound for the single input problem of deciding whether $2^a \bmod a = 2$. We will use the computation tree model here, although an extension to RAMs with indirect addressing can be obtained as in [MST88b].

We start by recalling two number theoretical results. The first one depends on standard estimates on

the distribution of prime numbers, for which [RS62] is a good reference. The second one is a consequence of Linnik's work [Lin44].

Lemma 1 *For any positive integer m, let $q(m)$ be the least prime not dividing m. Then*

$$q(m) \leq O(\log m).$$

Lemma 2 (Least Prime in an Arithmetic Progression) *If m, r, s are integers with $m, s > 1$ and $\gcd(r, m) = 1$ then the least prime solution to*

$$\begin{aligned} x &\equiv r \pmod{m} \\ x &> s \end{aligned}$$

satisfies $x \leq (ms)^{O(1)}$.

To complete our prerequisites, we state a theorem on single input computation trees proved in [MST88b]. Given a positive integer λ, we define

$$S(\lambda) = \{a \in Z \mid a > 1, \ a \equiv 1 \pmod{\lambda}\}.$$

Theorem 1 *Let T be a 1-input computation tree of depth h and domain D. There exists $\lambda \in Z$ such that for every input in $S(\lambda) \cap D$ the computation terminates normally at the same leaf of T. Furthermore,*

$$\lambda \leq 2^{2^{4h^2}}.$$

Actually, the theorem is proved for computation trees using the operations $+, -, *, /, \lfloor \cdot \rfloor, <$, but we know that mod and div can be simulated by these in a constant numbers of steps (see Section 2). Also, our definition of the set $S(\lambda)$ is slightly different, but the proof can be easily adapted to this case.

We are now ready for the main result.

Theorem 2 *Let T be a computation tree with depth h that decides whether or not $2^a \bmod a = 2$ for all n-bit integers a. Then $h \geq \Omega(\sqrt{\log n})$.*

Proof: Without loss of generality, we may assume that inputs which cause the execution to go to the same leaf generate the same output. According to Theorem 1, there is a $\lambda \in Z$ such that all n-bit inputs in $S(\lambda)$ should yield the same output.

Our goal is to exhibit two integers a_1, a_2 in $S(\lambda)$ that generate different outputs. Of course, at least one of them should not be an n-bit integer to avoid a contradiction. Take a_1 as the least prime in $S(\lambda)$. By Fermat's Theorem, $2^{a_1} \bmod a_1 = 2$ and by Lemma 2 we have $a_1 \leq \lambda^{O(1)}$.

To construct a_2, first let q be the least prime not dividing λ, and then choose p as the least prime that satisfies

$$\begin{aligned} p &\equiv q^{-1} \pmod{\lambda} \\ p &> 2^q \end{aligned}$$

where q^{-1} is the inverse of q modulo λ (it exists since $\gcd(q, \lambda) = 1$). The bounds given by Lemmas 1 and 2 are

$$q \leq O(\log \lambda), \quad p \leq (2^q \lambda)^{O(1)} \leq \lambda^{O(1)}.$$

We claim that $a_2 = pq$ is the required element. Indeed, if $2^{a_2} \bmod a_2 = 2$ we would have

$$2^q \equiv 2^{pq} \equiv 2 \pmod{p},$$

contradicting $p > 2^q$. Notice that both a_1, a_2 are $\leq \lambda^{O(1)}$. As we said earlier, at least one of them should be greater than 2^n to avoid a contradiction, so

$$2^n \leq \lambda^{O(1)} \leq 2^{O(2^{4h^2})},$$

which implies the desired lower bound for h.

End of proof.

4.2 Two input problems

In this section we study the computation of the Jacobi symbol, which is a function $\left(\frac{a}{p}\right)$ of two relatively prime integers a, p with p odd and greater than 1, characterized by the following properties.

- If p is prime,

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue mod } p \\ -1 & \text{if } a \text{ is a quadratic nonresidue mod } p \end{cases}$$

- If p and q are odd integers (not necessarily prime),

$$\left(\frac{a}{pq}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$$

From now on, we will use the notation $\mathrm{Jac}(a,p)$ to indicate the Jacobi symbol. Given three positive integers $\alpha_1, \alpha_2, \alpha_3$, we define

$$S(\alpha_1,\alpha_2,\alpha_3) = \{\,(x,y) \in Z^2 \mid x > y^{\alpha_1},\quad y > \alpha_2,\quad \text{and}\quad x \equiv y \equiv 1 \pmod{\alpha_3}\,\}.$$

Given a positive integer δ, we define a polynomial transformation A_δ by the formula

$$A_\delta(x,y) = (2x^\delta - y, x).$$

If $\Delta = (\delta_1, \delta_2, \ldots, \delta_r)$ is a sequence of positive integers, we define A_Δ as the composition

$$A_\Delta = A_{\delta_1} \circ A_{\delta_2} \circ \ldots \circ A_{\delta_r}.$$

We will use the convention A_Δ = identity when $\Delta = ()$, the empty sequence.

Lemma 3 (Correspondence Property) *If* $x, y > 1$, $\gcd(x,y) = 1$, *and* $x \equiv y \equiv 1 \pmod 4$ *then* $\mathrm{Jac}(x,y) = \mathrm{Jac}(A_\Delta(x,y))$ *for any* Δ.

Proof: It suffices to prove for A_δ; the general result then follows by induction. If $(u,v) = A_\delta(x,y) = (2x^\delta - y, x)$, notice that u and v are relatively prime, congruent to 1 modulo 4, and

$$\mathrm{Jac}(u,v) = \left(\frac{u}{v}\right) = \left(\frac{2x^\delta - y}{x}\right) = \left(\frac{-y}{x}\right) = \left(\frac{-1}{x}\right)\left(\frac{y}{x}\right).$$

But $\mathrm{Jac}(-1,x) = 1$ since x is 1 modulo 4, and $\mathrm{Jac}(y,x) = \mathrm{Jac}(x,y)$ by the quadratic reciprocity law. These are standard facts on the Jacobi symbol that can be found in, e.g., [Hua82]. We conclude that $\mathrm{Jac}(u,v) = \mathrm{Jac}(x,y)$, as claimed.

End of proof.

The following theorem, which corresponds to Lemma 10 in [MST88a], is analogous to Theorem 1 for 2-input trees.

Theorem 3 *Let T be a 2-input computation tree of depth h and domain D. There exist parameters $\alpha_1, \alpha_2, \alpha_3$ and a sequence $\Delta = (\delta_1, \delta_2, \ldots, \delta_r)$, with $r \leq h$, such that for every input in $A_\Delta(S(\alpha_1, \alpha_2, \alpha_3)) \cap D$ the computation terminates normally at the same leaf of T. Furthermore, for $i \in 1..r$,*

$$2 \leq \alpha_1 \leq 2^{2^{4h}},$$

$$\alpha_2, \alpha_3 \leq 2^{2^{2^{4h}}},$$

$$\delta_i \leq 2^{2^{4h}}.$$

There are a few differences between our setting and the one in [MST88a], but the proof given there can be easily adapted to work under our definitions. In particular, we use a different sort of A-transformation; however, the important properties of these transformations needed in the proof hold for our functions.

Theorem 4 *Let T be a computation tree with depth h that computes the Jacobi symbol for all pairs of n-bit integers (a, b), where b is odd, $b > 1$, and $gcd(a, b) = 1$. Then $h \geq \Omega(\log \log n)$.*

Proof: Without loss of generality, we assume that inputs which cause the computation to terminate at the same leaf have the same output. Let D_n be the set of all pairs (a, b) of odd, relatively prime n-bit integers with $b > 1$. By hypothesis, $D_n \subseteq \text{domain}(T)$. From Theorem 3 we have the parameters $\alpha_1, \alpha_2, \alpha_3$ and the sequence Δ with the stated properties.

Our goal is to exhibit pairs of relatively prime odd integers in $A_\Delta(S(\alpha_1, \alpha_2, \alpha_3))$ with different values for the Jacobi symbol. By the Correspondence Property, it suffices to exhibit such pairs in $S(\alpha_1, \alpha_2, \alpha_3)$, as long as all numbers involved are congruent to 1 modulo 4. We claim that the pairs (x_1, y) and (x_2, y) satisfy these conditions, where

$$y = 4\alpha_2\alpha_3 + 1,$$

$$x_1 = 4y^{\alpha_1}\alpha_3 + 1,$$

$$x_2 = z + y - yz + 4y^{\alpha_1}\alpha_3,$$

and z is any integer between 1 and y such that $\text{Jac}(z, y) = -1$.

It's easy to verify that (x_i, y) belongs to $S(\alpha_1, \alpha_2, \alpha_3)$, $\gcd(x_i, y) = 1$, and $x_i \equiv 1 \pmod 4$ for $i = 1, 2$. Moreover, $\text{Jac}(x_1, y) \neq \text{Jac}(x_2, y)$, since x_1 is congruent to 1 whereas x_2 is congruent to z modulo y.

We need to estimate the size of the actual inputs given to the tree. Notice that these inputs are not the pairs (x_i, y) themselves, but rather $A_\Delta(x_i, y)$. Using the conclusions of Theorem 3, we get

$$A_\Delta(x_i, y) \leq 2^{2^{2^{4h+3+r}}} \leq 2^{2^{2^{5h+3}}}.$$

where the inequalities are meant to hold for each component.

To avoid a contradiction, one of the four numbers a_1, a_2, b_1, b_2 has to be larger than or equal to 2^n, where $(a_i, b_i) = A_\Delta(x_i, y), i = 1, 2$, and from that we derive the relation

$$2^n \leq 2^{2^{2^{5h+3}}},$$

which implies the desired lower bound.

End of proof.

5 Conclusions

We believe that many other number theoretical problems, such as computing the square root modulo a prime, can be given lower bounds using similar arguments.

The lower bounds obtained here, although not trivial, are very small. For both problems a $O(n)$ upper bound is possible. It would be interesting to bring these bound closer to each other.

Although the inclusion of the floor operation is a step toward a more realistic model, the fact that only the number of operations is taken into account for the running time needs to be reviewed. In practice, multiplications and divisions are more expensive to compute than additions and subtractions, and the size of the operands also makes a difference.

Acknowledgements. I would like to thank my adviser, Prof. Eric Bach, for suggesting the problem, providing references, and for the constant encouragement and help ever since my early years as a graduate student at UW-Madison. I am also grateful to Prasoon Tiwari for copies of [MST88b], [MST88a], and for fruitful discussions, and to Anne Condon for valuable comments on the draft.

References

[BO83] M. Ben-Or. Lower bounds for algebraic computation trees. In *15th ACM Symp. on Theory of Computing*, pages 80–86, May 1983.

[Hua82] Hua Loo Keng. *Introduction to Number Theory*. Springer-Verlag, 1982.

[Lin44] U. V. Linnik. On the least prime in an arithmetic progression. *Mat. Sbornik*, 15:139–178, 1944.

[MST88a] Y. Mansour, B. Schieber, and P. Tiwari. A lower bound for integer greatest common divisor computations. Technical Report RC 14271, IBM T. J. Watson Research Center, Yorktown Heights, December 1988.

[MST88b] Y. Mansour, B. Schieber, and P. Tiwari. Lower bounds for computations with the floor operation. Technical Report RC 14272, IBM T. J. Watson Research Center, Yorktown Heights, December 1988.

[PS81] W. Paul and J. Simon. *Decision Trees and Random Access Machines*, volume 30 of *Monographies de l'Enseignement Mathématique*, pages 331–340. Université de Genève, 1981.

[RS62] J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Ill. J. Math.*, 6:64–94, 1962.

[Str83] V. Strassen. The computational complexity of continued fractions. *SIAM J. on Computing*, 12(1):1–27, February 1983.

[SY82] J. M. Steele and A. C. Yao. Lower bounds for algebraic decision trees. *J. of Algorithms*, 3:1–8, 1982.

[vzG87] J. von zur Gathen. Computing powers in parallel. *SIAM J. Comput.*, 16(5):930–945, October 1987.