Lower Bounds for the Complexity of Functions in a Realistic RAM model

Nader H. Bshouty
Department of Computer Science
The University of Calgary
Calgary, Alberta, Canada T2N 1N4
e-mail: bshouty@cpsc.ucalgary.ca

Abstract

This paper develops a new technique that finds lower bounds for the complexity of programs that compute or approximate functions in a *realistic RAM model*. The nonuniform realistic RAM model is a model that uses the arithmetic operations $\{+,-,\times\}$, the standard bit operations *Shift, Rotate, AND, OR, XOR, NOT* (bitwise), comparisons and indirect addressing. We prove general results that give almost tight bounds for the complexity of computing and approximating functions in this model. The functions considered here are integer division, modulo, square root, gcd and logarithms.

We also show that if we add the integer division to the realistic RAM model then no nontrivial lower bound can be proven.

Our results can be also generalized to probabilistic, nondeterministic and parallel RAM models.

1 Introduction

The literature contains many lower bound results for the complexity of computing (or approximating) functions in random access machines (RAMs) that use arithmetic operations, comparisons and indirect addressing. However, no nontrivial lower bound is known when the RAM model also uses bitwise boolean operations or bit shift operations. This paper is the first that attempts to define a *realistic* RAM and prove nontrivial lower bounds for computing and approximating functions in this model.

Our computation model is the nonuniform random access machine (RAM) with some fixed set of operations $F \subseteq \{+,-,\times,\text{DIV},\cdots\}$. An $F-$RAM, M,

has an unbounded memory $M[0], M[1], \cdots$, where each location can store an integer. The computation is directed by a finite program that consists of instructions of the following type: direct and indirect addressing storage accesses, conditional branching (IF-THEN-ELSE) and operations from F. Each of them can be executed at unit cost. For a function $f : \mathcal{N}^n \rightarrow \mathcal{N}^m$, where $\mathcal{N}$ is the set of nonnegative integers, the complexity of f in the $F-$RAM, $C_F(f,n)$, is the maximal number of steps taken over all inputs of size n and over all programs that compute f. We also define $C_F(f,\lambda,n)$ to be the complexity of $\lambda-$approximating f, i.e., the program outputs f^* where $\frac{1}{\lambda} \leq \frac{f^*}{f} \leq \lambda$.

The set of operations F considered in the literature are either the arithmetic operations or the arithmetic operations with the addition of integer division. The first known lower bound is $\Omega(n \log n)$ for sorting n elements using only comparisons ($F = \emptyset$). Ben Or [B] gave tight lower bounds for decision problems in the $\{+,-,\times,/\}-$RAM model without indirect addressing. Yao [Y] generalizes Ben Or's result by restricting the inputs to be integers. Both papers, [B] and [Y], give tight bounds for the complexity of computing the functions modulo, integer division and greatest common divisor (gcd) of two integers in the $\{+,-,\times,/\}-$RAM model without indirect addressing. Paul and Simon [PS] developed a new technique that handles RAM models with indirect addressing. The technique they used can be applied to prove tight bounds for the complexity of computing the functions modulo, integer division and gcd in the $\{+,-,\times,/\}-$RAM with indirect addressing. Bshouty [BS2,BS3] and Mansour, Shieber and Tiwary [MST2,MST3] gave other techniques that handle $\{+,-,\times,/\}-$RAM models with indirect addressing when the domain of the input is finite. Mansour, Shieber and Tiwary found a new technique that establishes lower bounds when the RAM model also contains the integer division DIV. They proved that there does not exist a program with complexity $O(1)$ that computes the gcd function in the $\{+,-,\times,/,\text{DIV}\}$-RAM model. Mansour, Shieber and Tiwari [MST*], and Bshouty, Mansour, Shieber and Tiwari [BMST], also apply these new techniques for computing the square root. Other results for this model can be found in [Bs0,Bs4,BJM,H,JM,S].

There are many other lower bounds in different RAM models but there were no nontrivial lower bound results for the complexity of functions in a *realistic computer model.* One that also uses bits operations such as bitwise AND, XOR and AND and bitwise shift operations such as Rotate and

Shift. In this paper we develop a new technique that handles RAM models of computation that contain the bitwise boolean operations and the bit shift and rotate operations. Our techniques find lower bounds for the complexity of computing and approximating functions in a realistic RAM model. The realistic RAM model is an $R-$RAM where

$$R = \{+, -, \times, \text{R-Shift}, \text{L-Shift}, \text{R-Rotate}, \text{L-Rotate}, \text{AND}, \text{OR}, \text{XOR}, \text{NOT}\}$$

(R- and L- stand for Right and Left). Our model also has indirect addressing and comparisons. All the lower bounds in this paper are true even if the RAM model has unlimited power for answering YES/NO questions. The boolean operations in the R-RAM can be executed on any consecutive bits. For example, L-Shift$(M[k], i, j)$ is an operation that shift the bits $i, i+1, \ldots, j$ in the content of $M[k]$ to the left. Therefore, if

$$M[k] = \cdots m_{i+1} m_i m_{i-1} \cdots m_{j+1} m_j m_{j-1} \cdots m_1 m_0,$$

is the binary representation of the k-th memory content then

$$\text{L-Shift}(M[k], i, j) = \cdots m_{i+1} m_{i-1} m_{i-2} \cdots m_{j+1} m_j 0 m_{j-1} \cdots m_1 m_0.$$

To the best of our knowledge, prior to this work, no lower bounds are known for the complexity of functions using the operations in R.

It is true that the model will be more realistic if we also add the integer division operation to the set of operation R, but it has been proven by Bshouty, Mansour, Shieber and Tiwary [BMST] that all functions with one variable can be computed with complexity $O(1)$ if the integer division DIV is added to the model. This implies that no nontrivial lower bound can be proven for functions with one variable in the model $R \cup \{DIV\}-$RAM. In this paper we push this result further. We prove that no nontrivial lower bound can be proven for any function (or any language, in the sense of decision problems) in the $R \cup \{DIV\}-$RAM model.

The following table summarizes our results. (Here $\tilde{\Omega}(k) = \Omega(k/\log k)$)

	Complexity	λ-approximation any constant λ	α-approximation
x^2	$\Omega(\frac{n}{\log n})$	$\Omega(\frac{\log n}{\log\log n})$	$\tilde{\Omega}(\log n - \log\log\alpha)$
$R\backslash\{\times\}$-RAM	$O(n)$	$O(n)$	$O(n)$
$x \times y$	$\Omega(\frac{n}{\log n})$	$\Omega(\frac{\log n}{\log\log n})$	$\tilde{\Omega}(\log n - \log\log\alpha)$
$R\backslash\{\times\}$-RAM	$O(n)$	$O(n)$	$O(n)$
$\mathrm{DIV}(x,y)$	$\Omega(\frac{n}{\log n})$	$\Omega(\frac{\log n}{\log\log n})$	$\tilde{\Omega}(\log n - \log\log\alpha)$
R–RAM	$O(n)$	$O(\log n)$	$O(\log n - \log\log\alpha)$
$x \bmod y$	$\Omega(\frac{n}{\log n})$	$\Omega(\frac{\log n}{\log\log n})$	$\tilde{\Omega}(\log n - \log\log\alpha)$
R–RAM	$O(n)$	$O(n)$	$O(n)$
$\lfloor\sqrt{x}\rfloor$	$\Omega(\frac{n}{\log n})$	$\Omega(\frac{\log n}{\log\log n})$	$\tilde{\Omega}(\log n - \log\log\alpha)$
R–RAM	$O(n)$	$O(\log n)$	$O(\log n - \log\log\alpha)$
$\gcd(x,y)$	$\Omega(\frac{n}{\log n})$	$\Omega(\frac{\log n}{\log\log n})$	$\tilde{\Omega}(\log n - \log\log\alpha)$
R–RAM	$O(n)$	$O(n)$	$O(n)$
$\lfloor\log x\rfloor$	$\Omega(\frac{\log n}{\log\log n})$	$\Omega(\frac{\log\log n}{\log\log\log n})$	$\tilde{\Omega}(\log\log n - \log\log\alpha)$
R–RAM	$O(\log n)$	$O(\log\log n)$	$O(\log\log n - \log\log\alpha)$
$\lfloor\log\log x\rfloor$	$\Omega(\frac{\log\log n}{\log\log\log n})$	$\Omega(\frac{\log\log\log n}{\log\log\log\log n})$	$\tilde{\Omega}(\log\log\log n - \log\log\alpha)$
R–RAM	$O(\log\log n)$	$O(\log\log\log n)$	$O(\log\log\log n - \log\log\alpha)$
$F(x_1,\ldots,x_k)$ k constant $R\cup\{DIV\}$–RAM	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
$F(x_1,\ldots,x_r)$ Any r $R\cup\{DIV\}$-RAM	$\Theta(r)$	$\Theta(r)$	$\Theta(r)$
Is $x \in L$ Finite $L \subset \mathcal{N}^r$ $R\cup\{DIV\}$-RAM	$\Theta(r)$	$\Theta(r)$	$\Theta(r)$

Table 1: Summary of Results

The first column in the table contains the functions that we are considering in this paper. The second column contains the complexities of exactly computing those functions. The third and forths columns contain the complexity of λ-approximating the functions for constant λ and α-approximating the functions for any α. Each row in the table is split into two. The upper one is for the lower bound and the lower one is for the upper bound. In the first two rows we have lower and upper bounds for computing the the square of an integer and the product of two integers in the $R\backslash\{\times\}$-RAM. Rows 3-8 contain lower and upper bounds for computing integer division, modulo, integer square root, greatest common divisor, integer logarithm and integer double logarithm. The last three rows in the table tell the reader that when we add the integer division DIV to our model then no nontrivial lower bound can be proven.

Notice that all the lower bounds in this paper are for the nonuniform R–RAM model and therefore they also apply for the uniform R–RAM

model. In the nonuniform model we allow one unit cost for exponential size of integers, unbounded memory and distinct programs for different input size so a lower bound in this model implies lower bound for any weaker model. Unfortunately, the upper bounds do not necessarily apply for the uniform RAM model (otherwise, the result in the previous paragraph proves P=NP and solves the integer factoring problem). The upper bounds we have in this paper are important in the sense that they show that it is impossible to prove certain lower bounds.

2 Definitions

Let F be a set of random access machine (RAM) operations. Any F-RAM program, P, can be regarded as a computation tree T_P with labeled vertices. The label of vertex v is denoted by l_v. The tree has five types of vertices:

- *Input vertex*: The root of the tree T_P is the input vertex that assigns the input of the program to $M[1], \ldots, M[n]$.
- *Computation vertices*: Each computation vertex v has one child and is labeled with a binary operation $M[i] \leftarrow M[j] \circ M[k]$ where $\circ \in F$.
- *Assignment vertices*: Each assignment vertex v has one child and is labeled with one of $M[i] \leftarrow a$ for some constant a, $M[i] \leftarrow M[j]$ or $M[i] \leftarrow M[M[j]]$.
- *Comparison vertices*: Each comparison vertex v has two children and is labeled with "$M[i] > 0$?".
- *Halt vertices*: The halt vertices are the leaves of T_P. Each leaf is labeled with $Halt$.

The computation in T_P begins at the root of the tree. We always execute the command in the vertex and go to its child. In the comparison vertex we go to the left child if $M[i] > 0$ and to the right child otherwise. The computation in T_P terminates when we arrive at a leaf. The outputs of the function are in $M[1], \ldots, M[s]$, where s is the number of outputs. It is obvious that the complexity of computing f in a program P is the height of the tree T_P.

When the program P does not contain IF-THEN-ELSE commands, then we say that P is a *straight line program.*

Let A and B be sets of integers. A *generator program* that generates B from A with the operations in F is a straight line program that uses only the constants in A and generates the constants in B using the operations in F. The complexity of a generator program is the number of operations used in the generator. We define $\Delta_F(A, B)$ to be the minimal number of operations needed to generate the integers in B using only the integers in A and the operations in F. We also define $\Delta_F^{(\lambda)}(A, t)$ to be the minimal number of operations needed to generate a constant c such that $t/\lambda \leq c \leq \lambda t$ using only the integers in A and the operations in F.

The set of operations we shall consider in this paper is

$$R = \{+, -, \times, \text{R-Shift}, \text{L-Shift}, \text{R-Rotate}, \text{L-Rotate}, \text{AND}, \text{OR}, \text{XOR}, \text{NOT}\}.$$

Here R-Shift and L-Shift are as defined in the introduction and

$$\text{AND}(a, b, i, j) = \cdots a_l \cdots a_{i+1}(a_i \wedge b_i) \cdots (a_j \wedge b_j) a_{j-1} \cdots a_0$$

where $\cdots a_l a_{l-1} \cdots a_0$ and $\cdots b_l b_{l-1} \cdots b_0$ are the binary representations of a and b, respectively, and $\wedge$ is the *and* boolean operation. It is easy to verify that the operations in R can be simulated in $O(1)$ using the operations

$$\{+, -, \times, \text{R-Shift}(a, i, 0), \text{NAND}(a, b, i, 0)\}$$

where for two bits x and y, x NAND $y = \bar{x} \wedge \bar{y}$. For example, L-Shift$(a, i, j)$ can be simulated in $O(1)$ steps using NAND, multiplication by 2 and the constants $2^{i+1} - 2^{j-1}$, 2^j and $2^j - 1$ as follows:

$b \leftarrow a$	$b = \cdots a_{i+1}$	$a_i \cdots\cdots a_j$	$a_{j-1} \cdots a_0$
$\text{NAND}(a, 2^{i+1} - 2^{j-1}, i, 0)$	$a = \cdots a_{i+1}$	$0 \cdots\cdots 0$	$\bar{a}_{j-1} \cdots \bar{a}_0$
$\text{NAND}(b, 2^j - 1, i, 0)$	$b = \cdots a_{i+1}$	$\bar{a}_i \cdots\cdots \bar{a}_j$	$0 \cdots 0$
$b \leftarrow 2 \times b$	$b = \cdots a_{i+1}\bar{a}_i$	$\bar{a}_{i-1} \cdots \bar{a}_j 0$	$0 \cdots 0$
$\text{NAND}(a, b, i, 0)$	$a = \cdots a_{i+1}$	$a_{i-1} \cdots a_j 1$	$a_{j-1} \cdots a_0.$
$a \leftarrow a - 2^j$	$a = \cdots a_{i+1}$	$a_{i-1} \cdots a_j 0$	$a_{j-1} \cdots a_0.$

We will also assume that when the RAM uses the subtraction $a - b$ then $a \geq b$. The RAM can always simulate computation with positive and negative numbers using only positive numbers. This can be done by saving each

number in two memory cells one contains the absolute value of the number and the other contains its sign. This gives $O(1)$ blow up time. Therefore, without loss of generality, we assume that

$$R = \{+, -, \times, \text{R-Shift}(a, i, 0), \text{NAND}(a, b, i, 0)\}$$

and the memory cells always contain positive integers.

All of the lower bounds in this paper are true even if the model has unlimited power for answering YES/NO questions.

Other notation that will be frequently used in this paper is the following. For a set, H, the number of elements in H will be denoted by $|H|$. Unless stated otherwise log will always mean $\log_2$. The set of integers is $\mathcal{N}$ and $\mathcal{N}_j = \{0, 1, \ldots, j\}$. For an integer a we will write $a = \cdots a_i a_{i-1} \cdots a_1 a_0$ for the binary representation of a where a_0 is the least significant bit of a. The number $[a]_{i,j}$ is $a_i a_{i-1} \ldots a_j$. For a vector $\bar{a} = (a_1, \ldots, a_n) \in \mathcal{N}^n$, $[\bar{a}]_{i,j} = ([a_1]_{i,j}, \cdots, [a_n]_{i,j})$ and for a set of vectors S we write $[S]_{i,j}$ for $\{[s]_{i,j} | s \in S\}$.

3 Complexity of exactly computing functions

3.1 The main theorems

In this subsection we will give a general theorem that implies lower bounds for the complexity of exactly computing functions.

The following lemma uses a simple counting argument to show that with "few" constants and operations there is an integer in a set of integers H that requires at least $\log |H| / \log \log |H|$ operations to be generated.

Lemma 1 *Let $H \subseteq \mathcal{N}^s$. Let W be a set of constants and F be a set of operations where $|F| + |W| \leq (\log |H|)^{O(1)}$. Then, there exists $(t_1, \ldots, t_s) \in H$ such that*

$$\Delta_F(W, (t_1, \ldots, t_s)) \geq \Omega\left(\frac{\log |H|}{\log \log |H|}\right).$$

(Here, $s = O(1)$ with respect to $|H|$)

Proof. The proof follows from a simple counting argument. The number of generator programs with at most h steps is less than or equal to

$$(|F||W|^2)(|F|(|W|+1)^2) \cdots (|F|(|W|+h-1)^2) \leq |F|^h (|W|+h)^{2h}.$$

This is because after l steps we have $|W| + l$ constants, so we can choose any two constants w_1 and w_2 ($\leq (|W| + l)^2$ possible choices) and choose an operation $\circ$ in F and then generate the new number $w_1 \circ w_2$.

Since the number of programs is greater than or equal to the number of elements in H modulo the order, we get

$$|F|^h (|W| + h)^{2h} > \frac{|H|}{s!}$$

This implies the result of the lemma.□

The next corollary shows that the bound in lemma 1 is tight when $F = \{+, \times\}$, $W = \{0, 1\}$ and $H = \{1, \cdots, t\}$.

Corollary 1. Let $F = \{+, \times\}$ and $W = \{1\}$. Any integer in $H = \{1, \cdots, t\}$ can be generated with $\log t / \log \log t$ operations.

Proof. Let $a \in H$ and let $a = a_l a_{l-1} \cdots a_0$ be the binary representation of a. We first generate all the numbers $0, 1, 2, 3, \ldots, 2^b - 1$ where $b = \lfloor \log l - \log \log l \rfloor$. This takes $2^b \leq l / \log l \leq t / \log t$ addition operations. Then compute $2^b, 2^{2b}, \ldots, 2^{\lfloor l/b \rfloor b}$. This takes $l/b \leq 2l / \log l \leq 2t / \log t$ multiplications. Now we can write a as

$$a = a^{(0)} + a^{(1)} 2^b + a^{(2)} 2^{2b} + \cdots + a^{(y)} 2^{yb}$$

where $0 \leq a^{(j)} \leq 2^b$ and $y \leq \lfloor \log a/b \rfloor \leq \lfloor l/b \rfloor$. We use this to compute a using $2y \leq 4t / \log t$ more operations. Notice that all $a^{(j)}$ were computed in the first round and 2^{jb} in the second.□

The main theorems in this section are:

Theorem 1 *Let $f : \mathcal{N}^r \to \mathcal{N}^s$ be an integer function. Let τ be an integer, $\bar{c} \in \mathcal{N}_{2^n}^r$ and $M \subseteq \mathcal{N}_{2^n}^r$ be a such that*

$$[M]_{\tau,0} = \{\bar{c}\}. \qquad (\star)$$

That is, all the elements in M agree on the least significant $\tau + 1$ bits. Then, the complexity of computing f for inputs of size n in the R-RAM model is

$$C_R(f, n) \geq \Omega\left(\frac{\log |[f(M)]_{\tau,0}|}{\log \log |[f(M)]_{\tau,0}|}\right).$$

Theorem 2 *Let $f : \mathcal{N}^r \to \mathcal{N}^s$ be an integer function. Let τ, ξ be an integer, $\bar{c} \in \mathcal{N}^r_{2^n}$ and $M \subseteq \mathcal{N}^r_{2^n}$ be such that*

$$[M]_{\tau,\xi} = \{\bar{c}\}.$$

Then, the complexity of computing f for inputs of size n in the $R\backslash\{\times\}$-RAM model is

$$C_{R\backslash\{\times\}}(f,n) \geq \Omega\left(\frac{\log|[f(M)]_{\tau,\xi}|}{\log\log|[f(M)]_{\tau,\xi}|}\right).$$

We will give the proof for theorem 1 here. The proof for theorem 2 is almost identical and will be left to the reader.

Proof of Theorem 1. The proof will be for the set of operations

$$R = \{+, -, \times, \text{R-Shift}(a,i,0), \text{NAND}(a,b,i,0)\}.$$

All the other operations in the realistic RAM model can be simulated by the operations of R in $O(1)$ steps. Let $H = [f(M)]_{\tau,0}$. Let P be a program that computes f for inputs of size n with complexity $C_R(f,n)$. We change the program to a computation tree T_P as described in section 2. If the number of leaves in T_P is greater than $|H|^{1/2}$, then the height of T_P, which is the complexity $C_R(f,n)$, is greater than $(1/2)\log|H| = \Omega(\log|[f(M)]_{\tau,0}|)$ and the result follows. Therefore we may assume that the number of leaves in the tree T_P is less than $|H|^{1/2}$. Since $|[f(M)]_{\tau,0}| = |H|$, and each input $x \in M$ terminates the computation at some leaf of the tree, there exists a leaf v in the tree such that the set of inputs $M' \subset M$ that arrive in the computation at this leaf satisfies $|[f(M')]_{\tau,0}| \geq |H|^{1/2}$ (Pigeon-hole principle). Let $H' = [f(M')]_{\tau,0}$. Then

$$h = |H'| \geq |H|^{1/2}.$$

The path from the root to the leaf v computes f for the inputs in M'. We now take this path and delete all the comparison vertices in it. We will be left with a straight line program that computes f for the inputs in M'. Let $P' \equiv P_1, P_2, \ldots, P_\eta$ be the instructions in this straight line program. If $\eta \geq \log|H|$ then the result follows because the height of the tree, which is the complexity of the program, is at least η. Therefore we may assume that the number of commands in this straight line program satisfies

$$\eta \leq \log|H|.$$

Now the proof proceeds as follows. We will change this program to generators that uses different operations and generate the constants in H'. This generator will have a complexity that is in the same order as the complexity of the program. Then we will use lemma 1 to pick up a constant in H' that is "hard to generate". This gives a lower bound on the number of operations for the generator and therefore for the program.

To formalize the above let $W_{P'}$ and $R_{P'}$ be the set of constants and the set of operations, respectively, that are used in the straight line program P'. Define the set of constants

$$\tilde{W} = (W_{P'} \bmod 2^{\tau+1}) \cup \{2^{\tau}, 2^{\tau+1} - 1\} \cup C,$$

where C is the set of entries of the vector $\bar{c}$ in $(\star)$. $\tilde{W}$ will be the set of constants that will be used in the generators. We now define a new set of operations $\bar{R}$. The set $\bar{R}$ contains the operations $\{+_{2^{\tau+1}}, -_{2^{\tau+1}}, \times_{2^{\tau+1}}\}$. Those are the arithmetic operations modulo $2^{\tau+1}$, i.e., $a +_{2^{\tau+1}} b = (a+b) \bmod 2^{\tau+1}$. It also contains NAND$(a, b, i, 0)$ for $0 \leq i \leq \tau$ and R-Shift$(a, i, 0)$ for $0 \leq i \leq \tau$. We define

$$\tilde{R} = (\bar{R} \cap \Lambda(R_{P'})) \cup \{+_{2^{\tau+1}}\}$$

where $\Lambda(R_{P'}) = \{\Lambda(z) | z \in R_{P'}\}$ and for $z \in R_{P'}$, we have

$$\Lambda(z) = \begin{cases} -_{2^{\tau+1}} & z = - \\ +_{2^{\tau+1}} & z = + \\ \times_{2^{\tau+1}} & z = \times \\ \text{NAND}(\star, \star, \tau, 0) & z = \text{NAND}(M[a], M[b], i, 0) \text{ and } i > \tau \\ \text{NAND}(\star, \star, i, 0) & z = \text{NAND}(M[a], M[b], i, 0) \text{ and } i \leq \tau \\ \text{R-Shift}(\star, \tau, 0) & z = \text{R-Shift}(M[a], i, 0) \text{ and } i > \tau \\ \text{R-Shift}(\star, i, 0) & z = \text{R-Shift}(M[a], i, 0) \text{ and } i \leq \tau \end{cases}.$$

The set $\tilde{R}$ will be the set of the operations that will be used in the generators. Since

$$|\tilde{W}| \leq |W_{P'}| + r + 2 \leq \eta + r + 2 \leq O(\log |H|)$$

and

$$|\tilde{R}| \leq |R_{P'}| + 1 \leq \eta + 2 \leq O(\log |H|),$$

by lemma 1, there exists $\bar{t} \in H'$ such that

$$\Delta_{\tilde{R}}(\tilde{W}, \bar{t}) \geq \Omega\left(\frac{\log |H'|}{\log \log |H'|}\right) = \Omega\left(\frac{\log |H|}{\log \log |H|}\right). \tag{1}$$

Now, we will show that

$$C_R(f,n) \geq \frac{1}{3}\Delta_{\tilde{R}}(\tilde{W},\bar{t}) \tag{2}$$

and then combining this with (1) will give the result. To prove (2) we will show how to change the straight line program P' to a generator program that generates $\bar{t}$ from $\tilde{W}$ using at most $3C_R(f,n)$ operations in $\tilde{R}$.

By the condition $(\star)$ in the theorem and since $\bar{t} \in H' = [f(M')]_{\tau,0}$ and $[M']_{\tau,0} \subseteq [M]_{\tau,0} = \{\bar{c}\}$, there exists $\bar{a} \in M'$ such that

$$[\bar{a}]_{\tau,0} = \bar{c} \quad , \quad [f(\bar{a})]_{\tau,0} = \bar{t}.$$

We now substitute $\bar{a}$ in the algorithm P' as an input and observe the least significant τ bits in the numbers generated by the algorithm P'. We will show that using the constants in $\tilde{W}$ and the operations in $\tilde{R}$ we will be able generate the first τ bits of the numbers generated in P'.

First we need to get rid of the indirect addressing in the algorithm. Since the input in P' is a fixed integer $\bar{a}$, all indirect addressing can be changed to direct addressing and then $M[i]$ in the algorithm can be replaced by its content. The resulting program $\tilde{P} = P'(\bar{a})$ is a generator program that generates $f(\bar{a})$ from the entries of $\bar{a}$ and the constants in $W_{P'}$ using the operations in $R_{P'}$. We now change the generator algorithm to a new generator algorithm that generates $\bar{t}$ from the constants in $\tilde{W}$ using only the operations in $\tilde{R}$. Step P_i in algorithm $\tilde{P}$ will be changed to step $\Gamma(P_i)$, defined as follows:

(1) We change all the constants a in the algorithm $\tilde{P}$ to $a' = a \mod 2^{\tau+1}$.

(2) If $P_i \equiv (a \leftarrow b \circ c)$ where $\circ \in \{+,\times\}$, then we change the step to

$$\Gamma(P_i) \equiv a' \leftarrow b' \circ_{2^{\tau+1}} c'.$$

(3) If $P_i \equiv (a \leftarrow b - c)$ then if $(b \bmod 2^{\tau+1}) \geq (c \bmod 2^{\tau+1})$ we change the step to

$$a' \leftarrow b' -_{2^{\tau+1}} c'$$

and if $(b \bmod 2^{\tau+1}) < (c \bmod 2^{\tau+1})$ then we change it to the following three steps

$$a''' \leftarrow (2^{\tau+1}-1) -_{2^{\tau+1}} b' \;\; ; a'' \leftarrow c' + 1; \;\; a' \leftarrow a''' + a''.$$

(4) If $P_i \equiv \mathrm{NAND}(a,b,i,0)$, then

$$\Gamma(P_i) \equiv \begin{cases} \mathrm{NAND}(a',b',\tau,0) & i > \tau, \\ \mathrm{NAND}(a',b',i,0) & i \leq \tau. \end{cases}$$

(5) If $P_i \equiv \text{R-Shift}(a, i, 0)$, then

$$\Gamma(P_i) \equiv \begin{cases} \text{R-Shift}(a', \tau, 0) & i > \tau, a_{\tau+1} = 0, \\ \text{R-Shift}(a', \tau, 0);\ a' \leftarrow a' + 2^\tau & i > \tau, a_{\tau+1} = 1, \\ \text{R-Shift}(a', i, 0) & i \leq \tau. \end{cases}$$

We will show that the new generator generates $\bar{t}$ from $\tilde{W}$ using the operations in $\tilde{R}$. To show this we will prove by induction that the constants generated in the new generator a' satisfy $a' = a \bmod 2^{\tau-1}$. First Notice that from (1) all the constants a used in $\tilde{P}$ are replaced by $a' = a \bmod 2^{\tau+1}$. In particular $\bar{a}$ is replaced by $\bar{a} \bmod 2^{\tau+1} = [\bar{a}]_{\tau,0} = \bar{c}$. Now assuming that $b' = b \bmod 2^{\tau+1}$ and $c' = c \bmod 2^{\tau+1}$ we have the following cases.

Case I. $P_i \equiv (a \leftarrow b \circ c)$ where $\circ \in \{+, \times\}$. Then

$$\begin{aligned} a' &= b' \circ_{2^{\tau+1}} c' \\ &= (b' \circ c') \bmod 2^{\tau+1} \\ &= ((b \bmod 2^{\tau+1}) \circ (c \bmod 2^{\tau+1}) \bmod 2^{\tau+1} \\ &= (b \circ c) \bmod 2^{\tau+1} = a \bmod 2^{\tau+1}. \end{aligned}$$

Case II. $P_i \equiv (a \leftarrow b - c)$. Then we have $b > c$. If $b \bmod 2^{\tau+1} \geq c \bmod 2^{\tau+1}$ then, as in case I, we have $a' = a \bmod 2^{\tau+1}$. If $b \bmod 2^{\tau+1} < c \bmod 2^{\tau+1}$ then we have

$$\begin{aligned} a' &= 2^{\tau+1} + b' - c' \\ &= (b - c) \bmod 2^{\tau+1} \\ &= a \bmod 2^{\tau+1}. \end{aligned}$$

Case III. $P_i \equiv \text{NAND}(a, b, i, 0)$. This command take the least significant i bits in b and takes the NAND of it with a puts the result in a. Since $a' = a \bmod 2^{\tau+1}$ is the least significant τ bits of a the NAND of the modulo is the modulo of the NAND.

Case IV. $P_i \equiv \text{R-Shift}(a, i, 0)$. This is a right shift of the least significant i bits of a to the right. If $i < \tau$ then the least significant τ bits of the shift is the shift of the least significant τ bits. If $i > \tau$ then the resulting shift depend on whether the $\tau + 1$ bit of a is 0 or 1. If it is 0 then we have the same as before. If it is 1 then we add this bit by adding 2^τ to the shifted number. In all cases a' will be $a \bmod 2^{\tau+1}$.

This complete the proof of the claim.

Since the number of steps in the new generator algorithm $\Gamma(\tilde{P})$ is less than or equal to 3 times the number of steps in $\tilde{P}$, the result (2) follows.□

Notice that in our proof we haven't used the fact that the queries in the comparison verteces are comparisons. Therefore the lower bound of the theorem follows even if the RAM model has unlimited power for answering YES/NO questions.

3.2 Lower bounds

In this subsection we show how to use Theorem 1 and 2 to prove lower bounds for the complexity of (exactly) computing functions.

Result 1. Let $E_2 : \mathcal{N}_{2^n} \to \mathcal{N}_{2^{2n}}$ be the integer function that computes the square of integers. Then

$$C_{R\setminus\{\times\}}(E_2) \geq \Omega\left(\frac{n}{\log n}\right).$$

Proof. We will use theorem 2. Let $M = \{1, 2, \ldots, 2^n\}$, $\tau = 2n$ and $\xi = n + 1$. We have $[M]_{2n,n+1} = \{0\}$ because for any $x \in M$, $[x]_{2n,n+1} = DIV(x, 2^{n+1}) = 0$. Now we show that $[E_2(M)]_{2n,n+1} \geq 2^n/3$. Then using theorem 2 the result follows.

Let $x > 2^{(2/3)(n+3)}$. Then

$$\begin{aligned}\left[E_2(x + \lceil 2^{n/3} \rceil)\right]_{2n,n+1} - [E_2(x)]_{2n,n+1} &= \left\lfloor \frac{(x + \lceil 2^{n/3} \rceil)^2}{2^{n+1}} \right\rfloor - \left\lfloor \frac{x^2}{2^{n+1}} \right\rfloor \\ &\geq \frac{x^2 + 2x2^{n/3}}{2^{n+1}} - \frac{x^2}{2^{n+1}} - 2 \\ &\geq 1.\end{aligned}$$

This shows that for $x = 2^{(2/3)(n+3)} + i\lceil 2^{n/3} \rceil$, $i = 1, \ldots, \lceil 2^{n/3} \rceil$, the integers $[E_2(x)_{2n,n+1}]$ are distinct. Therefore the set $[E_2(M)]_{2n,n+1}$ contains at least $2^n - 2^{(2/3)(n+3)} + 1$ distinct elements.□

Result 2. Let $\times : \mathcal{N}^2_{2^n} \to \mathcal{N}_{2^{2n}}$ be the integer function that computes the product of two integers. Then

$$C_{R\setminus\{\times\}}(\times) \geq \Omega\left(\frac{n}{\log n}\right).$$

Proof. Follows immediately from result 1.□

Result 3. Let $DIV : \mathcal{N}_{2^n}^2 \rightarrow \mathcal{N}_{2^n}$ be the integer division function. Then

$$C_R(DIV) \geq \Omega\left(\frac{n}{\log n}\right).$$

Proof. Let n be even and $M = \{(2^{n/2+1}z, 2^{n/2+1}) | z = 1, 2, \ldots, 2^{n/2-1}\}$. We choose $\tau = n/2$. Then $[M]_{\tau,0} = M \bmod 2^{n/2} = \{(0,0)\}$. Now since $z = 2^{n/2+1}z/2^{n/2+1} \in [DIV(M)]_{\tau,0}$ for $z = 1, 2, \ldots, 2^{n/2-1}$ we have $|[DIV(M)]_{\tau,0}| \geq 2^{n/2-1}$. Now by theorem 2 the result follows.□

Result 4. Let $\bmod : \mathcal{N}_{2^n}^2 \rightarrow \mathcal{N}_{2^n}$ be the modulo function. Then

$$C_R(\bmod) \geq \Omega\left(\frac{n}{\log n}\right).$$

Proof. Take $M = \{(2^{n/2}y, 2^{n/2} - 1) | y = 1, 2, \ldots, 2^{n/2} - 1\}$. Then $M \bmod 2^{n/2} = \{(0, 2^{n/2} - 1)\}$ and since $2^{n/2}y \bmod 2^{n/2} - 1 = y$ we have that $|[\bmod(M)]_{2^{n/2},0}| = 2^{n/2} - 1$. Now by theorem 1 the result follows.□

Result 5. Let $\surd : \mathcal{N}_{2^n} \rightarrow \mathcal{N}_{2^{n/2}}$ be the integer square root function. Then

$$C_R(\surd) \geq \Omega\left(\frac{n}{\log n}\right).$$

Proof. Let $n = 4k$. Consider $M = \{2^{n/2}y | y = 1, 2, \cdots, 2^{n/2} - 4\}$. Then $[M]_{n/2-1,0} = \{0\}$. Now,

$$\begin{aligned} \lfloor\sqrt{2^{n/2}(y+8)}\rfloor - \lfloor\sqrt{2^{n/2}y}\rfloor &\geq \sqrt{2^{n/2}(y+8)} - \sqrt{2^{n/2}y} - 2 \\ &= \frac{2^{n/4}8}{\sqrt{y+4} + \sqrt{y}} - 2 \\ &\geq 4 - 2 = 2. \end{aligned}$$

Therefore $\left|[\surd(M)]_{n/2-1,0}\right| \geq 2^{n/2}/16$ and by theorem 1 the result follows.□

Result 6. *Let $gcd : \mathcal{N}_{2^n}^2 \rightarrow \mathcal{N}_{2^n}$ be the integer function that computes the greatest common divisor of two integers. Then*

$$C_R(gcd, n) \geq \Omega\left(\frac{n}{\log n}\right).$$

Proof. Let $n = 4k$, $m = n/2$ and $A \in \{1, 3, 5, \ldots, 2^{n/4} + 1\}$ be an odd integer. Since A is relatively prime to 2^m there is an integer $b_A \leq A$ such that

$$b_A 2^m \equiv 1 \mod A.$$

Now define the set

$$M = \{(b_A 2^m - 1, A2^m) \mid A = 1, 3, 5, \ldots, 2^{\lfloor n/4 \rfloor} + 1\}.$$

Since $b_A 2^m - 1 \leq A2^m \leq 2^n$ we have $M \subset \mathcal{N}_{2^n}^2$. Take $\tau = \lfloor n/3 \rfloor$. Then $[M]_{\tau,0} = \{(2^{\tau+1} - 1, 0)\}$. Now, since

$$gcd(b_A 2^m - 1, A2^m) = gcd(b_A 2^m - 1, A) = A,$$

we have

$$[f(M)]_{\tau,0} = \{1, 3, 5, \ldots, 2^{\lfloor n/4 \rfloor} - 1\} \quad \text{and} \quad |[f(M)]_{\tau,0}| \geq 2^{\lfloor n/4 \rfloor - 1},$$

therefore, by Theorem 1,

$$C_R(gcd, n) \geq \Omega\left(\frac{\log |[f(M)]_{\tau,0}|}{\log \log |[f(M)]_{\tau,0}|}\right) = \Omega\left(\frac{n}{\log n}\right). \square$$

Result 7.

$$C_R(\log, n) \geq \Omega\left(\frac{\log n}{\log \log n}\right).$$

Proof. We take $M = 2^{n/2}\mathcal{N}_{2^{n/2}}$ and $\tau = \log n$. It can be shown that $[M]_{\tau,0} = \{0\}$ and $|[f(M)]_{\tau,0}| = n$, so by Theorem 1 the result follows.$\square$

A similar proof yields the following.

Result 8.

$$C_R(\log \log, n) \geq \Omega\left(\frac{\log \log n}{\log \log \log n}\right).$$

4 Complexity of λ-approximating functions

4.1 Lower bound

In this section we prove lower bounds for λ-approximating functions. To prove the main result we first prove a lemma similar to lemma 1.

Lemma 2 *Let W be a set of constants and F be a set of operations such than $|F| + |W| \leq (\log\log|H|)^{O(1)}$. Let $H = \{1, 2, \ldots, r\}$. Then there exists $t \in H$ such that*

$$\Delta_F^{(\lambda)}(W, t) \geq \Omega\left(\frac{\log\log|H| - \log\lambda}{\log\log\log|H|}\right).$$

Proof. The number of generator programs with $\leq h$ steps is less that or equal to

$$c \leq |F|^h(|W|)^2 \cdots (|W| + h)^2.$$

Let $G_1, \cdots, G_c$ be those generator programs and let $t_1 < t_2 < \cdots < t_c$ be the integers that they generate, respectively. Then G_i is a generator program that λ-approximates all the integers in $[t_i/\lambda, \lambda t_i]$. Next we will show that if $c < \log|H|/\log\lambda - 1$ then

$$\bigcup_{i=1}^{c}[t_i/\lambda, \lambda t_i] \neq H.$$

We first show by induction that $t_i \leq \sum_{j=1}^{i}\lambda^j$. Notice that if $t_1 > \lambda$ then the integer 1 in H is not generated by those generators which gives a contradiction. Therefore $t_1 \leq \lambda$. Assuming $t_i \leq \sum_{j=1}^{i}\lambda^j$, if $t_{i+1} > \sum_{j=1}^{i+1}\lambda^j$ then since

$$\frac{t_{i+1}}{\lambda} - \lambda t_i > 1$$

there is an integer between t_{i+1}/λ and λt_i that is not been approximated by any one of the generators. This gives a contradiction. Therefore

$$|H| \leq \lambda t_c \leq \lambda^{c+1}$$

which implies that c must be greater than or equal to $\log|H|/\log\lambda - 1$ and the result follows.□

The proof of the Theorem 3 is similar to the proof of Theorem 2.

Theorem 3 *Let $f : \mathcal{N}^r \to \mathcal{N}$ be an integer function. Suppose that for every integer n there exist integers τ, ξ, a vector $\bar{c} \in \mathcal{N}_{2^n}^r$ and a subset $M \subseteq \mathcal{N}_{2^n}^r$ such that*

$$[M]_{\tau,\xi} = \{\bar{c}\}.$$

If $[f(M)]_{\tau,\xi} = \{1, 2, \cdots, r\}$ *then*

$$C_{R\setminus\{\times\}}(f, \lambda, n) \geq \Omega\left(\frac{\log\log|[f(M)]_{\tau,\xi}| - \log\lambda}{\log(\log\log|[f(M)]_{\tau,\xi}| - \log\lambda)}\right).$$

If $\xi = 0$, *then*

$$C_R(f, \lambda, n) \geq \Omega\left(\frac{\log\log|[f(M)]_{\tau,0}| - \log\lambda}{\log(\log\log|[f(M)]_{\tau,0}| - \log\lambda)}\right).$$

This theorem provides all of the approximation results in Table 1.

4.2 Upper bounds

In this subsection we prove two upper bounds. The first upper bound proves the bounds for the square root, $\log n$ and $\log\log n$ in the table. The second result shows that adding integer division to the model solves all the problems in computer science (in the nonuniform model which allow one unit cost for exponential size of integers, unbounded memory and distinct programs for different input size).

Theorem 4 *Let* $f : \mathcal{N} \to \mathcal{N}$ *be any monotone nondecreasing function. Then*

$$C_R(f, n) \leq O(\log f(\mathcal{N}_{2^n}))$$

and

$$C_R(f, \lambda, n) \leq O(\log\log f(2^n) - \log\log\lambda).$$

Proof. We will save integers $v_1, f(v_1), \ldots, v_s, f(v_s)$ in a table. This sequence satisfies $v_1 < v_2 < \cdots < v_s$ and for every i we have

$$f(v_i) = f(v_i + 1) = \cdots = f(v_{i+1} - 1) \neq f(v_{i+1}).$$

Now $f(x)$ for any integer x can be computed using a binary search in the table for i such that $v_i \leq x < v_{i+1}$ and then $f(x) = f(v_i)$ is the value in the table after v_i.

We now prove the second statement. Consider the list $1, \lambda, \lambda^2, \cdots, \lambda^{\left\lceil \frac{\log f(2^n)}{\log\lambda} \right\rceil}$ with the list of $f^{-1}(\lambda^i)$. The program simply does a binary search for the

input x in the second list. If $f^{-1}(\lambda^{i-1}) \leq x \leq f^{-1}(\lambda^i)$, then λ^i is a λ-approximation for $f(x)$. The binary search takes

$$\log(\lceil \log f(2^n)/\log(\lambda) \rceil) = \log\log f(2^n) - \log\log\lambda + O(1)$$

operations.□

Obviously, the above proof gives a nonuniform program for computing $f(x)$.

Theorem 5 *For any function* $(f_1, \ldots, f_k)(x_1, \ldots, x_j)$, *we have*

$$C_{R\cup\{DIV\}}((f_1, \ldots, f_k), n) = \Theta(k + j).$$

Proof . Since the domain of the inputs is finite ($\mathcal{N}_{2^n}^j$), we may assume that $f_1, \ldots, f_k$ are polynomials (interpolation of a finite number of points). For $k = 1$ and $j = 1$ the proof can be found in [BMST]. The idea of the proof for any k and any j is the following. We pack the inputs $x_1, \ldots, x_j$ in one input (e.g. $X = x_1 + x_2 2^{n+1} + \cdots + x_j 2^{(j-1)(n+1)}$)and we pack the output $f_1, \cdots, f_k$ in one integer output. Then we use the Bshouty, Mansour, Shieber and Tiwary result [BMST] to compute a polynomial with one variable in $O(1)$ operations and then unpack the output. The costs of the packing and unpacking in $R \cup \{DIV\}$ are $O(k)$ and $O(j)$, respectively.□

5 OPEN PROBLEMS

We list two open problems

1. Find a nontrivial lower bound for computing functions in the $R \cup \{DIV\}$-RAM when the integers in the program are not allowed to grow exponentially large with the input size.

2. Are the lower bounds in this paper tight?

Acknowledgment. I would like to thank David Wilson for proofreading the paper.

References

[B] M. Ben-Or. Lower bound for algebraic computation trees. in *STOC* 1983, pp. 80-86.

[Bs0] N. H. Bshouty. Euclid's GCD algorithm is not optimal. Manuscript, 1990.

[Bs1] N. H. Bshouty. On the extended direct sum conjecture. *STOC* 1989, pp. 177-185.

[Bs2] N. H. Bshouty. Lower bounds for algebraic computation trees of functions with finite domains. TR-576, Technion, Israel, July 1989.

[Bs3] N. H. Bshouty. On the complexity of functions for random access machines. (To appear in Journal of ACM).

[Bs4] N. H. Bshouty. $\Omega(\log\log(1/\epsilon))$ lower bound for approximating the square root. TR No. 89/367/29, University of Calgary.

[BG] A. M. Ben-Amram, Z. Galil. Lower bounds for data structure problems on RAMs. *FOCS* 1991, pp. 622-631.

[BJM] L. Babai, B. Just, F Meyer auf der Heide. On the limits of computations with the floor function. *Information and Computation* 78,4, 99-107, (1988).

[BMST] N. H. Bshouty, Y. Mansour, B. Schieber, P. Tiwari. The complexity of approximating with the floor operation. Manuscript. Available at: http://www.cpsc.ucalgary.ca/ ~bshouty/ home.html.

[H] J. Hong. On lower bounds of time complexity of some algorithms. *Scientia Sinica*, **22**, 890-900, (1979).

[IMR83] O. H. Ibarra, S. Moran, L. E. Rosier. On the control power of integer division. *Theoretical Computer Science*, 24:35–52, 1983.

[JM] B. Just, F. Meyer auf der Heide, A. Wigderson. On computation with integer division. In *Proc. 5th STACS, Lecture Notes in Computer Science.* 294,pp. 29-37. Springer-Verlage, February 1988.

[Knu81] D.E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, Ma, second edition, 1981.

[MSM] S. Moran, M. Snir, U. Manber. Applications of Ramsey's theorem to decision tree complexity. *J. of ACM*, **32**, 938-949, (1985).

[MST1] Y. Mansour, B. Schieber, P. Tiwari. Lower bounds for integer greatest common divisor computation. In *FOCS* 1988, pp. 54-63.

[MST2] Y. Mansour, B. Schieber, P. Tiwari. Lower bounds for computations with the floor operations. In *Proceeding of ICALP*, 1989.

[MST2] Y. Mansour, B. Schieber, and P. Tiwari. The complexity of approximating the square root. *FOCS* 1989, pp. 325-330.

bibitem[PS]PS W. Paul, J. Simon. Decision trees and random access machines. In *Monographie 30, L'Enseignement Mathematique, Logic and Algorithmic — An International Symposium Held in Honor of Ernst Specker. Univ. Geneva Press,* 331-340, (1982).

[S] A. Schönhage. On the power of random access machines. 6th ICALP, 520-529, 1979

[Y] A. Yao. Lower bounds for Algebraic Computation Trees with Integer Inputs. *FOCS*, 1989.