A tight bound for approximating the square root

Nader H. Bshouty[*] Yishay Mansour[†] Baruch Schieber[‡] Prasoon Tiwari[‡]

Abstract

We prove an $\Omega(\log\log(1/\varepsilon))$ lower bound on the depth of any computation tree and any RAM program with operations $\{+,-,*,/,\lfloor\cdot\rfloor, \text{NOT}, \text{AND}, \text{OR}, \text{XOR}\}$, unlimited power of answering YES/NO questions, and constants $\{0,1\}$ that computes $\sqrt{x}$ to accuracy ε, for all $x \in [1,2]$. Since Newton method achieves such an accuracy in $O(\log\log(1/\varepsilon))$ depth, our bound is tight.

1 Introduction

In this paper consider the problem of approximating the square root of a given number up to accuracy ε. The models of computation considered in this paper are the computation tree model [Str72, Str83, Ben83], and the Random Access Machine (RAM) model [Sch79, AHU74]. It seems that the two computation models are incomparable. On one hand, a computation tree is a non-uniform model of computation, while a RAM is the corresponding uniform model of computation, and hence, weaker in this sense. On the other hand, a RAM is capable of indirect addressing while a computation tree is not.

We prove an $\Omega(\log\log(1/\varepsilon))$ lower bound on the depth of any computation tree and any RAM program with operations $\{+,-,*,/,\lfloor\cdot\rfloor, \text{NOT}, \text{AND}, \text{OR}, \text{XOR}\}$, unlimited power of answering YES/NO questions, and constants $\{0,1\}$ that computes $\sqrt{x}$ to accuracy ε, for all $x \in [1,2]$. This is an improvement of the $\Omega(\sqrt{\log\log(1/\varepsilon)})$ lower bound on the depth of any computation tree and any RAM program with operations $\{+,-,*,/,\lfloor\cdot\rfloor,<\}$ and constants $\{0,1\}$ that computes $\sqrt{x}$ to accuracy ε, for all $x \in [1,2]$, that is given in [MST89, MST91]. Our result should be contrasted with an $O(\sqrt{\log\log(1/\varepsilon)})$ *upper bound* for computing $\sqrt{x}$ to accuracy ε, for all $x \in [1,2]$, using either a computation tree or a RAM with operations $\{+,-,*,/,\lfloor\cdot\rfloor,<\}$ and constants $\{0,1,\varepsilon\}$ as proved in [MST89]. (See also [BMST92].) The worst-case arithmetic complexity of approximating zeros of polynomials in general is also discussed in [Ren87].

[*]Department of Computer Science, University of Calgary, Calgary, Alberta, Canada.
E-mail: `bshouty@cpsc.uclagary.ca`

[†]Department of Computer Science, Tel Aviv University, Tel Aviv, Israel 69978.
E-mail: `mansour@math.tau.ac.il`

[‡]IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, USA 10598.
E-mail: `{sbar,ptiwari}@watson.ibm.com`

Finally, it is well known that using Newton iterations we can design either a computation tree or a RAM program with operations $\{+, -, *, /, <\}$ and constants $\{0, 1\}$ that computes $\sqrt{x}$ to accuracy ε with depth $O(\log\log(1/\varepsilon))$. Therefore, our lower bound is tight.

2 The computation models

We assume that the reader is familiar with the computation tree model (see, e.g., [Str83, Ben83]) and the RAM model (see, e.g., [AHU74, Sch79]) used in this paper. Below, we briefly recall these models.

The computation tree model

A computation tree T for a one input problem is a tree with labelled vertices. The label of a vertex ν is denoted f_ν. The tree T has four types of vertices:

1. *An input vertex:* The root of the tree is the input vertex and it is labelled with the single input.

2. *Computation vertices:* Each computation vertex ν is labelled with either a function $f_\nu = g \circ h$, or a function $f_\nu = \circ g$, such that $g, h \in \mathcal{C} \cup \{f_\mu | \mu \text{ is an ancestor of } \nu \text{ in } T\}$ and $\circ \in OP$, where $\mathcal{C}$ is the set of available constants, and OP is the set of available operations. In this paper, $OP = \{+, -, *, /, \lfloor\cdot\rfloor, \text{NOT}, \text{AND}, \text{OR}, \text{XOR}\}$, and the set $\mathcal{C}$ will be restricted to the set $\{0, 1\}$. (Here, $\lfloor\rfloor g = \lfloor g \rfloor$, and NOT, AND, OR, XOR are bitwise operations.) Each computation vertex has only one child.

3. *Branching vertices:* Each branching vertex ν is labelled with an arbitrary YES/NO question that depends on $\mathcal{C} \cup \{f_\mu | \mu \text{ is an ancestor of } \nu \text{ in } T\}$. Each branching vertex has two children.

4. *Output vertices:* The output vertices are the leaves of T. They are labelled the same as computation vertices.

The computation for input a starts at the root of the tree T. When it arrives at a computation vertex ν, the function f_ν is evaluated at the input a, and then the computation proceeds to the only child of ν. When the computation arrives at a branching vertex the YES/NO question is evaluated. The computation proceeds to the left child if the answer is "YES", and to the right child otherwise. The computation terminates at a leaf by producing the value of the function associated with it as the output. The *depth* of a computation tree is the maximum length of a path from its root to one of its output leaves.

A computation tree is said to compute $\sqrt{x}$ to accuracy ε for all $x \in [1, 2]$, if for each input $x \in [1, 2]$ the computation follows a path to an output leaf whose label evaluates to some y such that $|y - \sqrt{x}| \le \varepsilon$.

The Random Access Machine (RAM) model

A RAM program is a sequence $((1 : \gamma_1), (2 : \gamma_2), \ldots, (r : \gamma_r))$, where each γ_i is either (i) a *common instruction* defined in [Sch79], or (ii) an instruction which applies an operation from the set of available operations OP to a set of operands, and stores the result in a memory location. A memory location can be accessed using either *direct* addressing, that is by specifying its address explicitly, or *indirect* addressing, that is by specifying an address of a location containing its address.

The input for a RAM program for a one input problem is given in the first memory location when the program starts. The *depth* of a RAM program is the maximum over all possible input instances of the number of instructions executed by the program given this input.

A RAM program is said to compute $\sqrt{x}$ to accuracy ε for all $x \in [1, 2]$, if for each input $x \in [1, 2]$, when the RAM executes the halt instruction the first memory location contains some y such that $|y - \sqrt{x}| \leq \varepsilon$.

In order to be able to consider both models simultaneously in the sequel we consider computation trees with indirect addressing capability. In this model we allow the operands of the functions associated with the vertices of the computation tree to be also of the form $\{f_{f_\mu} | \mu$ is an ancestor of ν in $T\}$. Clearly, this model is stronger than both models defined above.

3 The main lemma

In this section we prove the main lemma needed for our result. We present the proof due to W. Eberly because it is shorter than our proof.

Lemma 1 *Any computation tree with indirect addressing capability with operations* $\{+, -, *, /, \lfloor\cdot\rfloor, \text{NOT}, \text{AND}, \text{OR}, \text{XOR}\}$*, unlimited power of answering YES/NO questions, and constants* $\{0, 1\}$ *that computes a number* N *must have depth* $\Omega(\log\log(|N|))$*.*

Proof: Consider such a computation tree T that computes N. Since the only available constants are $\{0, 1\}$, all numbers computable by T are rational. For a computation (or output) vertex ν in T, let the value of f_ν denoted $v(f_\nu)$ computed at this vertex be $v(f_\nu) = s_i \frac{x_{i,1}}{x_{i,2}}$, where $s_i = sign(v(f_\nu))$, and $x_{\nu,1}, x_{\nu,2}$ are positive integers, $gcd(x_{\nu,1}, x_{\nu,2}) = 1$. For a vertex ν, define M_ν to be the maximum over all ancestors μ of ν in T of $\max\{x_{\mu,1}, x_{\mu,2}\}$. Clearly, for each ancestor μ of ν, $M_\nu \leq 2M_\mu^2$. This implies that if ν is at depth i, then $M_\nu \leq 2^{2^{i+1}-1}$. Suppose that N is computed at a vertex ν of depth t then $|N| = |v(f_\nu)| \leq M_t \leq 2^{2^{t+1}-1}$. The lemma follows. $\square$

4 The lower bound

In this section we prove our lower bound.

Theorem 2 *Any computation tree with indirect addressing capability with operations* $\{+, -, *, /, \lfloor\cdot\rfloor, \text{NOT}, \text{AND}, \text{OR}, \text{XOR}\}$*, unlimited power of answering YES/NO questions, and constants* $\{0, 1\}$ *that computes* $\sqrt{x}$ *to accuracy* ε*, for all* $x \in [1, 2]$ *must have depth* $\Omega(\log\log(1/\varepsilon))$.

Proof: Let T be a computation tree that computes an ε-approximation of $\sqrt{x}$ for all $x \in [1, 2]$. We look at the path that computes $\sqrt{2}$. Let ν be the output vertex at the end of this path. For this input it must be that $v(f_\nu) = \sqrt{2} + \delta$ where $\delta \in [-\varepsilon, \varepsilon]$. Since the only available constants are $\{0, 1\}$, all numbers computed, including $\sqrt{2} + \delta$, are rational, and therefore $\delta \neq 0$.

Now, we add five more computation vertices to this path. We show that the last vertex in the extended path computes a number whose absolute value is bigger than $1/\varepsilon$. By Lemma 1 this implies that the depth of the path must be $\Omega(\log\log(1/\varepsilon))$.

The new computation vertices are $\nu_1, \ldots, \nu_5$. Below we give the functions associated with each such vertex and how these functions are evaluated for the instance approximating $\sqrt{2}$.

FUNCTION	VALUE
$f_{\nu_1} = f_\nu * f_\nu$	compute $(\sqrt{2} + \delta)^2 = 2 + 2\sqrt{2}\delta + \delta^2$
$f_{\nu_2} = 1 + 1$	compute $2 = 1 + 1$
$f_{\nu_3} = f_{\nu_1} - f_{\nu_2}$	compute $2 + 2\sqrt{2}\delta + \delta^2 - 2 = 2\sqrt{2}\delta + \delta^2$
$f_{\nu_4} = f_{\nu_2} + f_{\nu_2}$	compute $4 = 2 + 2$
$f_{\nu_5} = f_{\nu_4} / f_{\nu_3}$	compute $4/(2\sqrt{2}\delta + \delta^2) = \sqrt{2}/\delta + 4/\delta^2$

It is easy to verify that $|\sqrt{2}/\delta + 4/\delta^2| \geq 1/\varepsilon$, and the theorem follows. □

Acknowledgement. We would like to thank Wayne Eberly and Lisa Higham for a number of helpful comments.

References

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Ma, 1974.

[Ben83] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. of the 15th ACM Symp. on Theory of Computing*, pages 80–86, Boston, May 1983.

[BMST92] N.H. Bshouty, Y. Mansour, B. Schieber, and P. Tiwari. Fast exponentiation using the truncation operation. *Computational Complexity*, 2:244–255, 1992.

[MST89] Y. Mansour, B. Schieber, and P. Tiwari. The complexity of approximating the square root. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 325–330, October 1989.

[MST91] Y. Mansour, B. Schieber, and P. Tiwari. Lower bounds for computations with the floor operation. *SIAM Journal on Computing*, 20(2):315–327, April 1991.

[Ren87] J. Renegar. On the worst-case arithmetic complexity of approximating zeros of polynomials. *Journal of Complexity*, 3:90–113, 1987.

[Sch79] A. Schönhage. On the power of random access machines. In *Proc. 6th ICALP, Lect. Notes. in Comp. Sci.* No. 71, pages 520–529, Graz, July 1979.

[Str72] V. Strassen. Berechnung und Programm I. *Acta Informatica*, 1:320–335, 1972.

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