Tables should be sorted (on random access machines)

Faith Fich
Peter Bro Miltersen

BRICS Report Series RS-95-26

ISSN 0909-0878 May 1995

Tables should be sorted (on random access machines)

Faith Fich and Peter Bro Miltersen
Department of Computer Science,
University of Toronto.

Abstract

We consider the problem of storing an n element subset S of a universe of size m, so that membership queries (is $x \in S$?) can be answered efficiently. The model of computation is a random access machine with the standard instruction set (direct and indirect adressing, conditional branching, addition, subtraction, and multiplication). We show that if s memory registers are used to store S, where $n \leq s \leq m/n^{\epsilon}$, then query time $\Omega(\log n)$ is necessary in the worst case. That is, under these conditions, the solution consisting of storing S as a sorted table and doing binary search is optimal. The condition $s \leq m/n^{\epsilon}$ is essentially optimal; we show that if $n + m/n^{o(1)}$ registers may be used, query time $o(\log n)$ is possible.

1 Introduction

In Yao's influential paper "Should tables be sorted?" [Yao81], the following basic data structure problem was considered: Given a subset S of size n of the universe $U = \{0, \ldots, m-1\}$, store it as a data structure $\phi(S)$ in the memory of a unit cost random access machine, using few memory registers, each containing an element of U, so that membership queries "Is $x \in S$?" can be answered efficiently for any value of x. The set S can be stored as a sorted table using n memory registers. Then queries can be answered using binary search in $O(\log n)$ time. Yao considered the possibility of improving this solution.

One of Yao's results was that, if for each S, $\phi(S)$ is a table of size n containing the elements of S in some order (i.e. the data structure is *implicit*) and m is sufficiently large compared to n, then query time $\Omega(\log n)$ is necessary and the sorted table is optimal. The proof is an elegant Ramsey theoretic argument. The lower bound holds in the *cell probe* model, i.e. only the number of memory cells accessed is considered.

However, Yao also observed that by allowing one extra element of U to be stored (which is interpreted as the name of a hash function), the lower bound can be beaten and constant query time is possible, if m is sufficiently large compared to n. This result was improved by Tarjan and Yao [TY79] and Fredman, Komlòs and Szemerédi [FKS84]. The latter paper shows that, for *all* values of m and n, there is a storage scheme using $n + o(n)$ memory cells, so that queries can be answered in constant time.

For most practical purposes, this answers the question of whether tables should be sorted: they should not. However, there are still some questions that remain unanswered. One concerns the exact amount of extra memory needed to get constant query time. The upper bound in [FKS84] has been improved by

[FNSS92] and [BM94]. It is still an open problem when exactly n memory cells are sufficient. In particular, Yao's lower bound does not apply if the implicitness restriction on ϕ is removed. Fiat, Naor, Schmidt, and Siegel [FNSS92], [FN93] consider bounds on the size of the universe m as a function of n for which there is an implicit data structure with constant query time. However, there is still a wide gap between their upper and lower bounds.

Instead of putting a restriction on the data structure, we can put a restriction on the query algorithm. The technique of [FKS84] is based on the family of hash functions $h_k(x) = (kx \bmod p) \bmod s$, i.e. integer division is used. However, the only arithmetic operations usually included in the instruction set of random access machines are addition, subtraction and multiplication. What can be done if only the standard instruction set is available? In this paper, we address this question, showing that on a unit cost random access machine with the standard instruction set, tables *should* be sorted. More precisely, we show:

- Let $U = \{0, \dots, m-1\}$ and let $n \leq s \leq m/n^{\epsilon}$ for some constant $\epsilon > 0$. There is an n-element subset S of U, so that for any representation $\phi_S \in \mathbf{Z}^s$ of S as a sequence of s integers, the following is true. Consider any RAM program that, on inputs x and ϕ_S, accepts if $x \in S$ and rejects if $x \notin S$. Then, for some $x \in U$, the program uses $\Omega(\log n)$ steps on inputs x and ϕ_S.

We first prove that, for any suggested representation scheme, some set is hard (i.e. query time is $\Omega(\log n)$ in the worst case). Then, using the universality of the RAM model, we show that some specific set is hard no matter what representation is used.

Note that we do not restrict the contents of the registers in the data structure to be elements from the universe U, as in the cell probe model. These values can be arbitrary integers, as is customary in the random access machine model. Furthermore, we do not require any bound on the complexity of constructing ϕ_S from S. Finally, the space bound s only refers to the data structure itself. We do not need to limit the number of registers used by the query algorithm.

In Section 3, we show that there is a data structure for the membership problem using space $O(\max(n, m/2^t))$ and with query time $O(t)$. Thus, the space upper bound $s \in m/n^{\Omega(1)}$ in our theorem is essentially optimal.

Our proof technique has a strong communication complexity flavor and can, in fact, be viewed as a modification of the *richness* technique used for showing lower bounds on the communication complexity of membership in [MNSW95]. Communication complexity has previously been used for showing lower bounds for data structure problems in the cell probe model [MNSW95]. Our proof is the first application of this kind of technique taking advantage of a restricted instruction set, while allowing memory registers to contain arbitrary integers. Another lower bound proof technique has previously been transferred from the cell probe model to the random access machine model: Ben-Amram and Galil [BG91] modified Fredman and Saks' *time stamp* technique for the cell probe model [FS89] to obtain the same lower bounds for the random access machine model, with registers that can contain arbitrarily large integers. An interesting feature of our technique is that we can get larger lower bounds in the random access machine model than in the cell probe model (where the complexity of membership is constant).

Instruction	Semantics
`load` k	The accumulator is assigned the integer constant k.
`direct read` k	The accumulator is assigned the value in register k.
`indirect read`	The accumulator is assigned the value in the register whose index is the value in the pointer.
`direct write` k	Register k is assigned the value in the accumulator.
`indirect write`	The register whose index is the value in the pointer is assigned the value in the accumulator.
`swap`	The accumulator and the pointer exchange values.
`add` k	The value in register k is added to the accumulator.
`subtract` k	The value in register k is subtracted from the accumulator.
`multiply` k	The accumulator is multiplied by the value in register k.
`conditional jump` j	If the value in the accumulator is positive, then the program counter is assigned the value j.
`accept`	The program halts in the accept state.
`reject`	The program halts in the reject state.

Table 1: RAM Instruction Set

2 The Random Access Machine Model

The random access machine (RAM) is an important and well studied model of sequential computation, first formalized by Cook and Reckow [CR73]. It has an infinite sequence of registers, indexed by the integers. Each register can contain an arbitrarily large integer. For ease of presentation, we include in our version of the RAM model two additional registers, the *accumulator* and the *pointer*. The accumulator is where arithmetic operations are performed. The pointer is used for indirect addressing.

The instruction set is described in Table 1. Here, as in [PS82, BG91, Maa88], we assume that addition, subtraction, and multiplication can be performed. In other papers, the RAM instruction set does not include multiplication [CR73] or restricts multiplication to reasonably small operands [AHU74].

A RAM *program* is a finite sequence of instructions. The *program counter* indicates which instruction to execute. In most cases, the program counter is incremented at the completion of each instruction. However, if the instruction `conditional jump` j is performed and the value in the accumulator is positive, the program counter is assigned the value j. The program counter is not updated when the instruction `accept` or `reject` is performed. We say that a computation *accepts* if the program counter eventually points to an `accept` instruction and that the computation *rejects* if the program counter eventually points to a `reject` instruction. The running time of a computation is the number of instructions executed until the program counter points to an `accept` or `reject` instruction. Thus `accept` and `reject` are free and all other instructions have unit cost.

3 Upper bounds

In this section, we present data structures for representing a set S and algorithms that query these data structures to determine if a given element x is in S. Initially, register 0 contains the input x and registers $1, \ldots, s$ contain the cells of the data structure $\phi(S)$. All other registers have initial value 0. A correct algorithm must accept inputs x and $\phi(S)$ if $x \in S$ and must reject them if $x \notin S$.

Although binary search normally involves division by 2, it can be implemented on a RAM. The idea is to precompute the first $\lfloor \log_2 n \rfloor$ powers of 2. The initial comparison is performed at location $2^{\lfloor \log_2 n \rfloor}$ and the locations of subsequent comparisons are obtained by either adding or subtracting successively smaller powers of 2. (See [Knu73, vol. 3 pages 413-414].) Alternatively, the full binary search tree containing the values in S can be represented implicitly with the left and right children of the node in location l at locations $2l$ and $2l+1$, respectively (as in a heap). (See [Ben86, pages 136,183–184] and [Knu73, vol. 1 page 401, vol. 3 pages 422,670].) Then the equivalent of binary search can be carried out using no extra registers.

Constant query time is possible with a bit vector representation, using m registers. By packing many bits together in one word, it is possible to use less space at the expense of more time. To do this, it is helpful to implement some functions which are not part of the standard RAM instruction set. We use the notation $[a..b]$ to denote the set of integers between a and b, inclusive.

Proposition 1 *A RAM can compute the quotient and remainder of x divided by y in time $O(\log(x/y))$.*

Proof: By repeated doubling, we construct an array containing the values y, $2y$, $4y$, …, $2^{t-1}y$, where t is the smallest value such that $2^t y > x$. Note that $t \in O(\log(x/y))$. The quotient q of x divided by y is the largest integer such that $q \cdot y \leq x$. Using the array and a variant of binary search, q as well as qy can be computed in $O(t)$ steps. Then we can compute the remainder by subtracting qy from x. $\square$ A RAM can shift a number left k bit positions by multiplying the number by 2^k. This takes constant time, if the number 2^k is available. Shifting right is harder, but it can be done in constant time in restricted situations.

Proposition 2 *With tables of size 2^{2^t} and 2^{t-1}, a RAM can shift a $(2^t - k)$-bit number right k bit positions in constant time, for any $k \in [1..2^{t-1}]$.*

Proof: Use a table containing the first 2^{t-1} powers of 2 and a lookup table of size 2^{2^t} whose l'th entry is the left half (i.e. the 2^{t-1} most significant bits) of l, when viewed as a 2^t-bit number. Then a 2^t-bit number can be shifted right by 2^{t-1} positions using one `indirect read` operation.

A $(2^t - k)$-bit number can be shifted right by k positions by first multiplying the number by $2^{2^{t-1}-k}$ (to shift it left $2^{t-1} - k$ positions) and then shifting the result right 2^{t-1} positions using the lookup table, as above. $\square$

Proposition 3 *There is a data structure for storing subsets of $[0..m-1]$ using $m/2^t + 2^{2^t} + 2^{t-1} + O(1)$ registers and $O(t)$ query time on a RAM.*

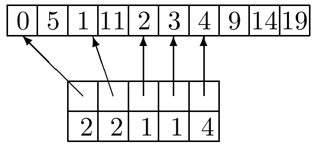
Proof: The main part of the data structure $\phi(S)$ is an array of $\lceil m/2^t \rceil$ cells, each containing a 2^t-bit non-negative integer. Together, the bits of these integers form a bit vector representation of S, as follows: Let q and r denote the quotient and remainder when x is divided by $\lceil m/2^t \rceil$. Then $x \in S$ if and only if the $(q+1)$'st least significant bit in cell $r+1$ of the array is 1. For example, if $m = 20$ and $t = 2$, the set $S = \{0, 1, 2, 3, 4, 5, 9, 11, 14, 19\}$ is represented by the following array.

0011	0101	0001	0001	1111

Given x, Proposition 1 can be used to compute r and q in time $O(t)$. Then the $(q+1)$'st least significant bit is extracted from the 2^t-bit integer in cell $r+1$, as follows: Using Proposition 2, first shift the integer right $q+1$ positions, then shift it left $q+1$ positions, and subtract the result from the integer. The resulting value is shifted right q positions to get the desired bit. $\Box$ Division can be combined with binary search to obtain the following data structure. It improves the solution in Proposition 3 for most natural choices of the parameters.

Proposition 4 *There is a data structure for storing n-element subsets of $[0..m-1]$ using $m/2^t + n + O(1)$ registers and $O(t)$ query time on a RAM.*

Proof: The data structure $\phi(S)$ representing S consists of three arrays. One array contains the elements of S, arranged so that all elements that are equivalent modulo $\lceil m/2^{t+1}\rceil$ are grouped together and, within each group, the elements are in sorted order. The second array consists of $\lceil m/2^{t+1}\rceil$ pointers to the beginning of the regions of the first array that contain elements equivalent to r modulo $\lceil m/2^{t+1}\rceil$ for $r \in [0..\lceil m/2^{t+1}\rceil - 1]$. The third array contains the number of elements in S equivalent to r modulo $\lceil m/2^{t+1}\rceil$ for $r \in [0..\lceil m/2^{t+1}\rceil - 1]$. In this representation, with $m = 20$, $t = 1$, and $S = \{0, 1, 2, 3, 4, 5, 9, 11, 14, 19\}$, this is what $\phi(S)$ looks like.



To determine whether x is in S, the remainder r of x divided by $\lceil m/2^{t+1}\rceil$ is computed in time $O(t)$ using Proposition 1. A pointer to the beginning of the relevant region of the first array and the size of this region are obtained from the second and third arrays. Binary search is then performed within this region. Since there are at most 2^{t+1} elements of S in the region, the search takes $O(t)$ time. $\Box$

4 Lower Bounds

Throughout this section, we use the notation $U^{(n)}$ to denote the set of n-element subsets of U. We begin with two simple lemmas about low-degree polynomials.

Lemma 5 *Let $D, R \subseteq \mathbf{R}$ and let $p(x) \in \mathbf{R}[x]$ be a polynomial of degree at most d. If there are more than $d \cdot \#R$ elements of D that are mapped to R by the polynomial p, then p is a constant polynomial.*

Proof: Let $D' = \{x \in D \mid p(x) \in R\}$ and suppose $\#D' > d \cdot \#R$. Then there are at least $d+1$ elements in D' that are mapped to the same element of R by p. Since p has degree at most d, it follows that p is constant. $\Box$

Lemma 6 *Let $Z \subseteq \mathbf{Z}$ be a set of integers at least distance g apart from one another (i.e. if $x, y \in Z$ are distinct integers, then $|x - y| \geq g$). Let $p(x) \in \mathbf{Z}[x]$ be a nonconstant polynomial of degree at most d. If $d \cdot s \leq g$, then at most d elements of Z are mapped to $[1..s]$ by p.*

Proof: Suppose, to the contrary, that $d \cdot s \leq g$, but there exist $d+1$ integers $x_0 < x_1 < \cdots < x_d \in Z$ that are all mapped to $[1..s]$ by p.

Let $i \in [1..d]$. Since p is nonconstant, it follows from Lemma 5 (with $D = [x_{i-1}..x_i]$ and $R = [0..s-1]$) that there is an integer $x \in [x_{i-1}..x_i]$ such that $p(x) \notin [1..s]$. If $p(x) \leq 0$, then there are two real numbers z_{2i-1} and z_{2i} such that $x_{i-1} < z_{2i-1} < x < z_{2i} < x_i$ and $p(z_{2i-1}) = p(z_{2i}) = \frac{1}{2}$. Otherwise $p(x) \geq s+1$, since p has integer coefficients. In this case, there are two real numbers z_{2i-1} and z_{2i} such that $x_{i-1} < z_{2i-1} < x < z_{2i} < x_i$ and $p(z_{2i-1}) = p(z_{2i}) = s + \frac{1}{2}$.

Furthermore, since p is a nonconstant polynomial,

$$\lim_{x\to\infty} p(x), \lim_{x\to-\infty} p(x) \in \{-\infty, \infty\}.$$

Hence, there exist real numbers $z_0 < x_0$ and $z_{2d+1} > x_d$ such that $p(z_0), p(z_{2d+1}) \in \{\frac{1}{2}, s+\frac{1}{2}\}$. It follows from Lemma 5 (with $D = \{z_0, z_1, \ldots, z_{2d+1}\}$ and $R = \{\frac{1}{2}, s+\frac{1}{2}\}$) that p is constant, which is a contradiction. $\Box$

We are now ready to present our main lemma, which is the same as our main theorem, except that the order of two quantifiers is switched; it states that in any representation, some set is hard.

Lemma 7 *Let $\phi : U^{(n)} \to \mathbf{Z}^s$ be a map that defines a representation of any n-element subset of $U = [0..m-1]$ as a sequence of s integers. Consider any correct RAM program for querying this data structure. For any $\epsilon > 0$, if $n \le s \le m/n^\epsilon$, there is a constant $\delta > 0$, an element $x \in U$, and a set $S \in U^{(n)}$ so that the program uses more than $\delta \log n$ steps on inputs x and $\phi(S)$.*

Proof: Let $l = \lfloor n^{\epsilon/4} \rfloor$. We assume, without loss of generality, that $\epsilon < 1/10$ and $l \ge 10$.

An essential part of our proof technique is to consider a subdomain of U where the elements are spaced widely apart. Then Lemma 6 can be applied to show that indirect reads to locations that are non-constant functions of x can essentially be ignored. This is because the range $[1..s]$, where the data structure is stored, is accessed for relatively few values of x. To achieve the optimal space bound $s \le m/n^\epsilon$, we need wider spacing than can be achieved in a set of size n. Therefore, we restrict our attention to sets of size l. Since $\log l \in \Omega(\log n)$, the lower bound is not affected by more than a constant factor.

The data structure defined by ϕ can also be used for l-element sets from the universe $[0..m-(n-l)-1]$. The representation $\phi'(S)$ of such a set S is simply $\phi(S \cup [m-(n-l)..m-1])$. The original program is correct on all inputs x and $\phi'(S)$, where $x \in [0..m-(n-l)-1]$ and $S \in [0..m-(n-l)-1]^{(l)}$.

Let $g = l \cdot s$ and $U' = \{0, g, 2g, \ldots, (l^2-1)g\}$. Note that $(l^2-1)g \le l^3 s \le n^{3/4\epsilon} s \le m/n^{\epsilon/4} \le m/l \le m-n$, so $U' \subseteq [0..m-(n-l)-1]$.

Let $T = \lfloor \frac{\log l}{10} \rfloor$. To achieve a contradiction, suppose that on all inputs x and $\phi'(S)$, where S is an l-element subset of U', the RAM program reaches an `accept` or `reject` instruction within T steps.

At each step t, we choose a subdomain $U_t \subseteq U'$, a collection V_t of l-element subsets of U', and, for each $S \in V_t$, a small set of bad elements $B_{S,t} \subseteq U_t$ with the following property. For all $S \in V_t$ and all $x \in U_t - B_{S,t}$, the same sequence of instructions are performed and the outcomes of all tests are the same during the first t steps of the computation on inputs x and $\phi'(S)$. For each set $S \in V_t$, it is convenient to represent the values of the accumulator and the pointer at the end of step t by low degree polynomials, $a_{S,t}$ and $p_{S,t}$. The other registers to which values have been written are represented by a small set $Q_{S,t}$ of low degree integer polynomials. Specifically, for each element $x \in U_t - B_{S,t}$, $\{q(x) \mid q \in Q_{S,t}\}$ is the set of registers to which the algorithm writes during the first t steps on inputs x and $\phi'(S)$. In addition, for each polynomial $q \in Q_{S,t}$, the value of these cells is described by a low degree integer polynomial $v_{S,q,t}$ (i.e. $v_{S,q,t}(x)$ is the value in cell $q(x)$ after step t is performed on inputs x and $\phi'(S)$). The above discussion is formalized in the following claim.

Claim For each integer $t \in [0..T]$, there exist

- a subdomain $U_t \subseteq U'$,
- a collection V_t of l-element subsets of U',

- a set of bad elements $B_{S,t} \subseteq U_t$ for each $S \in V_t$
- two polynomials $a_{S,t}$ and $p_{S,t}$ for each set $S \in V_t$.
- a set of polynomials in $Q_{S,t} \subseteq \mathbf{Z}[x]$ for each set $S \in V_t$,
- a polynomial $v_{S,q,t}(x) \in \mathbf{Z}[x]$ for each set $S \in V_t$ and each polynomial $q \in Q_{S,t}$, and
- an integer c_t,

such that, for each set $S \in V_t$,

- $\#(U_t \cap S) \geq l/2^t$,
- $\#V_t \geq \binom{l^2}{l} 2^{-2t}(l^2)^{1-2^t}$,
- $\#B_{S,t} < 2^{2t}$,
- $\text{degree}(a_{S,t}), \text{degree}(p_{S,t}) \leq 2^t$.
- $\#Q_{S,t} \leq t+1$,
- $\text{degree}(q), \text{degree}(v_{S,q,t}) \leq 2^t$ for every polynomial $q \in Q_{S,t}$, and
- $q(x) \neq q'(x)$ for all $x \in U_t - B_{S,t}$ and all distinct polynomials $q, q' \in Q_{S,t}$,

and, for each set $S \in V_t$ and each element $x \in U_t - B_{S,t}$, after t steps on inputs x and $\phi'(S)$,

- the program counter has value c_t,
- the accumulator has value $a_{S,t}(x)$ and the pointer has value $p_{S,t}(x)$.
- register $q(x)$ has value $v_{S,q,t}(x)$ for each polynomial $q \in Q_{S,t}$,
- each register $r \in [1..s] - \{q(x) \mid q \in Q_{S,t}\}$ has value $\phi'(S)_r$, and
- each register $r \notin [1..s] \cup \{q(x) \mid q \in Q_{S,t}\}$ has value 0.

Proof:[of claim] By induction on t.

When $t = 0$, $U_0 = U'$, V_0 is the collection of all l-element subsets of U', $B_{S,0} = \emptyset$ for all $S \in V_0$, and c_0 is the initial value of the program counter, which is the same for all inputs. For each $S \in V_0$, let $a_{S,t} = p_{S,t} = 0$, $Q_{S,t} = \{0\}$, and $v_{S,0,0} = x$. Here 0 denotes the constant polynomial with value 0. In general, we will use k to denote the polynomial with constant value k. Then, $\#V_0 = \binom{\#U'}{l} = \binom{l^2}{l} = \binom{l^2}{l} 2^{-2\cdot 0}(l^2)^{1-2^0}$ and for each set $S \in V_0$ we have $\#(U_0 \cap S) = \#S = l = l/2^0$, $\#B_{S,t} = 0 < 2^{2\cdot 0}$, and $\#Q_{S,t} = 1$.

Now let $t < T$ and assume the claim is true for t. Since the program counter has the same value c_t for all $S \in V_t$ and $x \in U_t - B_{S,t}$, the same instruction is performed at step $t+1$ on all these inputs. We consider a number of different cases, depending on this instruction. To avoid tedious repetition, we will only mention explicitly those things which change. For example, if $V_{t+1} = V_t$, we will not mention this. Also, unless we state otherwise, $c_{t+1} = c_t + 1$.

The `accept` and `reject` instructions do not change the value of the program counter or the contents of any register. Thus everything can remain unchanged.

When the `load` k instruction is performed, the accumulator is assigned the constant value k. We let $a_{S,t+1} = k$.

The `swap` instruction swaps the values of the accumulator and pointer. Thus $p_{S,t+1} = a_{S,t}$ and $a_{S,t+1} = p_{S,t}$.

When `direct write` k is performed, the value $a_{S,t}(x)$ in the accumulator is written to register k. If $Q_{S,t}$ contains the constant polynomial k, then no new polynomial is added to this set, but $v_{S,k,t+1} = a_{S,t}$.

Otherwise, $Q_{S,t+1} = Q_{S,t} \cup \{k\}$, $v_{S,k,t+1} = a_{S,t}$, and $B_{S,t+1} = B_{S,t} \cup \{x \in U_t | \; q(x) = k \text{ for some } q \in Q_{S,t}\}$. Since there are at most $t+1$ polynomials in $Q_{S,t}$, each having degree at most 2^t, it follows that $\#B_{S,t+1} \leq \#B_{S,t} + (t+1)2^t \leq 2^{2(t+1)}$.

When `direct read` k is performed, the value in register k is written to the accumulator. We will deal with each $S \in V_{t+1} = V_t$ one at a time. There are three cases: If $k \in Q_{S,t}$, then $a_{S,t+1} = v_{S,k,t}$. If $k \in [1..s] - Q_{S,t}$, let $a_{S,t+1}$ be the constant polynomial with value $\phi'(S)_k$. If $k \notin Q_{S,t} \cup [1..s]$, then let $a_{S,t+1} = 0$. In the last two cases, $B_{S,t+1} = B_{S,t} \cup \{x \in U_t | \; q(x) = k \text{ for some } q \in Q_{S,t}\}$ and, as above, $\#B_{S,t+1} \leq 2^{2(t+1)}$.

The `add` k, `subtract` k, and `multiply` k instructions are handled similarly to `direct read` k, except that the polynomial assigned to $a_{S,t+1}$ is first added to, subtracted from, or multiplied by the polynomial $a_{S,t}$. For example, for the case of `add` k, if $k \in Q_{S,t}$, let $a_{S,t+1}$ be the polynomial $v_{S,k,t} + a_{S,t}$. The degree can increase only when `multiply` is performed. In this case, since the multiplicands each have degree at most 2^t, the product has degree at most 2^{t+1}.

When `indirect write` is performed, the value in the accumulator is written to the register specified by the contents of the pointer. When $p_{S,t}(x)$ is a constant polynomial with value k, the operation is equivalent to `direct write` k and is handled in the same way. If $p_{S,t} \in Q_{S,t}$, the polynomial $v_{S,q,t+1}$ describing the contents of the cells $\{q(x) \mid x \in U_t - B_{S,t}\}$ is set equal to the polynomial $a_{S,t}$. Otherwise, let $Q_{S,t+1} = Q_{S,t} \cup \{p_{S,t}\}$ (so $\#Q_{S,t+1} \leq t+2$), $v_{S,p_{S,t},t+1} = a_{S,t}$, and

$$B_{S,t+1} = B_{S,t} \cup \{x \in U_t | \; p_{S,t}(x) = q(x) \text{ for some } q \in Q_{S,t}\}.$$

By construction, $p_{S,t}(x) \neq q(x)$ for each $q \in Q_{S,t}$ and $x \in U_{t+1} - B_{S,t+1}$. Each polynomial $q \in Q_{S,t}$ has degree at most 2^t, so $p_{S,t}$ and q intersect in at most 2^t places. Since $\#Q_{S,t} \leq t+1$, it follows that $\#\{x \in U_t | \; p_{S,t}(x) = q(x) \text{ for some } q \in Q_{S,t}\} \leq (t+1)2^t$. Thus $\#B_{S,t+1} < 2^{2t} + (t+1)2^t < 2^{2(t+1)}$.

When `indirect read` k is performed, the value in the register specified by the contents of the pointer is written to the accumulator. If $p_{S,t}(x)$ is a constant polynomial with value k, the operation is equivalent to `direct read` k and is handled in the same way. If $p_{S,t} \in Q_{S,t}$, the polynomial $a_{S,t+1}$ is set equal to the polynomial $v_{S,p_{S,t},t}$. Otherwise,

$$\begin{array}{rcl} B_{S,t+1} & = & B_{S,t} \cup \{x \in U_t | \; p_{S,t}(x) \in [1..s]\} \\ & \cup & \{x \in U_t | \; p_{S,t}(x) = q(x) \text{ for some } q \in Q_{S,t}\}. \end{array}$$

In this case, the value of the `indirect read` is 0 for all $x \in U_t - B_{S,t+1}$, so the polynomial $a_{S,t+1}$ is the constant 0 polynomial. Since $p_{S,t}$ has degree at most $2^t \leq g/s$, it follows from Lemma 6 that $\#\{x \in U_t | \; p_{S,t}(x) \in [1..s]\} \leq 2^t$. So we have $\#B_{S,t+1} < 2^{2t} + 2^t + (t+1)2^t < 2^{2(t+1)}$.

Finally, suppose the instruction executed is `conditional jump` j. For each $S \in V_t$, consider the sequence of test outcomes (either $a_{S,t}(x) > 0$ or $a_{S,t}(x) \leq 0$), for $x \in U_t$ taken in increasing order. Since a polynomial of degree at most 2^t has at most 2^t roots and, hence, can change sign at most 2^t times along any increasing sequence of domain points, there are

$$2 \sum_{i=0}^{2^t} \binom{\#U_t}{i} < 2 \cdot (\#U_t)^{2^t} \leq 2(l^2)^{2^t}$$

different sequences of test outcomes that can occur. Let V' be the largest subcollection of V_t such that all sets $S \in V'$ produce the same sequence of test outcomes. Then $\#V' \geq \#V_t/2(l^2)^{2^t}$.

Let $Y = \{x \in U_t \mid a_{S,t}(x) > 0 \text{ for all } S \in V'\}$ and $N = \{x \in U_t \mid a_{S,t}(x) \leq 0 \text{ for all } S \in V'\}$. Then $Y \cup N = U_t$. For each $S \in V'$, the induction hypothesis says that $\#(U_t \cap S) \geq l/2^t$; so either $\#(Y \cap S) \geq l/2^{t+1}$ or $\#(N \cap S) \geq l/2^{t+1}$. If $\#(Y \cap S) \geq l/2^{t+1}$ for at least half the sets $S \in V'$, then $U_{t+1} = Y$, $V_{t+1} = \{S \in V' \mid \#(Y \cap S) \geq l/2^{t+1}\}$, $\#V_{t+1} \geq \#V'/2$, and $\#(U_{t+1} \cap S) \geq l/2^{t+1}$ for all sets $S \in V_{t+1}$. Also, for each element $x \in U_{t+1} - B_{S,t} \subseteq Y$ and each set $S \in V_{t+1}$, the test succeeds (i.e. $a_{S,t}(x) > 0$), so the jump is performed and the program counter is assigned the value j (i.e. $c_{t+1} = j$). Otherwise, $\#(N \cap S) \geq l/2^{t+1}$ for at least half the sets $S \in V'$. Then $U_{t+1} = N$, $V_{t+1} = \{S \in V' \mid \#(N \cap S) \geq l/2^{t+1}\}$, $\#V_{t+1} \geq \#V'/2$, and $\#(U_{t+1} \cap S) \geq l/2^{t+1}$ for all sets $S \in V_{t+1}$. For each element $x \in U_{t+1}$ and each set $S \in V_{t+1}$, the test fails (i.e. $a_{S,t}(x) \leq 0$), so the jump is not performed and the program counter is incremented (i.e. $c_{t+1} = c_t + 1$). In both cases, $B_{S,t+1} = B_{S,t} \cap U_{t+1}$ (so $\#B_{S,t+1} \leq 2^{2t}$) and

$$\#V_{t+1} \geq \#V'/2 \geq \#V_t/4(l^2)^{2^t} \geq \tbinom{l^2}{l} 2^{-2(t+1)} (l^2)^{1-2^{t+1}}.$$

Since all the desired conditions are true no matter which instruction is performed at step $t+1$, the claim is true for $t+1$ and, by induction, for all integers $t \in [0..T]$. $\Box$

To end the proof, we consider the meaning of the claim for $t = T$. We have a subdomain $U_T \subseteq U'$, a collection V_T of l-element subsets of U', and, for each $S \in V_T$, a set of bad points $B_{S,T} \subseteq U_T$ satisfying the following properties: For each set $S \in V_T$ and each element $x \in U_T - B_{S,T}$, $\#(U_T \cap S) \geq l/2^T$, $\#V_T \geq \binom{l^2}{l} 2^{-2T}(l^2)^{1-2^T}$, $\#B_{S,T} < 2^{2T}$, and, after T steps on inputs x and $\phi'(S)$, the program counter has value c_T.

By assumption, c_T is either an `accept` or `reject` instruction. For any $S \in V_T$,

$$\#((U_T - B_{S,T}) \cap S) = \#((U_T \cap S) - B_{S,T}) \geq l/2^T - 2^{2T} > 0,$$

so there is some $x \in U_T - B_{S,T}$, such that $x \in S$. Thus c_T is an `accept` instruction.

In other words, the program accepts inputs x and $\phi'(S)$ for *all* $S \in V_T$ and $x \in U_T - B_{S,T}$. Since the program is correct, we must have $x \in S$ for all $S \in V_T$ and $x \in U_T - B_{S,T}$. Thus $U_T - B_{S,T} \subseteq S \subseteq U'$ for all $S \in V_T$. Furthermore, $\#U_T \geq l/2^T$ and $|U'| = l^2$. It follows that

$$\#V_T \leq \sum_{i=0}^{2^{2T}} \tbinom{l/2^T}{i} \tbinom{l^2 - l/2^T}{l - l/2^T + i} \leq 2 \tbinom{l/2^T}{2^{2T}} \tbinom{l^2 - l/2^T}{l - l/2^T + 2^{2T}}.$$

Combined with the lower bound $\#V_T \geq \binom{l^2}{l} 2^{-2T}(l^2)^{1-2^T}$, this yields

$$\tbinom{l^2}{l} 2^{-2T} (l^2)^{1-2^T} \leq 2 \tbinom{l/2^T}{2^{2T}} \tbinom{l^2 - l/2^T}{l - l/2^T + 2^{2T}}.$$

By elementary estimates of binomial coefficients, this inequality can be shown to be false for $T = \lfloor \frac{\log l}{10} \rfloor$. Therefore, our assumption that the program always accepts or rejects within T steps must be false. This concludes the proof of Lemma 7. $\Box$ Our main theorem follows easily from Lemma 7.

Theorem 8 *Let $n \leq s \leq m/n^\epsilon$ for some constant $\epsilon > 0$. There is a set $S \in U^{(n)}$, so that for any representation $\phi_S \in \mathbf{Z}^s$ the following is true. Consider* *any RAM program that, on inputs x and ϕ_S, accepts if $x \in S$ and rejects if $x \notin S$. Then, for some $x \in U$, the program uses $\Omega(\log n)$ steps on inputs x and ϕ_S.*

Proof: Given a program A with p instructions, we can encode A as a string of $2p$ integers, representing each instruction by an integer in [1..12] that specifies the instruction type (i.e. 1 for `load`, 2 for `direct read`, etc.) and its argument, if any. Denote this encoding by $\tau(A)$. Cook and Rechow [CR73] show that there exists a universal RAM interpreter with the property that if the interpreter is executed with inputs $\mathbf{z} \in \mathbf{Z}^*$ and $\tau(A)$ (stored in an interleaved fashion), the result is the same as if A was executed on input $\mathbf{z}$. Furthermore, if A uses times T, then the interpreter uses time $O(T)$.

Given any positive constant $\epsilon \leq \frac{1}{2}$, let $\epsilon' = \epsilon/2$. Let n, m, and s be given and assume, without loss of generality, that $n^{\epsilon'} \geq 3$. Suppose that, for all $S \subseteq U$, there exists a representation ϕ_S and a program A_S that runs in time $T < \frac{\log n}{2}$ and accepts inputs x and ϕ_S if and only if $x \in S$. We can assume, without loss of generality, that A_S contains at most $2^T < n^{1/2}$ instructions. For any S, define $\phi(S)$ to consist of $\tau(A_S)$ and ϕ_S interleaved. The size of this data structure is at most $s' = 2\max(2n^{1/2}, s) < m/n^{\epsilon'}$. When the interpreter is run on inputs $x \in U$ and $\phi(S)$, it accepts if and only if $x \in S$. Furthermore, it accepts or rejects within $O(T)$ steps. From Lemma 7, it follows that $T \in \Omega(\log n)$. $\square$

Acknowledgments

We are indebted to Dany Breslauer for suggesting how to remove the restriction on the number of registers used by the query algorithm that appeared in a previous version of our theorem.

The first author was supported in part by Natural Science and Engineering Research Council of Canada grant A-9176 and the Information Technology Research Centre of Ontario. The second author was supported by a postdoctoral fellowship from the Danish Natural Science Research Council through BRICS, Centre of the Danish National Research Foundation.

References

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

[BG91] A.M. Ben-Amram and Z. Galil, Lower bounds for data structure problems on RAMs. In *Proc. 32th IEEE Symposium on Foundations of Computer Science*, 1991, pages 622–631.

[Ben86] J.L. Bentley, *Programming Pearls*. Addison-Wesley, Reading, MA, 1986.

[BM94] A. Brodnik and J. Ian Munro, Membership in constant time and minimum space. In *Proc. 2nd European Symposium on Algorithms* 1994, pages 72–81.

[CR73] S.A. Cook and R.A. Reckhow, Time bounded random access machines. *JCSS*, vol. 7, 1973, pages 354–375.

[FN93] A. Fiat and M. Naor, Implicit $O(1)$ probe search. *SIAM J. Computing* vol. 22, 1993, pages 1–10.

[FNSS92] A. Fiat, M. Naor, J.P. Schmidt, and A. Seigel, Nonoblivious hashing. *J. ACM* vol. 39, 1992, pages 764–782.

[FKS84] M.L. Fredman, J. Komlòs, and E. Szemerédi, Storing a sparse table with O(1) worst case access time. *J. ACM*, vol. 31, 1984, pages 538–544.

[FS89] M.L. Fredman and M.E. Saks, The cell probe complexity of dynamic data structures. In *Proc. 21st Ann. ACM Symp. on Theory of Computing*, 1989, pages 345–354.

[Knu73] D.E. Knuth, *The Art of Computer Programming.* Addison-Wesley, Reading, MA, 1973.

[Maa88] W. Maass, On the use of inaccessible numbers and order indiscernibles in lower bound arguments for random access machines. *J. Symbolic Logic*, vol. 53, 1988, pages 1098–1109.

[MNSW95] P.B. Miltersen, N. Nisan, S. Safra, and A. Wigderson, On data structures and asymmetric communication complexity. In *Proc. 27th ACM Symposium on Theory of Computing*, 1995, to appear.

[PS82] W. Paul and J. Simon, Decision trees and random access machines. In *Logic and Algorithmic*, monograph no. 30 de l'enseignement mathématique, Univeristé de Genève, 1982.

[ST85] D.D. Sleator and R.E. Tarjan, Self-Adjusting Binary Search Trees. *J. ACM*, vol. 32, 1985, pages 652-686.

[TY79] R.E. Tarjan and A.C. Yao, Storing a sparse table. *C. ACM*, vol. 22, 1979, pages 606–611.

[Yao81] A.C. Yao, Should tables be sorted? *J. ACM*, vol. 28, 1981, pages 615–628.

Recent Publications in the BRICS Report Series