

Randomized Computations on Large Data Sets: Tight Lower Bounds

Martin Grohe

André Hernich

Nicole Schweikardt

Institut für Informatik, Humboldt-Universität, Berlin, Germany
{grohe|hernich|schweika}@informatik.hu-berlin.de

ABSTRACT

We study the randomized version of a computation model (introduced in [9, 10]) that restricts random access to external memory and internal memory space. Essentially, this model can be viewed as a powerful version of a data stream model that puts no cost on sequential scans of external memory (as other models for data streams) and, in addition, (like other external memory models, but unlike streaming models), admits several large external memory devices that can be read and written to in parallel.

We obtain tight lower bounds for the decision problems set equality, multiset equality, and checksort. More precisely, we show that any randomized one-sided-error bounded Monte Carlo algorithm for these problems must perform $\Omega(\log N)$ random accesses to external memory devices, provided that the internal memory size is at most $O(\sqrt[3]{N}/\log N)$, where N denotes the size of the input data.

From the lower bound on the set equality problem we can infer lower bounds on the worst case data complexity of query evaluation for the languages XQuery, XPath, and relational algebra on streaming data. More precisely, we show that there exist queries in XQuery, XPath, and relational algebra, such that any (randomized) Las Vegas algorithm that evaluates these queries must perform $\Omega(\log N)$ random accesses to external memory devices, provided that the internal memory size is at most $O(\sqrt[3]{N}/\log N)$.

Categories and Subject Descriptors

F.1.3 [Computation by Abstract Devices]: Complexity Measures and Classes; F.1.1 [Computation by Abstract Devices]: Models of Computation

General Terms

Theory, Languages

Keywords

complexity, data streams/real-time data, query processing/query optimization, semi-structured data, XML

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODS'06 June 26–28, 2006, Chicago, Illinois, USA.

Copyright 2006 ACM 1-59593-318-2/06/0003 ...\$5.00.

1. INTRODUCTION

Today's hardware technology provides a hierarchy of storage media from tapes and disks at the bottom through main memory and (even on-CPU) memory caches at the top. Storage media from different levels of this memory hierarchy considerably differ in price, storage size, and access time. Currently, the most pronounced performance and price (and consequently also size) gap is between main memory and the next-lower level in the memory hierarchy, usually magnetic disks which have to rely on comparably slow, mechanical, physically moving parts. One often refers to the upper layers above this gap by *internal memory* and the lower layers of the memory hierarchy by *external memory*. The technological reality is such that the time for accessing a given bit of information in external memory is five to six orders of magnitude larger than the time required to access a bit in internal memory. Apart from this, concerning external memory, *random accesses* (which involve moving the disk head to a particular location) are significantly more expensive than *sequential scans*.

Modern software and database technology uses clever heuristics to minimize the number of accesses to external memory and to prefer *streaming* over *random accesses* to external memory. There has also been a wealth of research on the design of so-called *external memory algorithms* (cf., e.g. [16, 18, 13]). The classes considered in *computational complexity theory*, however, usually do not take into account the existence of different storage media. In [9, 10], we introduced a formal model for such a scenario. The two most significant cost measures in our setting are the number of random accesses to external memory and the size of the internal memory. Our model is based on a standard multi-tape Turing machine. Some of the tapes of the machine, among them the input tape, represent the external memory. They are unrestricted in size, but access to these tapes is restricted by allowing only a certain number $r(N)$ (where N denotes the input size) of reversals of the head directions. This may be seen as a way of (a) restricting the number of sequential scans and (b) restricting random access to these tapes, because each random access can be simulated by moving the head to the desired position on a tape, which involves at most two head reversals. The remaining tapes of the Turing machine represent the internal memory. Access to these internal memory tapes (i.e., the number of head reversals) is unlimited, but their size is bounded by a parameter $s(N)$. We let $ST(r(N), s(N), O(1))$ denote the class of all problems that can be solved on such an $(r(N), s(N), O(1))$ -bounded Turing machine, i.e., a Turing machine with an arbitrary number of external memory tapes which, on inputs of size N , performs less than $r(N)$ head reversals on the external memory tapes, and uses at most space $s(N)$ on the internal memory tapes.

The astute reader who wonders if it is realistic to assume that the external memory tapes can be read in *both* directions (which disks

cannot so easily) and that a sequential scan of an entire external memory tape accounts for only one head reversal (and thus seems unrealistically cheap) be reminded that this paper’s main goal is not to design efficient external memory algorithms but, instead, to prove *lower* bounds. Thus, considering a rather powerful computation model makes our lower bound results only stronger.

In the present paper, we prove lower bounds for *randomized* computations (i.e., computations where in each step a coin may be tossed to determine the next configuration) in a scenario with several storage media. To this end, we introduce the complexity class $\text{RST}(r(N), s(N), O(1))$, which consists of all decision problems that can be solved by an $(r(N), s(N), O(1))$ -bounded randomized Turing machine with one-sided bounded error, where no false positive answers are allowed and the probability of false negative answers is at most 0.5 (in the literature, such randomized algorithms are often called *one-sided-error Monte Carlo algorithms*, cf. [12]). To also deal with computation problems where an output (other than just a yes/no answer) has to be generated, we write $\text{LasVegas-RST}(r(N), s(N), O(1))$ to denote the class of all functions f for which there exists an $(r(N), s(N), O(1))$ -bounded randomized Turing machine that, for every input word w , (a) always produces either the correct output $f(w)$ on one of its external memory tapes or gives the answer “*I don’t know*” and (b) gives the answer “*I don’t know*” with probability at most 0.5 (in the literature, such randomized algorithms are sometimes called *Las Vegas algorithms*, cf. [12]).

Contributions: Our first main result is a lower bound for three natural decision problems: The *set equality problem* and the *multiset equality problem* ask whether two given (multi)sets of strings are equal, and the *checksort problem* asks, given two sequences of strings, whether the second is a sorted version of the first. We show (Theorem 6) that neither problem is contained in $\text{RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$. This lower bound turns out to be *tight* in the following senses:

- If the number of sequential scans (i.e., head reversals) increases from $o(\log N)$ to $O(\log N)$, then each of the three problems can be solved with only constant internal memory and without using randomization. In other words (see Corollary 7), the (multi)set equality problem and the checksort problem belong to $\text{ST}(O(\log N), O(1), O(1))$.
- When using randomization with the complementary one-sided error model, i.e., machines where no false negative answers are allowed and the probability of false positive answers is at most 0.5, then the *multiset equality* problem can be solved with just two sequential scans of the input (and without ever writing to external memory), and internal memory of size $O(\log N)$. In other words (Theorem 8(a)), the multiset equality problem belongs to $\text{co-RST}(2, O(\log N), 1)$.
- When using nondeterministic machines, then (multi)set equality and checksort can be solved with three sequential scans on two external memory tapes and internal memory of size $O(\log N)$. In other words (Theorem 8(b)), the (multi)set equality problem and the checksort problem belong to $\text{NST}(3, O(\log N), 2)$.

As a consequence, we obtain a separation between the deterministic, the randomized, and the nondeterministic $\text{ST}(\dots)$ classes (Corollary 9).

Our lower bound for the checksort problem, in particular, implies that the *sorting problem* (i.e., the problem of sorting a sequence of input strings) does not belong to the complexity class

$\text{LasVegas-RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$ and thus generalizes the main result of [10] to randomized computations.

Our lower bound for the set equality problem leads to the following lower bounds on the worst case data complexity of database query evaluation problems in a streaming context:

- There is an $XQuery$ query Q such that the problem of evaluating Q on an input XML document stream of length N does *not* belong to the class $\text{LasVegas-RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$ (Theorem 12).

Speaking informally, this means that, no matter how many external memory devices (of arbitrarily large size) are available, as long as the internal memory is of size at most $O(\frac{\sqrt[4]{N}}{\log N})$, every randomized algorithm that produces the correct query result with probability at least 0.5 will perform $\Omega(\log N)$ random accesses to external memory. We obtain analogous results for *relational algebra* queries and for the node-selecting XML query language $XPath$:

- There is a *relational algebra* query Q such that the problem of evaluating Q on a stream consisting of the tuples of the input database relations does *not* belong to the complexity class $\text{LasVegas-RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$, where N denotes the total size of the input database relations. Furthermore, this bound is tight with respect to the number of random accesses to external memory, as the data complexity of every relational algebra query belongs to $\text{ST}(O(\log N), O(1), O(1))$ (Theorem 11).
- There is an $XPath$ query Q such that the problem of filtering an input XML document stream with Q (i.e., checking whether at least one node of the document matches the query) does *not* belong to the class $\text{co-RST}(o(\log N), O(\frac{\sqrt[4]{N}}{\log N}), O(1))$ (Theorem 13).

This means that there is an $XPath$ query Q such that, no matter how many external memory devices (of arbitrarily large size) are available, as long as the internal memory is of size at most $O(\frac{\sqrt[4]{N}}{\log N})$, every randomized algorithm which accepts every input document that matches Q , and which rejects documents not matching Q with probability ≥ 0.5 , will perform $\Omega(\log N)$ random accesses to external memory.

Related Work: Obviously, our model is related to the *bounded reversal Turing machines*, which have been studied in classical complexity theory (see, e.g., [19, 7]). However, in bounded reversal Turing machines, the number of head reversals is limited on *all* tapes, whereas in our model there is no such restriction on the internal memory tapes. This makes our model considerably stronger, considering that in our lower bound results we allow internal memory size that is polynomially related to the input size. Furthermore, to our best knowledge, all lower bound proofs previously known for reversal complexity classes on multi-tape Turing machines go back to the space hierarchy theorem (cf., e.g., [17, 7]) and thus rely on diagonalization arguments, and apply only to classes with $\omega(\log N)$ head reversals. In particular, these lower bounds do not include the checksort problem and the (multi)set equality problem, as these problems can be solved with $O(\log N)$ head reversals.

In the classical *parallel disk model* for external memory algorithms (see, e.g., [18, 13, 16]), the cost measure is simply the number of bits read from external memory divided by the page size. Several refinements of this model have been proposed to include a distinction between *random access* and *sequential scans* of the external memory, among them Arge and Bro Miltersen’s *external*

memory Turing machines [3]. We note that their notion of external memory Turing machines significantly differs from ours, as their machines only have a single external memory tape and process inputs that consist of a *constant* number m of input strings. Strong lower bound results (in particular, for different versions of the *sorting problem*) are known for the parallel disk model (see [18] for an overview) as well as for Arge and Bro Miltersen’s external memory Turing machines [3]. However, to the best of our knowledge, all these lower bound proofs heavily rely on the assumption that the input data items (e.g., the strings that are to be sorted) are *indivisible* and that at any point in time, the external memory consists, in some sense, of a permutation of the input items. We emphasize that the present paper’s lower bound proofs do not rely on such an indivisibility assumption.

Strong lower bounds for a number of problems are known in the context of *data streams* and for models which permit a small number of sequential scans of the input data, but no auxiliary external memory (that is, the version of our model with no extra external memory tapes apart from the input tape) [15, 2, 11, 4, 16, 5, 6, 1, 9]. All these lower bounds are obtained by communication complexity. Note that in the presence of at least two external memory tapes, communication between remote parts of memory is possible by simply copying data from one tape to another and then re-reading both tapes in parallel. These communication abilities of our model spoil any attempt to prove lower bounds via communication complexity, which is the tool of choice both for computation models permitting few scans but no auxiliary external memory, and for 1-tape Turing machines.

The deterministic $ST(\dots)$ -classes were introduced in [10, 9]. In [9] we studied those classes where only a *single* external memory tape is available and used methods from communication complexity to obtain lower bounds for these classes. The main result of [10] was a lower bound for the *sorting problem* concerning the deterministic $ST(\dots)$ -classes with an *arbitrary* number of external memory tapes. An important tool for proving this bound was to introduce deterministic *list machines* as an intermediate machine model. An overview of the methods used and the results obtained in [9, 10] was given in [8]. The present paper builds on [10], as it considers $ST(\dots)$ -classes with an *arbitrary* number of external memory tapes and it uses *list machines* as a key tool for proving lower bound results. However, the results presented here go significantly beyond those obtained in [10]. Here we obtain lower bounds for *decision* problems in the *randomized* versions of the model. The main result of [10] is that the sorting problem does not belong to $ST(o(\log N), O(\frac{\sqrt{N}}{\log N}), O(1))$, and the proof given there heavily relies on the fact that the machines are *deterministic* and the *output* of the sorting problem cannot be generated within the given resource bounds. In contrast to the present paper’s approach, the proof method of [10] neither works for decision problems, i.e. problems where no output is generated, nor for randomized computations. Finally, let us remark that the main result of [10] can be obtained as an immediate corollary of the present paper’s lower bound for the *checksort* problem.

Organization: After introducing the deterministic, the nondeterministic, and the randomized $ST(\dots)$ classes in Section 2, we formally state our main lower bounds for decision problems in Section 3. In Section 4 we use these results to derive lower bounds on the data complexity of query evaluation for the languages $XQuery$, $XPath$, and *relational algebra*. The subsequent sections are devoted to the proof of the lower bound on the decision problems (*multiset equality* and *checksort*): In Section 5, 6, and 7 we introduce randomized *list machines*, show that randomized Turing machines can be

simulated by randomized list machines, and prove that randomized list machines can neither solve the (*multi*)*set equality problem* nor the *checksort problem*. Afterwards, in Section 8 we transfer these results from list machines to Turing machines. We close with a few concluding remarks and open problems in Section 9.

Due to space limitations, many technical details of our proofs had to be deferred to the full version of this paper, available on the authors’ websites.

2. COMPLEXITY CLASSES

We write \mathbb{N} to denote the set of natural numbers (that is, nonnegative integers).

As our basic model of computation, we use standard multi-tape nondeterministic Turing machines (NTMs, for short); cf., e.g., [17]. The Turing machines we consider will have $t + u$ tapes. We call the first t tapes *external memory tapes* (and think of them as representing t disks). We call the other u tapes *internal memory tapes*. The first tape is always viewed as the input tape.

Without loss of generality we assume that our Turing machines are normalized in such a way that in each step at most one of its heads moves to the left or to the right.

Let T be an NTM and ρ a finite run of T . Let $i \geq 1$ be the number of a tape. We use $\text{rev}(\rho, i)$ to denote the number of times the head on tape i changes its direction in the run ρ . Furthermore, we let $\text{space}(\rho, i)$ be the number of cells of tape i that are used by ρ .

Definition 1 ((r, s, t)-bounded TM). Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ and $t \in \mathbb{N}$. A (nondeterministic) Turing machine T is (r, s, t)-bounded, if every run ρ of T on an input of length N (for arbitrary $N \in \mathbb{N}$) satisfies the following conditions: (1) ρ is finite, (2) $1 + \sum_{i=1}^t \text{rev}(\rho, i) \leq r(N)$, and¹ (3) $\sum_{i=t+1}^{t+u} \text{space}(\rho, i) \leq s(N)$, where $t + u$ is the total number of tapes of T . \dashv

Definition 2 ($ST(\dots)$ and $NST(\dots)$ classes).

Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ and $t \in \mathbb{N}$. A decision problem belongs to the class $ST(r, s, t)$ (resp., $NST(r, s, t)$), if it can be decided by a deterministic (resp., nondeterministic) (r, s, t)-bounded Turing machine. \dashv

Note that we put no restriction on the running time or the space used on the first t tapes of an (r, s, t)-bounded Turing machine. The following lemma shows that these parameters cannot get too large.

Lemma 3 ([10]). Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ and $t \in \mathbb{N}$, and let T be an (r, s, t)-bounded NTM. Then for every run $\rho = (\rho_1, \dots, \rho_\ell)$ of T on an input of size N we have $\ell \leq N \cdot 2^{O(r(N) \cdot (t+s(N)))}$ and thus $\sum_{i=1}^t \text{space}(\rho, i) \leq N \cdot 2^{O(r(N) \cdot (t+s(N)))}$. \dashv

In [10], the lemma has only been stated and proved for deterministic Turing machines, but it is obvious that the same proof also applies to nondeterministic machines (to see this, note that, by definition, every run of an (r, s, t)-bounded Turing machine is finite).

In analogy to the definition of *randomized* complexity classes such as the class RP of randomized polynomial time (cf., e.g., [17]), we consider the randomized versions $RST(\dots)$ and $Las Vegas-RST(\dots)$ of the $ST(\dots)$ and $NST(\dots)$ classes. The following definition of randomized Turing machines formalizes the intuition that in each step, a coin can be tossed to determine which particular successor configuration is chosen in this step. For a configuration γ of an NTM T , we write $\text{Next}_T(\gamma)$ to denote the set of all configurations γ' that can be reached from γ in a single step. Each

¹It is convenient for technical reasons to add 1 to the number $\sum_{i=1}^t \text{rev}(\rho, i)$ of changes of the head direction here. As defined here, $r(N)$ thus bounds the number of sequential scans of the external memory tapes rather than the number of changes of head directions.

such configuration $\gamma' \in \text{Next}_T(\gamma)$ is chosen with uniform probability, i.e., $\Pr(\gamma \rightarrow_T \gamma') = 1/|\text{Next}_T(\gamma)|$. For a run $\rho = (\rho_1, \dots, \rho_\ell)$, the probability $\Pr(\rho)$ that T performs run ρ is the product of the probabilities $\Pr(\rho_i \rightarrow_T \rho_{i+1})$, for all $i < \ell$. For an input word w , the probability that T accepts w (resp., that T outputs w') is defined as the sum of $\Pr(\rho)$ for all accepting runs ρ of T on input w (resp., of all runs of T on w that output w'). We say that a decision problem L is solved by a $(\frac{1}{2}, 0)$ -RTM if, and only if, there is an NTM T such that every run of T has finite length, and the following is true for all input instances w : If $w \in L$, then $\Pr(T \text{ accepts } w) \geq 1/2$; if $w \notin L$, then $\Pr(T \text{ accepts } w) = 0$. Similarly, we say that a function $f: \Sigma^* \rightarrow \Sigma^*$ is computed by a *Las Vegas*-RTM if, and only if, there is an NTM T such that every run of T on every input instance w has finite length and outputs either $f(w)$ or “*I don't know*”, and $\Pr(T \text{ outputs } f(w)) \geq 1/2$.

Definition 4 (RST(\dots) and *Las Vegas*-RST(\dots)).

Let $r, s: \mathbb{N} \rightarrow \mathbb{N}$ and $t \in \mathbb{N}$.

- (a) A decision problem L belongs to the class $\text{RST}(r, s, t)$, if it can be solved by a $(\frac{1}{2}, 0)$ -RTM that is (r, s, t) -bounded.
- (b) A function $f: \Sigma^* \rightarrow \Sigma^*$ belongs to *Las Vegas*-RST(r, s, t), if it can be solved by a *Las Vegas*-RTM that is (r, s, t) -bounded. \dashv

As a straightforward observation one obtains:

Proposition 5. For all $r, s: \mathbb{N} \rightarrow \mathbb{N}$ and $t \in \mathbb{N}$, $\text{ST}(r, s, t) \subseteq \text{RST}(r, s, t) \subseteq \text{NST}(r, s, t)$. \dashv

For classes R and S of functions we let $\text{ST}(R, S, t) := \bigcup_{r \in R, s \in S} \text{ST}(r, s, t)$ and $\text{ST}(R, S, O(1)) := \bigcup_{r \in R, s \in S} \text{ST}(r, s, t)$. Analogous notations are used for the $\text{NST}(\dots)$, $\text{RST}(\dots)$, and *Las Vegas*-RST(\dots) classes, too.

As usual, for every (complexity) class C of decision problems, $\text{co-}C$ denotes the class of all decision problems whose *complements* belong to C . Note that the $\text{RST}(\dots)$ -classes consist of decision problems that can be solved by randomized algorithms that allow a moderate number of false negatives, but no false positives. In contrast to this, the $\text{co-RST}(\dots)$ -classes consist of problems that can be solved by randomized algorithms that allow a moderate number of false positives, but no false negatives.

From Lemma 3, one immediately obtains for all functions r, s with $r(N) \cdot s(N) \in O(\log N)$ that $\text{ST}(r, s, O(1)) \subseteq \text{PTIME}$, $\text{RST}(r, s, O(1)) \subseteq \text{RP}$, and $\text{NST}(r, s, O(1)) \subseteq \text{NP}$ (where PTIME , RP , and NP denote the class of problems solvable in polynomial time on deterministic, randomized, and nondeterministic Turing machines, respectively).

3. LOWER BOUNDS FOR DECISION PROBLEMS

Our first main result is a lower bound for the *(multi)set equality problem* as well as for the *checksort problem*. The *(multi)set equality problem* asks if two given (multi)sets of strings are the same. The *checksort problem* asks for two input lists of strings whether the second list is the lexicographically sorted version of the first list. We encode inputs as strings over the alphabet $\{0, 1, \#\}$. Formally, the *(multi)set equality* and the *checksort problem* are defined as follows: The input instances of each of the three problems are

Instance: $v_1\#\dots\#v_m\#v'_1\#\dots\#v'_m\#$,
where $m \geq 0$, and $v_i, v'_i \in \{0, 1\}^*$ (for all $i \leq m$)

and the task is to decide the following:

SET-EQUALITY problem:

Decide if $\{v_1, \dots, v_m\} = \{v'_1, \dots, v'_m\}$.

MULTISET-EQUALITY problem:

Decide if the multisets $\{v_1, \dots, v_m\}$ and $\{v'_1, \dots, v'_m\}$ are equal (i.e., they contain the same elements with the same multiplicities).

CHECK-SORT problem:

Decide if v'_1, \dots, v'_m is the lexicographically sorted (in ascending order) version of v_1, \dots, v_m .

For an instance $v_1\#\dots\#v_m\#v'_1\#\dots\#v'_m\#$ of the above problems, we usually let $N = 2m + \sum_{i=1}^m (|v_i| + |v'_i|)$ denote the size of the input. Furthermore, in our proofs we will only consider instances where all the v_i and v'_i have the same length n , so that $N = 2m \cdot (n + 1)$.

The present paper's technically most involved result is the following lower bound:

Theorem 6. Let $r, s: \mathbb{N} \rightarrow \mathbb{N}$ such that $r(N) \in o(\log N)$ and $s(N) \in o(\sqrt[3]{N}/r(N))$. Then, none of the problems CHECK-SORT, SET-EQUALITY, MULTISET-EQUALITY belongs to the class $\text{RST}(r(N), s(N), O(1))$. \dashv

Sections 5–8 are devoted to the proof of Theorem 6. The proof uses an intermediate computation model called *list machines* and proceeds by (1) showing that randomized Turing machine computations can be simulated by randomized list machines that have the same acceptance probabilities as the given Turing machines and (2) proving a lower bound for (MULTI)SET-EQUALITY and CHECK-SORT on randomized list machines.

By applying the reduction used in [10, Theorem 9], we obtain that the lower bound of Theorem 6 also applies for the “SHORT” versions of (MULTI)SET-EQUALITY and CHECK-SORT, i.e., the restrictions of these problems to inputs of the form $v_1\#\dots\#v_m\#v'_1\#\dots\#v'_m\#$, where each v_i and v'_i is a 0-1-string of length at most $c \cdot \log m$, and c is an arbitrary constant ≥ 2 . By using the standard *merge sort* algorithm, one easily obtains that the “SHORT” versions of (MULTI)SET-EQUALITY and CHECK-SORT belong to $\text{ST}(O(\log N), O(\log N), 3)$. Moreover, in [7, Lemma 7] it has been shown that the (general) sorting problem can be solved by an $(O(\log N), O(1), 2)$ -bounded deterministic Turing machine. As an immediate consequence, we obtain:

Corollary 7. SET-EQUALITY, MULTISET-EQUALITY, CHECK-SORT, and their “SHORT” versions, are in $\text{ST}(O(\log N), O(1), 2)$, but not in $\text{RST}(o(\log N), O(\sqrt[3]{N}/\log N), O(1))$. \dashv

A detailed proof can be found in the full version of this paper.

As a further result, we show that

Theorem 8. (a) MULTISET-EQUALITY belongs to $\text{co-RST}(2, O(\log N), 1) \subseteq \text{co-NST}(2, O(\log N), 1)$.

(b) Each of the problems MULTISET-EQUALITY, CHECK-SORT, SET-EQUALITY belongs to $\text{NST}(3, O(\log N), 2)$. \dashv

Proof: (a): We apply fairly standard *fingerprinting techniques* and show how to implement them on a $(2, O(\log N), 1)$ -bounded randomized Turing machine. Consider an instance $v_1\#\dots\#v_m\#v'_1\#\dots\#v'_m\#$ of the MULTISET-EQUALITY problem. For simplicity, let us assume that all the v_i and v'_j have the same length n . Thus the input size N is $2 \cdot m \cdot (n + 1)$. We view the v_i and v'_i as integers in $\{0, \dots, 2^n - 1\}$ represented in binary.

We use the following algorithm to decide whether the multisets $\{v_1, \dots, v_m\}$ and $\{v'_1, \dots, v'_m\}$ are equal:

- (1) During a first sequential scan of the input, determine the input parameters n , m , and N .
- (2) Choose a prime $p_1 \leq k := m^3 \cdot n \cdot \log(m^3 \cdot n)$ uniformly at random.

- (3) Choose an arbitrary prime p_2 such that $3k < p_2 \leq 6k$. Such a prime exists by Bertrand's postulate.
- (4) Choose $x \in \{1, \dots, p_2 - 1\}$ uniformly at random.
- (5) For $1 \leq i \leq m$, let $e_i = (v_i \bmod p_1)$ and $e'_i = (v'_i \bmod p_1)$. If

$$\sum_{i=1}^m x^{e_i} \equiv \sum_{i=1}^m x^{e'_i} \pmod{p_2} \quad (1)$$

then accept, else reject.

Let us first argue that the algorithm is correct (for sufficiently large m, n): Clearly, if the multisets $\{v_1, \dots, v_m\}$ and $\{v'_1, \dots, v'_m\}$ are equal then the algorithm accepts. On the other hand, if they are distinct, the probability that the multisets $\{e_1, \dots, e_m\}$ and $\{e'_1, \dots, e'_m\}$ are equal is $O(1/m)$. This is due to the following.

CLAIM 1. *Let $n, m \in \mathbb{N}$, $k = m^3 \cdot n \cdot \log(m^3 \cdot n)$, and $0 \leq v_1, \dots, v_m, v'_1, \dots, v'_m < 2^n$. Then for a prime $p \leq k$ chosen uniformly at random, $\Pr(\exists i, j \leq m$ with $v_i \neq v'_j$ and $v_i \equiv v'_j \pmod{p}) \leq O(1/m)$.*

Proof: We use the following well-known result (see, for example, Theorem 7.5 of [14]): Let $n, \ell \in \mathbb{N}$, $k = \ell \cdot n \cdot \log(\ell \cdot n)$, and $0 < x < 2^n$. Then for a prime $p \leq k$ chosen uniformly at random,

$$\Pr(x \equiv 0 \pmod{p}) \leq O\left(\frac{1}{\ell}\right).$$

The claim then follows if we apply this result with $\ell = m^3$ simultaneously to the at most m^2 numbers $x = v_i - v'_j$ with $v_i \neq v'_j$. \square

To proceed with the proof of Theorem 8(a), suppose that the two multisets are distinct. Then the polynomial

$$q(X) = \sum_{i=1}^m X^{e_i} - \sum_{i=1}^m X^{e'_i}$$

is nonzero. Note that all coefficients and the degree of $q(X)$ are at most $k < p_2$. We view $q(X)$ as a polynomial over the field \mathbb{F}_{p_2} . As a nonzero polynomial of degree at most p_1 , it has at most p_1 zeroes. Thus the probability that $q(x) = 0$ for the randomly chosen $x \in \{1, \dots, p_2 - 1\}$ is at most $p_1 / (p_2 - 1) \leq 1/3$. Therefore, if the multisets $\{e_1, \dots, e_m\}$ and $\{e'_1, \dots, e'_m\}$ are distinct, the algorithm accepts with probability at most $1/3$, and the overall acceptance probability is at most

$$\frac{1}{3} + O\left(\frac{1}{m}\right) \leq \frac{1}{2}$$

for sufficiently large m . This proves the correctness of the algorithm.

Let us now explain how to implement the algorithm on a $(2, O(\log N), 1)$ -bounded randomized Turing machine. Note that the binary representations of the primes p_1 and p_2 have length $O(\log N)$. The standard arithmetical operations can be carried out in linear space on a Turing machine. Thus with numbers of length $O(\log N)$, we can carry out the necessary arithmetic on the internal memory tapes of our $(2, O(\log N), 1)$ -bounded Turing machine.

To choose a random prime p_1 in step (2), we simply choose a random number $\leq k$ and then test if it is prime, which is easy in linear space. If the number is not prime, we repeat the procedure, and if we do this sufficiently often, we can find a random prime with high probability. Steps (3) and (4) can easily be carried out in internal memory. To compute the number e_i in step (5), we proceed as follows: Suppose the binary representation of v_i is $v_{i,(n-1)} \dots v_{i,0}$, where $v_{i,0}$ is the least significant bit. Observe that

$$e_i = \left(\left(\sum_{j=0}^{n-1} 2^j \cdot v_{i,j} \right) \bmod p_1 \right).$$

We can evaluate this sum sequentially by taking all terms modulo p_1 ; this way we only have to store numbers smaller than p_1 . This requires one sequential scan of v_i and no head reversals.

To evaluate the polynomial $\sum_{i=1}^m x^{e_i}$ modulo p_2 , we proceed as follows: Let $t_i = (x^{e_i} \bmod p_1)$ and $s_i = ((\sum_{j=1}^i t_j) \bmod p_1)$. Again we can compute the sum sequentially by computing e_i , t_i , and $s_i = ((s_{i-1} + t_i) \bmod p_1)$ for $i = 1, \dots, m$. We can evaluate $\sum_{i=1}^m x^{e'_i}$ analogously and then test if (1) holds. This completes the proof of part (a) of Theorem 8.

(b): Let w be an input of length N , $w := v_1 \# v_2 \# \dots \# v_m \# v'_1 \# v'_2 \# \dots \# v'_m \#$. Note that the multisets $\{v_1, \dots, v_m\}$ and $\{v'_1, \dots, v'_m\}$ are equal if and only if there is a permutation π of $\{1, \dots, m\}$ such that for all $i \in \{1, \dots, m\}$, $v_i = v'_{\pi(i)}$. The idea is to "guess" such a permutation π (suitably encoded as a string over $\{0, 1, \#\}$), to write sufficiently many copies of the string $u := \pi \# w$ onto the first tape, and finally solve the problem by comparing v_i and $v'_{\pi(i)}$ bitwise, where in each step we use the next copy of u .

A $(3, O(\log N), 2)$ -bounded nondeterministic Turing machine M can do this as follows. In a forward scan, it nondeterministically writes a sequence u_1, u_2, \dots, u_ℓ of $\ell := m + N \cdot m$ many strings on its first and on its second tape, where

$$u_i := \pi_{i,1} \# \dots \# \pi_{i,m} \# v_{i,1} \# \dots \# v_{i,m} \# v'_{i,1} \# \dots \# v'_{i,m} \#$$

for binary numbers $\pi_{i,j}$ from $\{1, \dots, m\}$, and bit strings $v_{i,j}$ and $v'_{i,j}$ of length at most N . While writing the first $N \cdot m$ strings, it ensures that for every $i \in \{1, \dots, N \cdot m\}$, either $v_{i,[i/N]}$ and $v'_{i,\pi_{i,[i/N]}}$ coincide on bit $((i-1) \bmod N) + 1$, or that both strings have no such bit at all. While writing the last m strings, it ensures that for all $i \in \{1, \dots, m\}$ and $j \in \{i+1, \dots, m\}$, $\pi_{N-m+i,i} \neq \pi_{N-m+i,j}$. Finally, M checks in a backward scan of both external memory tapes that $u_i = u_{i-1}$ for all $i \in \{2, \dots, \ell\}$, and that $v_{1,j} = v_j$ and $v'_{1,j} = v'_j$ for all $j \in \{1, \dots, m\}$.

The SET-EQUALITY problem can be solved in a similar way.

Deciding CHECK-SORT is very similar: the machine additionally has to check that v'_i is smaller than or equal to v'_j for all $j \in \{i+1, \dots, m\}$. This can be done, e.g., by writing $N \cdot \sum_{i=1}^{m-1} i$ additional copies of u , and by comparing v'_i and v'_j bitwise on these strings for each i and $j \in \{i+1, \dots, m\}$. \square

Theorems 6 and 8, in particular, immediately lead to the following separations between the deterministic, randomized, and nondeterministic $\text{ST}(\dots)$ classes:

Corollary 9. *Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ with $r(N) \in o(\log N)$ and $s(N) \in o(\sqrt[3]{N}/r(N)) \cap \Omega(\log N)$. Then,*

- (a) $\text{RST}(O(r), O(s), O(1)) \neq \text{co-RST}(O(r), O(s), O(1))$,
- (b) $\text{ST}(O(r), O(s), O(1)) \subsetneq \text{RST}(O(r), O(s), O(1)) \subsetneq \text{NST}(O(r), O(s), O(1))$.

Let us note that the lower bound of Theorem 6 for the problem CHECK-SORT in particular implies the following generalization of the main result of [10] to randomized computations:

Corollary 10. *The sorting problem (i.e., the problem of sorting a sequence of input strings) does not belong to the class $\text{LasVegas-RST}(o(\log N), O(\sqrt[3]{N}/\log N), O(1))$.* \dashv

4. LOWER BOUNDS FOR QUERY EVALUATION

Our lower bound for the SET-EQUALITY problem (Theorem 6) leads to the following lower bounds on the worst case data com-

plexity of database query evaluation problems in a streaming context:

Theorem 11 (Tight Bound for Relational Algebra).

- (a) For every relational algebra query Q , the problem of evaluating Q on a stream consisting of the tuples of the input database relations can be solved in $ST(O(\log N), O(1), O(1))$.
- (b) There exists a relational algebra query Q' such that the problem of evaluating Q' on a stream of the tuples of the input database relations does not belong to the class *LasVegas-RST* $(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$. \dashv

Proof: (a): It is straightforward to see that for every relational algebra query Q there exists a number c_Q such that Q can be evaluated within c_Q sequential scans and sorting steps. Every sequential scan accounts for a constant number of head reversals and constant internal memory space. Each sorting step can be accomplished using the sorting method of [7, Lemma 7] (which is a variant of the merge sort algorithm) with $O(\log N)$ head reversals and constant internal memory space. Since the number c_Q of necessary sorting steps and scans is constant (i.e., only depends on the query, but not on the input size N), the query Q can be evaluated by an $(O(\log N), O(1), O(1))$ -bounded deterministic Turing machine.

(b): Consider the relational algebra query

$$Q' := (R_1 - R_2) \cup (R_2 - R_1)$$

which computes the symmetric difference of two relations R_1 and R_2 . Note that the query result is empty if, and only if, $R_1 = R_2$. Therefore, any algorithm that evaluates Q' solves, in particular, the SET-EQUALITY problem. Hence, if Q' could be evaluated in *LasVegas-RST* $(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$, then SET-EQUALITY could be solved in $RST(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$, contradicting Theorem 6. \square

We also obtain lower bounds on the worst case data complexity of evaluating *XQuery* and *XPath* queries against XML document streams:

Theorem 12 (Lower Bound for XQuery). *There is an XQuery query Q such that the problem of evaluating Q on an input XML document stream of length N does not belong to the class *LasVegas-RST* $(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$.* \dashv

Theorem 13 (Lower Bound for XPath). *There is an XPath query Q such that the problem of filtering an input XML document stream with Q (i.e., checking whether at least one node of the document matches the query) does not belong to the class *co-RST* $(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$.* \dashv

For proving the Theorems 12 and 13, we represent an instance $x_1\#\dots\#x_m\#y_1\#\dots\#y_m\#$ of the SET-EQUALITY problem by an XML document of the form

```
<instance>
  <set1>
    <item> <string> x1 </string> </item>
    ...
    <item> <string> xm </string> </item>
  </set1>
  <set2>
    <item> <string> y1 </string> </item>
    ...
    <item> <string> ym </string> </item>
  </set2>
</instance>
```

(For technical reasons, we enclose every string x_i and y_j by a string-element and an item-element. For the proof of Theorem 12, one of the two would suffice, but for the proof of Theo-

rem 13 it is more convenient if each x_i and y_j is enclosed by two element nodes.)

It should be clear that, given as input $x_1\#\dots\#x_m\#y_1\#\dots\#y_m\#$, the above XML document can be produced by using a constant number of sequential scans, constant internal memory space, and two external memory tapes.

Proof of Theorem 12:

The SET-EQUALITY problem can be expressed by the following XQuery query $Q :=$

```
<result>
  if ( every $x in /instance/set1/item/string satisfies
      some $y in /instance/set2/item/string satisfies
        $x = $y )
    and
    ( every $y in /instance/set2/item/string satisfies
      some $x in /instance/set1/item/string satisfies
        $x = $y )
    then <true/>
    else ()
</result>
```

Note that if $\{x_1, \dots, x_m\} = \{y_1, \dots, y_m\}$, then Q returns the document $\langle \text{result} \rangle \langle \text{true} \rangle \langle \text{result} \rangle$, and otherwise Q returns the “empty” document $\langle \text{result} \rangle \langle \text{result} \rangle$. Thus, if Q could be evaluated in *LasVegas-RST* $(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$, then the SET-EQUALITY problem could be solved in $RST(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$, contradicting Theorem 6. \square

Proof of Theorem 13:

The XPath query Q of Figure 1 selects all item-nodes below *set1* whose string content does not occur as the string content of some item-node below *set2* (recall the “existential” semantics of XPath [20]). In other words: Q selects all (nodes that represent) elements in $X - Y$, for $X := \{x_1, \dots, x_m\}$ and $Y := \{y_1, \dots, y_m\}$.

Now assume, for contradiction, that the problem of filtering an input XML document stream with the XPath query Q (i.e., checking whether at least one document node is selected by Q) belongs to the class *co-RST* $(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$. Then, clearly, there exists an $(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$ -bounded randomized Turing machine T which has the following properties for every input $x_1\#\dots\#x_m\#y_1\#\dots\#y_m\#$ (where $X := \{x_1, \dots, x_m\}$ and $Y := \{y_1, \dots, y_m\}$):

- (1) If Q selects at least one node (i.e., $X - Y \neq \emptyset$, i.e., $X \not\subseteq Y$), then T accepts with probability 1.
- (2) If Q does not select any node (i.e., $X - Y = \emptyset$, i.e., $X \subseteq Y$), then T rejects with probability ≥ 0.5 .

This machine T can be used to solve the SET-EQUALITY problem by a machine \tilde{T} as follows: First, \tilde{T} starts T with input $x_1\#\dots\#x_m\#y_1\#\dots\#y_m\#$. Afterwards, \tilde{T} starts T with input $y_1\#\dots\#y_m\#x_1\#\dots\#x_m\#$. If both runs reject, then \tilde{T} accepts its entire input. Otherwise, \tilde{T} rejects.

Let us analyze the acceptance/rejection probabilities of \tilde{T} :

- (i) If $X \neq Y$, then either $X \not\subseteq Y$ or $Y \not\subseteq X$, and thus, due to (1), at least one of the two runs of T has to accept. The machine \tilde{T} will therefore reject with probability 1.
- (ii) If $X = Y$, then $X \subseteq Y$ and $Y \subseteq X$. Due to (2), we therefore know that each of the two runs of T will accept with probability ≥ 0.5 and thus, in total, \tilde{T} will accept with probability ≥ 0.25 .

To increase the acceptance probability to 0.5, we can start two independent runs of \tilde{T} and accept if at least one of the two runs accept. In total, this leads to a $(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$ -bounded

```

descendant::set1/child::item[not child::string =
ancestor::instance/child::set2/child::item/child::string]

```

Figure 1: The XPath query Q used in the proof of Theorem 13.

randomized Turing machine which, on every input $x_1\#\dots\#x_m\#y_1\#\dots\#y_m\#$,

- accepts with probability ≥ 0.5 , if $\{x_1, \dots, x_m\} = \{y_1, \dots, y_m\}$,
- rejects with probability 1, otherwise.

In other words: This machine shows that the SET-EQUALITY problem belongs to $\text{RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$, contradicting Theorem 6. Therefore, the problem of filtering an input XML document stream with the XPath query Q does not belong to the class $\text{co-RST}(o(\log N), O(\sqrt[4]{N}/\log N), O(1))$. \square

5. LIST MACHINES

This section as well as the subsequent sections are devoted to the proof of Theorem 6. For proving Theorem 6 we use *list machines*. The important advantage that these list machines have over the original Turing machines is that they make it fairly easy to track the “flow of information” during a computation.

In [10] we introduced the notion of deterministic list machines with output. In what follows, we propose a nondeterministic version of such machines without output, i.e., nondeterministic list machines for solving decision problems. To introduce *nondeterminism* to the notion of [10] requires some care — the straightforward approach where, instead of the transition functions used in [10], transition *relations* are allowed, will lead to a machine model that is too weak for adequately simulating nondeterministic Turing machines. Therefore, instead of using transition *relations*, the following notion of nondeterministic list machines allows explicit nondeterministic choices in transitions.

Definition 14 (Nondeterministic List Machine).

A *nondeterministic list machine (NLM)* is a tuple

$$M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$$

consisting of

- a $t \in \mathbb{N}$, the *number of lists*.
- an $m \in \mathbb{N}$, the *length of the input*.
- a finite set I whose elements are called *input numbers* (usually, $I \subseteq \mathbb{N}$ or $I \subseteq \{0, 1\}^*$).
- a finite set C whose elements are called *nondeterministic choices*.
- a finite set A whose elements are called (*abstract*) *states*.

We assume that I , C , and A are pairwise disjoint and do not contain the two special symbols ‘ \langle ’ and ‘ \rangle ’.

We call $\mathbb{A} := I \cup C \cup A \cup \{\langle, \rangle\}$ the *alphabet* of the machine.

- an *initial state* $a_0 \in A$.
- a *transition function*

$$\alpha : (A \setminus B) \times (\mathbb{A}^*)^t \times C \rightarrow (A \times \text{Movement}^t)$$

with $\text{Movement} := \left\{ \begin{array}{l} (\text{head-direction}, \text{move}) \mid \\ \text{head-direction} \in \{-1, +1\}, \\ \text{move} \in \{\text{true}, \text{false}\} \end{array} \right\}$.

- a set $B \subseteq A$ of *final states*.

- a set $B_{acc} \subseteq B$ of *accepting states*. (We use $B_{rej} := B \setminus B_{acc}$ to denote the set of *rejecting states*.) \dashv

Intuitively, an NLM $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$ operates as follows: The input is a sequence $(v_1, \dots, v_m) \in I^m$. Instead of tapes (as a Turing machine), an NLM operates on t lists. In particular, this means that a new list cell can be inserted between two existing cells. As for tapes, there is a read-write head operating on each list. Cells of the lists store strings in \mathbb{A}^* (and not just symbols from \mathbb{A}). Initially, the first list, called the *input list*, contains (v_1, \dots, v_m) , and all other lists are empty. The heads are on the left end of the lists. The transition function only determines the NLM’s new state and the head movements, and *not what is written into the list cells*. In each step of the computation, the heads move according to the transition function, by choosing “nondeterministically” an arbitrary element in C . In each computation step, the current state, the content of all current head positions, and the nondeterministic choice $c \in C$ used in the current transition, are written *behind* each head. When a final state is reached, the machine stops. If this final state belongs to B_{acc} , the according run is *accepting*; otherwise it is *rejecting*. Figure 2 illustrates a transition of an NLM. The formal definition of the semantics of nondeterministic list machines can be found in the full version of this paper.

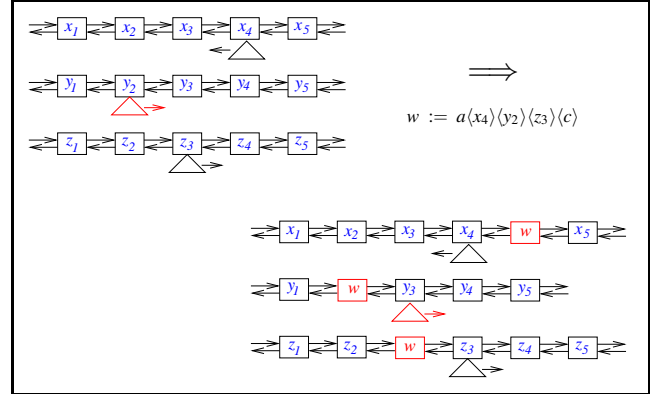


Figure 2: A transition of an NLM. The example transition is of the form $(a, x_4, y_2, z_3, c) \rightarrow (b, (-1, \text{false}), (1, \text{true}), (1, \text{false}))$. The new string w that is written in the tape cells consists of the current state a , the content of the list cells read before the transition, and the nondeterministic choice c .

An NLM is called *deterministic* if $|C| = 1$.

For every run ρ of an NLM M and for each list τ of M , we define $\text{rev}(\rho, \tau)$ to be the number of changes of the direction of the τ -th list’s head in run ρ . We say that M is (r, t) -*bounded*, for some $r, t \in \mathbb{N}$, if it has at most t lists, every run ρ of M is finite, and $1 + \sum_{\tau=1}^t \text{rev}(\rho, \tau) \leq r$.

Randomized list machines are defined in a similar way as randomized Turing machines: For configurations γ and γ' of an NLM M , the probability $\Pr(\gamma \rightarrow_M \gamma')$ that γ yields γ' in one step, is defined as $|\{c \in C : \gamma' \text{ is the } c\text{-successor of } \gamma\}|/|C|$. For a run $\rho = (\rho_1, \dots, \rho_\ell)$, the probability $\Pr(\rho)$ that M performs run ρ is the product of the probabilities $\Pr(\rho_i \rightarrow_M \rho_{i+1})$, for all $i < \ell$. For an input $\bar{v} \in I^m$, the probability that M accepts \bar{v} is defined as the sum of $\Pr(\rho)$ for all accepting runs ρ of M on input \bar{v} .

The following notation will be very convenient:

Definition 15 ($\rho_M(\bar{v}, \bar{c})$). Let M be an NLM and let $\ell \in \mathbb{N}$ such that every run of M has length $\leq \ell$. For every input $\bar{v} \in I^m$ and every sequence $\bar{c} = (c_1, \dots, c_\ell) \in C^\ell$, we use $\rho_M(\bar{v}, \bar{c})$ to denote the run (ρ_1, \dots, ρ_k) obtained by starting M with input \bar{v} and by making in its i -th step the nondeterministic choice c_i (i.e., ρ_{i+1} is the c_i -successor of ρ_i). \dashv

6. LIST MACHINES CAN SIMULATE TURING MACHINES

An important property of list machines is that they can simulate Turing machines in the following sense:

Lemma 16 (Simulation Lemma). Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$, $t \in \mathbb{N}$, and let $T = (Q, \Sigma, \Delta, q_0, F, F_{acc})$ be an (r, s, t) -bounded NTM with a total number of $t+u$ tapes and with $\{\square, \#\} \subseteq \Sigma$. Then for every $m, n \in \mathbb{N}$ there exists an $(r(m \cdot (n+1)), t)$ -bounded NLM $M_{m,n} = M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$ with $I = (\Sigma \setminus \{\square, \#\})^n$ and $|C| \leq 2^{O(\ell(m \cdot (n+1)))}$, where $\ell(N)$ is an upper bound on the length of T 's runs on input words of length N , and

$$|A| \leq 2^{d \cdot r(m \cdot (n+1)) \cdot s(m \cdot (n+1)) + 3t \cdot \log(m \cdot (n+1))}, \quad (2)$$

for some number $d = d(u, |Q|, |\Sigma|)$ that does not depend on r, m, n, t , such that for all $\bar{v} = (v_1, \dots, v_m) \in I^m$ we have

$$\Pr(M \text{ accepts } \bar{v}) = \Pr(T \text{ accepts } v_1 \# \dots \# v_m \#).$$

Furthermore, if T is deterministic, then M is deterministic, too. \dashv

In Section 8 we will use the simulation lemma to transfer the lower bound results for list machines to lower bound results for Turing machines.

In [10], the simulation lemma has been stated and proved for deterministic machines. For nondeterministic machines, the construction is based on the same idea. However, some further work is necessary to assure that the according list machine accepts with the same probability as the given Turing machine. Throughout the remainder of this section, the proof idea is given; a detailed proof of Lemma 16 can be found in the full version of this paper. For proving Lemma 16, the following straightforward characterization of probabilities for Turing machines is very convenient.

Definition 17 (C_T and $\rho_T(w, \bar{c})$). Let T be an NTM for which there exists a function $\ell : \mathbb{N} \rightarrow \mathbb{N}$ such that every run of T on a length N input word has length at most $\ell(N)$. Let $b := \max\{|\text{Next}_T(\gamma)| : \gamma \text{ is a configuration of } T\}$ be the maximum branching degree of T (note that b is finite since T 's transition relation is finite). Let $b' := \text{lcm}\{1, \dots, b\}$ be the least common multiple of the numbers $1, 2, \dots, b$, and let $C_T := \{1, \dots, b'\}$. For every $N \in \mathbb{N}$, every input word $w \in \Sigma^*$ of length N , and every sequence $\bar{c} = (c_1, \dots, c_{\ell(N)}) \in (C_T)^{\ell(N)}$, we define $\rho_T(w, \bar{c})$ to be the run (ρ_1, \dots, ρ_k) of T that is obtained by starting T with input w and by choosing in its i -th computation step the $(c_i \bmod |\text{Next}_T(\rho_i)|)$ -th of the $|\text{Next}_T(\rho_i)|$ possible next configurations. \dashv

Lemma 18. Let T be an NTM for which there exists a function $\ell : \mathbb{N} \rightarrow \mathbb{N}$ such that every run of T on a length N input word has length at most $\ell(N)$, and let C_T be chosen according to Definition 17. Then we have for every run ρ of T on an input w of length N that

$$\Pr(\rho) = \frac{|\{\bar{c} \in (C_T)^{\ell(N)} : \rho_T(w, \bar{c}) = \rho\}|}{|(C_T)^{\ell(N)}|}, \quad \text{and, in total,}$$

$$\Pr(T \text{ accepts } w) = \frac{|\{\bar{c} \in (C_T)^{\ell(N)} : \rho_T(w, \bar{c}) \text{ accepts}\}|}{|(C_T)^{\ell(N)}|}.$$

For proving Lemma 16, let T be an NTM. We construct an NLM M that simulates T . The lists of M represent the external memory tapes of T . More precisely, the cells of the lists of M represent segments, or *blocks*, of the corresponding external memory tapes of T in such a way that the content of a block at any step of the computation can be reconstructed from the content of the cell representing it. The blocks evolve dynamically in a way that is described below. M 's set C of *nondeterministic choices* is defined as $C := (C_T)^\ell$, where C_T is chosen according to Definition 17 and $\ell := \ell(m \cdot (n+1))$ is an upper bound on T 's running time and tape length, obtained from Lemma 3. Each step of the list machine corresponds to the sequence of Turing machine steps that are performed by T while none of its external memory tape heads changes its direction or leaves its current tape block. Of course, the length ℓ' of this sequence of T 's steps is bounded by T 's entire running time ℓ . Thus, if $c = (c_1, \dots, c_\ell) \in C = (C_T)^\ell$ is the nondeterministic choice used in M 's current step, the prefix of length ℓ' of c tells us, which nondeterministic choices (in the sense of Definition 17) T makes throughout the corresponding sequence of ℓ' steps. The *states* of M encode:

- The current state of the Turing machine T .
- The content and the head positions of the internal memory tapes $t+1, \dots, t+u$ of T .
- The head positions of the external memory tapes $1, \dots, t$.
- For each of the external memory tapes $1, \dots, t$, the boundaries of the block in which the head currently is.

Representing T 's current state and the content and head positions of the u internal memory tapes requires $|Q| \cdot 2^{O(s(m \cdot (n+1)))} \cdot s(m \cdot (n+1))^u$ states. The t head positions of the external memory tapes increase the number of states by a factor of ℓ^t . The $2t$ block boundaries increase the number of states by another factor of ℓ^{2t} . So overall, the number of states is bounded by $|Q| \cdot 2^{O(s(m \cdot (n+1)))} \cdot s(m \cdot (n+1))^u \cdot \ell^{3t}$. By Lemma 3, this yields the bound (2).

Initially, for an input word $v_1 \# \dots \# v_m \#$, the first Turing machine tape is split into m blocks which contain the input segments $v_i \#$ (for $1 \leq i < m$, respectively, $v_m \# \square^{\ell-(n+1)}$ (that is, the m -th input segment is padded by as many blank symbols as the Turing machine may enter throughout its computation). All other tapes just consist of one block which contains the blank string \square^ℓ . The heads in the initial configuration of M are on the first cells of their lists. Now we start the simulation: For a particular nondeterministic choice $c_1 = (c_{11}, c_{12}, \dots, c_{1\ell}) \in C = (C_T)^\ell$, we start T 's run $\rho_T(v_1 \# \dots, c_{11} c_{12} c_{13} \dots)$. As long as no head of the external memory tapes of T changes its direction or crosses the boundaries of its current block, M does not do anything. If a head on a tape $i_0 \in \{1, \dots, t\}$ crosses the boundaries of its block, the head i_0 of M moves to the next cell, and the previous cell is overwritten with sufficient information so that if it is visited again later, the content of the corresponding block of tape i_0 of T can be reconstructed. The blocks on all other tapes are split behind the current head position (“behind” is defined relative to the current direction in which the head moves). A new cell is inserted into the lists behind the head, this cell represents the newly created tape block that is behind the head. The newly created block starting with the current head position is represented by the (old) cell on which the head still stands. The case that a head on a tape $i_0 \in \{1, \dots, t\}$ changes its direction is treated similarly.

The simulation stops as soon as T has reached a final state; and M accepts if, and only if, T does. A close look at the possible runs of T and M shows that M has the same acceptance probabilities as T .

7. LOWER BOUNDS FOR LIST MACHINES

This section's main result is that it provides constraints on a list machine's parameters, which ensure that list machines which comply to these constraints can neither solve the *multiset equality problem* nor the *checksort problem*. In fact, we can show a slightly stronger result, the precise formulation of which requires the following observation.

Definition 19 (sortedness). Let $m \in \mathbb{N}$ and let π be a permutation of $\{1, \dots, m\}$. We define $\text{sortedness}(\pi)$ to be the length of the longest subsequence² of $(\pi(1), \dots, \pi(m))$ that is sorted in either ascending or descending order (i.e., that is a subsequence of $(1, \dots, m)$ or of $(m, \dots, 1)$). \dashv

Remark 20. It is well-known that for every permutation π of $\{1, \dots, m\}$, $\text{sortedness}(\pi) \in \Omega(\sqrt{m})$, and that there exists a particular permutation $\varphi := \varphi_m$ with $\text{sortedness}(\varphi) \leq 2\sqrt{m} - 1$. In fact, one way of finding such a permutation is to let $(\varphi(1), \dots, \varphi(m))$ be the numbers $1, \dots, m$, sorted lexicographically by their reverse binary representation. \dashv

Lemma 21 (Lower Bound for List Machines).

Let $k, m, n, r, t \in \mathbb{N}$ such that m is a power of 2 and $t \geq 2$, $m \geq 24 \cdot (t+1)^{4r} + 1$, $k \geq 2m + 3$, $n \geq 1 + (m^2 + 1) \cdot \log(2k)$. We let $I := \{0, 1\}^n$, identify I with the set $\{0, 1, \dots, 2^n - 1\}$, and divide it into m consecutive intervals I_1, \dots, I_m , each of length $2^n/m$. Let φ be a permutation of $\{1, \dots, m\}$ with $\text{sortedness}(\varphi) \leq 2\sqrt{m} - 1$, and let $\mathcal{S} := I_{\varphi(1)} \times \dots \times I_{\varphi(m)} \times I_1 \times \dots \times I_m$.

Then there is no (r, t) -bounded NLM $M = (t, 2m, I, C, A, a_0, \alpha, B, B_{acc})$ with $|A| \leq k$ and $I = \{0, 1\}^n$, such that for all $\bar{v} = (v_1, \dots, v_m, v'_1, \dots, v'_m) \in \mathcal{S}$ we have:

If $(v_1, \dots, v_m) = (v'_{\varphi(1)}, \dots, v'_{\varphi(m)})$, then $\Pr(M \text{ accepts } \bar{v}) \geq \frac{1}{2}$; otherwise $\Pr(M \text{ accepts } \bar{v}) = 0$. \dashv

It is straightforward to see that the above lemma, in particular, implies that neither the *(multiset equality problem)* nor the *checksort problem* can be solved by list machines with the according parameters.

The proof of Lemma 21 is based on the following ideas (due to space limitations, the detailed proof had to be deferred to the full version of this paper):

1. Suppose for contradiction that M is an NLM that meets the lemma's requirements.
2. Observe that there exists an upper bound ℓ on the length of M 's runs and a particular sequence $\bar{c} = (c_1, \dots, c_\ell) \in C^\ell$ of nondeterministic choices, such that for at least half of the inputs $\bar{v} := (v_1, \dots, v_m, v'_1, \dots, v'_m) \in \mathcal{S}$ with $(v_1, \dots, v_m) = (v'_{\varphi(1)}, \dots, v'_{\varphi(m)})$, the particular run $\rho_M(\bar{v}, \bar{c})$ accepts. We let $\mathcal{S}_{acc, \bar{c}} := \{\bar{v} \in \mathcal{S} : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}$ and, from now on, we only consider runs that are generated by the fixed sequence \bar{c} of nondeterministic choices.
3. Show that, throughout its computation, M can "mix" the relative order of its input values only to a rather limited extent. This can be used to show that for every run of M on every input $(v_1, \dots, v_m, v'_1, \dots, v'_m) \in \mathcal{S}$ there must be an index i_0 such that v_{i_0} and $v'_{\varphi(i_0)}$ are never compared throughout this run.
4. Use the notion of the *skeleton* of a run which, roughly speaking, is obtained from a run by replacing every input value v_i with

²A sequence (s_1, \dots, s_λ) is a *subsequence* of a sequence $(s'_1, \dots, s'_{\lambda'})$, if there exist indices $j_1 < \dots < j_\lambda$ such that $s_1 = s'_{j_1}, s_2 = s'_{j_2}, \dots, s_\lambda = s'_{j_\lambda}$.

its index i and by replacing every nondeterministic choice $c \in C$ with the wildcard symbol "?". In particular, the skeleton contains input *positions* rather than concrete input *values*; but given the skeleton together with the concrete input values and the sequence of nondeterministic choices, the entire run of M can be reconstructed.

5. Now choose ζ to be a skeleton that is generated by the run $\rho_M(\bar{v}, \bar{c})$ for as many input instances $\bar{v} \in \mathcal{S}_{acc, \bar{c}}$ as possible, and use $\mathcal{S}_{acc, \bar{c}, \zeta}$ to denote the set of all those input instances.
6. Due to 3. there must be an index i_0 such that for all inputs from $\mathcal{S}_{acc, \bar{c}, \zeta}$, the values v_{i_0} and $v'_{\varphi(i_0)}$ (i.e., the values from the input positions i_0 and $m + \varphi(i_0)$) are never compared throughout the run that has skeleton ζ . To simplify notation let us henceforth assume without loss of generality that $i_0 = 1$.
7. Now fix (v_2, \dots, v_m) such that the number of v_1 with $V(v_1) := (v_1, v_2, \dots, v_m, v_{\varphi^{-1}(1)}, \dots, v_{\varphi^{-1}(m)}) \in \mathcal{S}_{acc, \bar{c}, \zeta}$ is as large as possible.
8. Argue that, for our fixed (v_2, \dots, v_m) , there must be at least two distinct v_1 and w_1 such that $V(v_1) \in \mathcal{S}_{acc, \bar{c}, \zeta}$ and $V(w_1) \in \mathcal{S}_{acc, \bar{c}, \zeta}$. This is achieved by observing that the number of skeletons depends on the machine's parameters t, r, m, k , but *not* on n , and by using the lemma's assumption on the machine's parameters t, r, m, k, n .
9. Now we know that the input values of $V(v_1)$ and $V(w_1)$ coincide on all input positions except 1 and $m + \varphi(1)$. From 3. we know that the values from the positions 1 and $m + \varphi(1)$ are never compared throughout M 's (accepting) runs $\rho_M(V(v_1), \bar{c})$ and $\rho_M(V(w_1), \bar{c})$. From this we obtain an accepting run $\rho_M(\bar{u}, \bar{c})$ of M on input

$$\begin{aligned} \bar{u} &:= (u_1, \dots, u_m, u'_1, \dots, u'_m) \\ &:= (v_1, v_2, \dots, v_m, w_{\varphi^{-1}(1)}, w_{\varphi^{-1}(2)}, \dots, w_{\varphi^{-1}(m)}). \end{aligned}$$

In particular, this implies that $\Pr(M \text{ accepts } \bar{u}) > 0$. However, for this particular input \bar{u} we know that $u_1 = v_1 \neq w_1 = u_{\varphi(1)}$, and therefore, $(u_1, \dots, u_m) \neq (u'_{\varphi(1)}, \dots, u'_{\varphi(m)})$. This gives us a contradiction to the assumption that $\Pr(M \text{ accepts } \bar{v}) = 0$ for all inputs $\bar{v} = (v_1, \dots, v_m, v'_1, \dots, v'_m)$ with $(v_1, \dots, v_m) \neq (v'_{\varphi(1)}, \dots, v'_{\varphi(m)})$.

8. LOWER BOUNDS FOR TURING MACHINES

Lemma 22. Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ such that $r(N) = o(\log N)$ and $s(N) = o(\sqrt[4]{N}/r(N))$. Then, there is no $(r, s, O(1))$ -bounded $(\frac{1}{2}, 0)$ -RTM that solves the following problem CHECK- φ . \dashv

CHECK- φ

Instance: $v_1 \# \dots v_m \# v'_1 \# \dots v'_m \#$,

where $m \geq 0$ is a power of 2, and

$(v_1, \dots, v_m, v'_1, \dots, v'_m) \in I_{\varphi(1)} \times \dots \times I_{\varphi(m)} \times I_1 \times \dots \times I_m$, where $\varphi := \varphi_m$ is the permutation of $\{1, \dots, m\}$ obtained from Remark 20, and the sets $I_1, \dots, I_m \subseteq \{0, 1\}^{m^3}$ are obtained as the partition of the set $\{0, 1\}^{m^3}$ into m consecutive subsets, each of size $2^{(m^3)/m}$.

Problem: Decide if $(v_1, \dots, v_m) = (v'_{\varphi(1)}, \dots, v'_{\varphi(m)})$.

Proof: Suppose for contradiction that there is a $t \in \mathbb{N}$ such that the problem CHECK- φ is solved by an (r, s, t) -bounded $(\frac{1}{2}, 0)$ -RTM T . Without loss of generality we may assume that $t \geq 2$.

Let d be the constant introduced in Lemma 16 (the simulation lemma). Let m be a sufficiently large power of 2 such that

$$m \geq 24 \cdot (t+1)^{4 \cdot r \cdot (2m \cdot (m^3+1))} + 1 \quad \text{and} \quad (3)$$

$$m^3 \geq 1 + d \cdot t^2 \cdot r(2m \cdot (m^3+1)) \cdot s(2m \cdot (m^3+1)) + 3t \cdot \log(2m \cdot (m^3+1)). \quad (4)$$

Such an m exists because $r(N) = o(\log N)$ (for Equation (3)) and $r(N) \cdot s(N) = o(\sqrt[4]{N})$ (for Equation (4)). Let $n := m^3$ and $I := \{0, 1\}^n$. By Lemma 16, there is an $(r(2m \cdot (n+1)), t)$ -bounded NLM $M = (t, 2m, I, C, A, a_0, \alpha, B, B_{acc})$ with

$$k \leq 2^{d \cdot t^2 \cdot r(2m \cdot (n+1)) \cdot s(2m \cdot (n+1))} + 3t \cdot \log(2m \cdot (n+1))$$

states that simulates T on inputs from I^{2m} . In particular, for all instances

$$\bar{v} = (v_1, \dots, v_m, v'_1, \dots, v'_m) \in I_{\varphi(1)} \times \dots \times I_{\varphi(m)} \times I_1 \times \dots \times I_m$$

the following is true:

(*): If $(v_1, \dots, v_m) = (v'_{\varphi(1)}, \dots, v'_{\varphi(m)})$, then $\Pr(M \text{ accepts } \bar{v}) \geq \frac{1}{2}$; otherwise $\Pr(M \text{ accepts } \bar{v}) = 0$.

We clearly have $k \geq 2m + 3$. By (3), we have

$$m \geq 24 \cdot (t+1)^{4 \cdot r \cdot (2m \cdot (m^3+1))} + 1 = 24 \cdot (t+1)^{4 \cdot r \cdot (2m \cdot (n+1))} + 1.$$

By (4), we have

$$\begin{aligned} n &= m^3 \\ &\geq 1 + (m^2+1) \cdot (1 + d \cdot t^2 \cdot r(2m \cdot (m^3+1)) \cdot s(2m \cdot (m^3+1)) + 3t \cdot \log(2m \cdot (m^3+1))) \\ &= 1 + (m^2+1) \cdot (1 + d \cdot t^2 \cdot r(2m \cdot (n+1)) \cdot s(2m \cdot (n+1)) + 3t \cdot \log(2m \cdot (n+1))) \\ &\geq 1 + (m^2+1) \cdot (\log(2) + \log(k)) \\ &= 1 + (m^2+1) \cdot \log(2k). \end{aligned}$$

Thus, (*) is a contradiction to Lemma 21, and the proof of Lemma 22 is complete. \square

Proof of Theorem 6: For inputs that are instances of the problem CHECK- φ , the problems SET-EQUALITY, MULTISSET-EQUALITY, CHECK-SORT, and CHECK- φ coincide. Thus, Lemma 22 immediately implies Theorem 6. \square

9. CONCLUSION

We have proved tight lower bounds for the natural decision problems *(multi)set equality* and *checksort*, in our Turing machine based computation model for processing large data sets. These lower bounds do not only hold for deterministic, but even for *randomized* algorithms with one-sided bounded error probability. Our results are obtained by carefully analyzing the flow of information in a Turing machine computation.

As applications of these lower bound results, we obtained lower bounds on the worst case data complexity of query evaluation for the languages *XQuery*, *XPath*, and *relational algebra* on data streams.

We complement our lower bounds for *checksort* and *(multi)set equality* by proving that these problems can be solved by nondeterministic machines and by randomized machines with complementary one-sided error probabilities. As a consequence, we obtain

a separation between the deterministic, the randomized, the co-randomized, and the nondeterministic external memory complexity classes.

A specific problem for which we could not prove lower bounds, even though it looks very similar to the set equality problem, is the *disjoint sets problem*, which asks if two given sets of strings are disjoint. Another important future task is to develop techniques for proving lower bounds (a) for randomized computations with two-sided bounded error and (b) for appropriate problems in a setting where $\Omega(\log N)$ head reversals (i.e., sequential scans of external memory devices) are available.

10. REFERENCES

- [1] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *Proc. FOCS'04*, pages 540–549, 2004.
- [2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.
- [3] L. Arge and P. Bro Miltersen. On showing lower bounds for external-memory computational geometry problems. In J. Abello and J. Vitter, editors, *External Memory Algorithms and Visualization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 139–159. 1999.
- [4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. PODS'02*, pages 1–16, 2002.
- [5] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the memory requirements of XPath evaluation over XML streams. In *Proc. PODS'04*, pages 177–188, 2004.
- [6] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. Buffering in query evaluation over XML streams. In *Proc. PODS'05*, pages 216–227, 2005.
- [7] J. Chen and C.-K. Yap. Reversal complexity. *SIAM Journal on Computing*, 20(4):622–638, 1991.
- [8] M. Grohe, C. Koch, and N. Schweikardt. The complexity of querying external memory and streaming data. In *Proc. FCT'05*, volume 3623 of *Springer LNCS*, pages 1–16, 2005.
- [9] M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. In *Proc. ICALP'05*, volume 3580 of *Springer LNCS*, pages 1076–1088, 2005. (Best paper award at ICALP'05, track B.).
- [10] M. Grohe and N. Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proc. PODS'05*, pages 238–249, 2005.
- [11] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *External memory algorithms*, volume 50, pages 107–118. DIMACS Series In Discrete Mathematics And Theoretical Computer Science, 1999.
- [12] J. Hromkovic. *Design and Analysis of Randomized Algorithms*. Springer-Verlag, 1998.
- [13] U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *Springer LNCS*. 2003.
- [14] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [15] J. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
- [16] S. Muthukrishnan. Data streams: algorithms and applications. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 413–413, 2003.
- [17] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [18] J. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33:209–271, 2001.
- [19] K. Wagner and G. Wechsung. *Computational Complexity*. VEB Deutscher Verlag der Wissenschaften, 1986.
- [20] World Wide Web Consortium. XML Path Language (XPath) 2.0. W3C Candidate Recommendation 3 November 2005, 2005. <http://www.w3.org/TR/xpath20/>.