Streaming Complexity of Checking Priority Queues*†

Nathanaël François[1] and Frédéric Magniez[2]

1 **Univ Paris Diderot, Sorbonne Paris-Cité, LIAFA, CNRS, 75205 Paris, France**
nathanael.francois@liafa.univ-paris-diderot.fr

2 **CNRS, LIAFA, Univ Paris Diderot, Sorbonne Paris-Cité, 75205 Paris, France**
frederic.magniez@univ-paris-diderot.fr

Abstract

This work is in the line of designing efficient checkers for testing the reliability of some massive data structures. Given a sequential access to the insert/extract operations on such a structure, one would like to decide, a posteriori only, if it corresponds to the evolution of a reliable structure. In a context of massive data, one would like to minimize both the amount of reliable memory of the checker and the number of passes on the sequence of operations.

Chu, Kannan and McGregor [9] initiated the study of checking priority queues in this setting. They showed that the use of timestamps allows to check a priority queue with a single pass and memory space $\tilde{O}(\sqrt{N})$. Later, Chakrabarti, Cormode, Kondapally and McGregor [7] removed the use of timestamps, and proved that more passes do not help.

We show that, even in the presence of timestamps, more passes do not help, solving an open problem of [9, 7]. On the other hand, we show that a second pass, but in *reverse* direction, shrinks the memory space to $\tilde{O}((\log N)^2)$, extending a phenomenon the first time observed by Magniez, Mathieu and Nayak [15] for checking well-parenthesized expressions.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity

Keywords and phrases Streaming Algorithms, Communication Complexity, Priority Queue

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.454

1 Introduction

The reliability of memory is central and becomes challenging when it is massive. In the context of program checking [4] this problem has been addressed by Blum, Evans, Gemmell, Kannan and Naor [3]. They designed on-line checkers that use a small amount of reliable memory to test the behavior of some data structures. Checkers are allowed to be randomized and to err with small error probability. In that case the error probability is not over the inputs but over the random coins of the algorithm.

Chu, Kannan and McGregor [9] revisited this problem for priority queue data structures, where the checker only has to detect an error after processing an entire sequence of data accesses. This can be rephrased as a one-pass streaming recognition problem. Streaming algorithms sequentially scan the whole input piece by piece in one sequential pass, or in a small number of passes, while using sublinear memory space. In our context, the stream is defined by the sequence of insertions and extractions on the priority queue. Using a streaming

* Full version available on http://arxiv.org/abs/1209.4971

† Supported by the French ANR Defis program under contract ANR-08-EMER-012 (QRAC project) and by the French ANR Blanc program under contract ANR-12-BS02-005 (RDAM project)

30th Symposium on Theoretical Aspects of Computer Science (STACS'13).
Editors: Natacha Portier and Thomas Wilke; pp. 454–465

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

algorithm, the objective is then to decide if the stream corresponds to a correct implementation of a priority queue. We also consider collection data structures that implement multisets.

▶ **Definition 1** (COLLECTION,PQ)**.** Let Σ_0 be some alphabet. Let $\Sigma = \{\texttt{ins}(a), \texttt{ext}(a) : a \in \Sigma_0\}$. For $w \in \Sigma^N$, define inductively multisets M_i by $M_0 = \emptyset$, $M_i = M_{i-1} \setminus \{a\}$ if $w[i] = \texttt{ext}(a)$, and $M_i = M_{i-1} \cup \{a\}$ if $w[i] = \texttt{ins}(a)$.
Then $w \in \text{COLLECTION}(\Sigma_0)$ if and only if $M_n = \emptyset$ and $a \in M_{i-1}$ when $w[i] = \texttt{ext}(a)$, for $i = 1, \ldots, N$. Moreover, $w \in \text{PQ}(U)$, for $U \in \mathbb{N}$, if and only if $w \in \text{COLLECTION}(\{0, 1, \ldots, U\})$ and $a = \max(M_{i-1})$ when $w[i] = \texttt{ext}(a)$, for $i = 1, \ldots, N$.

Streaming algorithms were initially designed with a single pass: when a piece of the stream has been read, it is gone for ever. This makes those algorithms of practical interest for online context, such as network monitoring, for which first streaming algorithms were developed [1]. Motivated by the explosion in the size of the data that algorithms are called upon to process in everyday real-time applications, the area of streaming algorithms has experienced tremendous growth over the last decade in many applications. In particular, a streaming algorithm can model an external read-only memory. Examples of such applications occur in bioinformatics for genome decoding, or in Web databases for the search of documents. In that context, considering multi-pass streaming algorithm is relevant.

Using standard arguments one can establish that every p-pass randomized streaming algorithm needs memory space $\Omega(N/p)$ for recognizing COLLECTION. Nonetheless, Chakrabarti, Cormode, Kondapally and McGregor [7] gave a one-pass randomized algorithm for PQ using memory space $\tilde{O}(\sqrt{N})$. They also showed that several passes do not help, since any p-pass randomized algorithm would require memory space $\Omega(\sqrt{N}/p)$. A similar lower bound was showed independently, but using different tools, by Jain and Nayak [10]. The case of a single pass was established previously by Magniez, Mathieu and Nayak [15] for checking the well-formedness of parenthesis expressions, or equivalently the behavior of a stack.

A simpler variant of PQ with timestamps was in fact first studied by Chu, Kannan and McGregor [9], where now each item is inserted to the queue with its index.

▶ **Definition 2** (PQ-TS)**.** Let $\Sigma = \{\texttt{ins}(a), \texttt{ext}(a) : a \in \{0, 1, \ldots, U\}\} \times \mathbb{N}$. Let $w \in \Sigma^N$. Then $w \in \text{PQ-TS}(U)$ if and only if $w \in \text{COLLECTION}(\Sigma)$, $w[1, \ldots, N][1] \in \text{PQ}(U)$, and $w[i][2] = i$ when $w[i][1] = \texttt{ins}(a)$.

Nonetheless the two works [9, 7] left open problems. The lower bound of [7] was proved only for PQ, and no significant lower bounds for PQ-TS established. Moreover, the streaming complexity of PQ for algorithms processing the stream in both directions was not studied.

Even though recognizing PQ-TS is obviously easier than recognizing PQ, our first contribution (Section 3) consists in showing that they both obey the same limitation, even with multiple passes in the same direction.

▶ **Theorem 3.** *Every p-pass randomized streaming algorithm recognizing* PQ-TS$(3N/2)$ *with bounded error* $1/3$ *requires memory space* $\Omega(\sqrt{N}/p)$ *for inputs of length* N*.*

As a consequence, since this lower bound uses very restricted hard instances, it models most of possible variations. For instance, assuming that the input is in COLLECTION and has no duplicates is not sufficient to guarantee a faster algorithm. Theorem 3 is proved by introducing a related communication problem with $\Theta(\sqrt{N})$ players. Then we reduce the number of players to 3, and prove a lower bound on the information carried by players, leading to the desired lower bound. We are following the *information cost* approach taken in [8, 17, 2, 12, 11], among other works. Recently, the information cost appeared as one of

the most central notion in communication complexity [6, 5, 13]. The information cost of a protocol is the amount of information that messages carry about players' inputs. We adapt this notion to suit both the nature of streaming algorithms and of our problem.

Even if our result suggests that allowing multiple passes does not help, one could also consider the case of bidirectional passes. We believe that it is a natural relaxation of multi-pass streaming algorithms where the stream models some external read-only memory. In that case, we show that a second pass, but in reverse order, makes the problem of checking PQ easy, even with no timestamps (Section 4). A similar phenomenon has been established previously in [15] for checking the well-formedness of parenthesis expressions. Their problem is simpler than ours, and therefore our algorithm is more general.

▶ **Theorem 4.** *There is a bidirectional 2-pass randomized streaming algorithm recognizing* $\mathrm{PQ}(U)$ *with memory space* $\mathrm{O}((\log N)(\log U + \log N))$*, time per processing item* $\mathrm{polylog}(N, U)$*, and one-sided bounded error* N^{-c}*, for inputs of length* N *and any constant* $c > 0$*.*

Our algorithm uses a hierarchical data structure similar to the one introduced in [15] for checking well-parenthesized expressions. At high level, it also behaves similarly. It performs one pass in each direction and makes an on-line compression of past information in at most $\log N$ hashcodes. While this compression can lose information, the compression technique ensures that a mistake is always detected in one of the two directions. Nonetheless our algorithm differs on two main points. First, unlike parenthesized expressions, PQ is not symmetric. Therefore one has to design an algorithm for each pass. Second, the one-pass algorithm for PQ [7] is technically more advanced than the one of [15]. Thus designing a bidirectional 2-pass algorithm for PQ is more challenging.

Theorems 3 and 4 point out a strange situation but not isolated at all. Languages studied in [9, 15, 7, 14] and in this paper have space complexity $\Theta(\sqrt{N}\mathrm{polylog}(N))$ for a single pass, $\Omega(\sqrt{N}/p)$ for p passes in the same direction, and $\mathrm{polylog}(N)$ for 2 passes but one in each direction. We hope this paper makes progress in the study of that phenomenon.

2 Preliminaries

In streaming algorithms (see [16] for an introduction), a *pass* on an input $w \in \Sigma^N$, for some alphabet Σ, means that w is given as an *input stream* $w[1], w[2], \ldots, w[N]$, which arrives sequentially, i.e., letter by letter in this order. For simplicity, we assume throughout this article that the input length N is always given to the algorithm in advance. Nonetheless, all our algorithms can be adapted to the case in which N is unknown until the end of a pass.

▶ **Definition 5** (Streaming algorithm)**.** A p-pass randomized *streaming algorithm* with space $s(N)$ and time $t(N)$ is a randomized algorithm that, given $w \in \Sigma^N$ as an input stream,

- performs p sequential passes on w;
- maintains a memory space of size at most $s(N)$ bits while reading w;
- has running time at most $t(N)$ per processed letter $w[i]$;
- has preprocessing and postprocessing time at most $t(N)$.

The algorithm is *bidirectional* if it is allowed to access to the input in the reverse order, after reaching the end of the input. Then p is the total number of passes in either direction.

The proof of our lower bound uses the language of communication complexity with multi-players, and is based on information theory arguments. We consider *number-in-hand* and *message-passing* communication protocols. Each player is given some input, and can communicate with another player according to the rules of the protocol. Our players are

embedded into a directed circle, so that each player can receive (resp. transmit) a message from its unique predecessor (resp. successor). Each player send a message after receiving one, until the end of the protocol is reached. Players have no space and time restriction. Only the number of rounds and the size of messages are constrained.

Consider a randomized multi-player communication protocol P. We consider only two types of random source, that we call *coins*. Each player has access to its own independent source of *private coins*. In addition, all players share another common source of *public coins*. The output of P is announced by the last player. This is therefore the last message of the last player. We say that P is with bounded error ϵ when P errs with probability at most ε over the private and public coins. The *transcript* Π of P is the concatenation of all messages sent by all players, including all public coins. In particular, it contains the output of P, since it is given by the last player. Given a subset S of players, we let Π_S be the concatenation of all messages sent by players in S, including again all public coins.

We now remind the usual notions of entropy H and mutual information I. Let X, Y, Z be random variables. Then $\mathrm{H}(X) = -\mathbb{E}_{x\leftarrow X} \log \Pr(X = x)$, $\mathrm{H}(X|Y = y) = -\mathbb{E}_{y\leftarrow Y} \log \Pr(X = x|Y = y)$, $\mathrm{H}(X|Y) = \mathbb{E}_{y\leftarrow Y} \mathrm{H}(X|Y = y)$, and $\mathrm{I}(X : Y|Z) = \mathrm{H}(X|Z) - \mathrm{H}(X|Y, Z)$. The entropy and the mutual information are non negative and satisfy $\mathrm{I}(X : Y|Z) = \mathrm{I}(Y : X|Z)$.

The mutual information between two random variables is connected to the Hellinger distance h between their respective distribution probabilities. Given a random variable X we also denote by X its underlying distribution.

▶ **Proposition 6** (Average encoding). *Let X, Y be random variables. Then $\mathbb{E}_{y\leftarrow Y} \mathrm{h}^2(X|_{Y=y}, X) \leq \kappa \mathrm{I}(X : Y)$, where $\kappa = \frac{\ln 2}{2}$.*

The Hellinger distance also generalizes the cut-and-paste lemma to randomized protocols.

▶ **Proposition 7** (Cut and paste). *Let P be a 2-player randomized protocol. Let $\Pi(x, y)$ denote the random variable representing the transcript in P when Players A, B have resp. inputs x, y. Then $\mathrm{h}(\Pi(x, y), \Pi(u, v)) = \mathrm{h}(\Pi(x, v), \Pi(u, y))$, for all pairs (x, y) and (u, v).*

Last we use that the square of the Hellinger distance is convex, and the following connection to the more convention ℓ_1-distance: $\mathrm{h}(X, Y)^2 \leq \frac{1}{2}\|X - Y\|_1 \leq \sqrt{2}\mathrm{h}(X, Y)$. For a reference on these results, see [10].

3 Lower bound for PQ-TS

The proof of our lower bound consists in first translating it into a $3m$-player communication problem, for some large m; then reducing the number of players to 3 using the information cost approach; and last studying the base case of 3 players using information theory arguments.

3.1 From streaming algorithms to communication protocols

In this section, we write a instead of $\texttt{ins}(a)$ and $\bar{a}$ instead of $\texttt{ext}(a)$. Consider the following set of hard instances of size $N = (2n + 2)m$:

RAINDROPS(m, n) (see LHS of Figure 1)

- For $i = 1, 2, \ldots, m$, repeat the following motif:
  - For $j = 1, 2, \ldots, n$, insert either $v_{i,j} = 3(ni - j)$ or $v_{i,j} = 3(ni - j) + 2$
  - Insert either $a_i = 3(ni-(k_i-1))+1$ or $a_i = 3(ni-k_i)+1$, for some $k_i \in \{2, \ldots, n\}$
  - Extract $v_{i,1}, v_{i,2}, \ldots, v_{i,k_i-1}, a_i$ in decreasing order
- Extract everything left in decreasing order

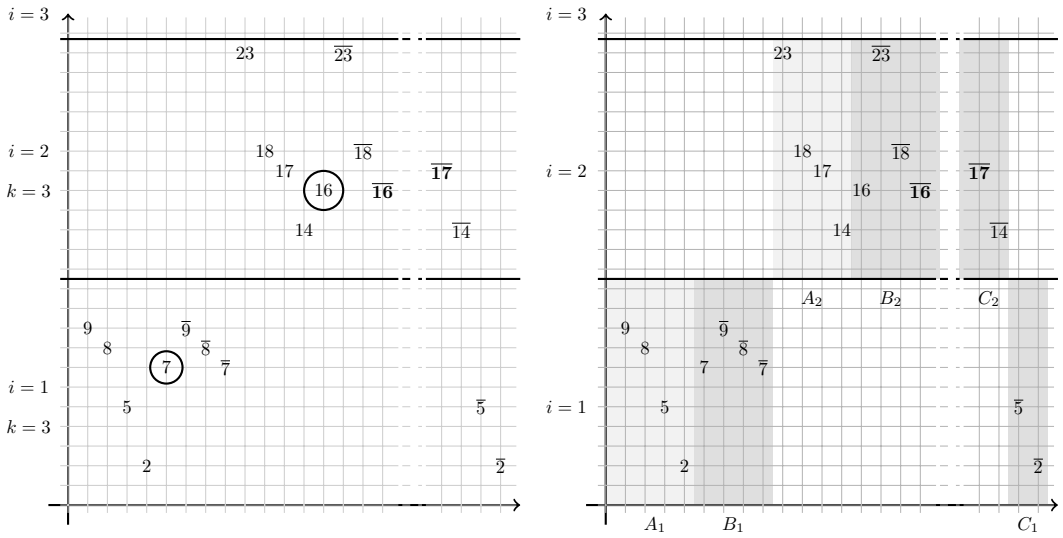

Figure 1 Left: Instance of RAINDROPS$(m, 4)$ with one error: 17 is extracted after 16. Insertions a_i are circled. Right: Cutting RAINDROPS$(m, 4)$ into $3m$ pieces to make it a communication problem. Players' input are within each corresponding region.

Observe that such an instance is in COLLECTION. One can compute the timestamps for each value by maintaining only $\mathrm{O}(\log N)$ additionnal bits. Last, there is only one potential error in each motif that can make it outside of PQ-TS. Indeed, $v_{i,1}, v_{i,2}, \ldots, v_{i,k_i-1}, a_i$ are in decreasing order up to a switch between a_i and v_{i,k_i-1}.

Given such an instance as a stream, an algorithm for PQ-TS must decide if an error occurs between $\overline{a_i}$ and $\overline{v_{i,k_i}}$, for some i. Intuitively, if the memory space is less than εn, for a small enough constant $\varepsilon > 0$, then the algorithm cannot remember all the values $(v_{i,j})_j$ when a_i is extracted, and therefore cannot check a potential error with a_i. The next opportunity is during the last sequence of extractions. But then, the algorithm has to remember all values $(a_i)_i$, which is again impossible if the memory space is less than εm.

In order to formalize this intuition, Lemma 8 first translates our problem into a communication one between $3m$ players in the same way as [15], as shown on the RHS of Figure 1. Then we analyze its complexity using information theory arguments in Section 3.2.

Any insertion and extraction of an instance in RAINDROPS(m, n) can be described by its index and a single bit. Let $x_i[j] \in \{0, 1\}$ such that $v_{i,j} = 3(ni - j) + 2x_i[j]$. Similarly, let $d_i \in \{0, 1\}$ such that $a_i = 3(ni - k_i) + 1 + 3d_i$. For simplicity, we write $\mathbf{x}$ instead of $(x_i)_{1 \le i \le m}$. Similarly, we use the notations $\mathbf{k}$ and $\mathbf{d}$. Then our related communication problem is:

WEAKINDEX(m, n)

- Input for players $(A_i, B_i, C_i)_{1 \le i \le m}$:
  - Player A_i has a sequence $x_i \in \{0, 1\}^n$
  - Player B_i has $x_i[1, k_i - 1]$, with $k_i \in \{2, \ldots, n\}$ and $d_i \in \{0, 1\}$
  - Player C_i has $x_i[k_i, n]$
- Output: $f_m(\mathbf{x}, \mathbf{k}, \mathbf{d}) = \bigvee_{i=1}^{m} f(x_i, k_i, d_i)$, where $f(x, k, d) = [(d = 0) \wedge (x[k] = 1)]$
- Communication settings:
  - One round: each player sends a message to the next player according to the diagram $A_1 \to B_1 \to A_2 \to \cdots \to B_m \to C_m \to C_{m-1} \to \cdots \to C_1$.
  - Multiple rounds: If there is at least one round left, C_1 sends a message to A_1, and then players continue with the next round.

▶ **Lemma 8.** *Assume there is a p-pass randomized streaming algorithm for deciding if an instance of* RAINDROPS(n, m) *is in* PQ-TS$(3mn)$ *with memory space $s(m, n)$ and bounded error ε. Then there is a p-round randomized protocol for* WEAKINDEX(n, m) *with bounded error ε such that each message has size at most $s(m, n)$.*

We are now ready to give the structure of the proof of Theorem 3, which has techniques based on information theory. Define the following collapsing distribution μ_0 of hard inputs (x, k, d), encoding instances of RAINDROPS$(1, n)$, where f always takes value 0. Distribution μ_0 is such that (x, k) is uniform on $\{0,1\}^n \times \{2, \dots, n\}$ and, given x, k, the bit $d \in \{0,1\}$ is uniform if $x[k] = 0$, and $d = 1$ if $x[k] = 1$. From now on, (X, K, D) are random variables distributed according to μ_0, and (x, k, d) denote any of their values.

Then the proof of Theorem 3 consists in studying the information cost of any communication protocol for WEAKINDEX(n, m), which is a lower bound on its communication complexity. Using that μ_0 is collapsing for f, Lemma 9 establishes a direct sum on the information cost of WEAKINDEX(n, m). Then, even if f is constant on μ_0, Lemma 12 lower bounds the information cost of a single instance of WEAKINDEX$(n, 1)$.

Proof of Theorem 3. Let n, N be positive integers such that $N = (2n + 2)n$. Assume that there exists a p-pass randomized algorithm that recognizes PQ-TS$(3N/2)$, with memory space αn and bounded error ε, for inputs of size N. Then, by Lemma 8, there a p-round randomized protocol P for WEAKINDEX(n, n) such that each message has size at most αn. By Lemma 9, one can derive from P another $(p + 1)$-round randomized protocol P' for WEAKINDEX$(n, 1)$ with bounded error ε, and transcript Π' satisfying $|\Pi'| \leq 3(t + 1)\alpha n$ and $\max\{\mathrm{I}(D : \Pi'_B | X, K), \mathrm{I}(K, D : \Pi'_C | X)\} \leq (p + 1)\alpha$. Then by Lemma 12, $3(p + 1)\alpha \geq (1 - 2\varepsilon)/10$, that is $\alpha = \mathrm{O}(1/p)$, concluding the proof. ◀

3.2 Communication complexity lower bound

We first reduce the general problem WEAKINDEX(n, m) with $3m$ players to a single instance of WEAKINDEX$(n, 1)$ with 3 players. In order to do so we exploit the direct sum property of the information cost. The use of a collapsing distribution where f is always 0 is crucial.

▶ **Lemma 9.** *If there is a p-round randomized protocol P for* WEAKINDEX(n, m) *with bounded error ε and messages of size at most $s(m, n)$, then there is a $(p + 1)$-round randomized protocol P' for* WEAKINDEX$(n, 1)$ *with bounded error ϵ, and transcript P' satisfying $|\Pi'| \leq 3(p + 1)s(m, n)$ and $\max\{\mathrm{I}(D : \Pi'_B | X, K), \mathrm{I}(K, D : \Pi'_C | X)\} \leq \frac{p+1}{m} s(m, n)$.*

Sketch of proof. Given a protocol P, we show how to construct another protocol P' for any instance (x, k, d) of WEAKINDEX$(n, 1)$. In order to avoid any confusion, we denote by A, B and C the three players of P', and by $(A_i, B_i, C_i)_i$ the ones of P.

Protocol P'

- Using public coins, all players generate uniformly at random $j \in \{1, \dots, m\}$, and $x_i \in \{0,1\}^n$ for $i \neq j$
- Players A, B and C set respectively their inputs to the ones of A_j, B_j, C_j
- For all $i > j$, Player B generates, using its private coins, uniformly at random $k_i \in \{2, \dots, n\}$, and then it generates uniformly at random d_i such that $f(x_i, k_i, d_i) = 0$
- For all $i < j$, Player C generates, using its private coins, uniformly at random $k_i \in \{2, \dots, n\}$, and then it generates uniformly at random d_i such that $f(x_i, k_i, d_i) = 0$
- Players A, B and C run P as follows. A simulates A_j only, B simulates B_j and $(A_i, B_i, C_i)_{i>j}$, and C simulates C_j and $(A_i, B_i, C_i)_{i<j}$.

Observe that A starts the protocol if $j = 1$, and C starts otherwise. Moreover C stops the simulation after p rounds if $j = 1$, and after $p + 1$ rounds otherwise. For all $i \neq j$, entries are generated such that $f(x_i, k_i, a_i) = 0$, therefore $f_m(\mathbf{X}, \mathbf{k}, \mathbf{d}) = f(x_j, k_j, a_j) = f(x, k, a)$, and P' has the same bounded error than P.

By applying the chain rule, one can see that P' satisfies the required conditions of the lemma. ◀

We now prove a trade-off between the bounded error of a protocol for a single instance of WeakIndex$(n, 1)$ and its information cost. The proof involves some of the tools of [10] but with some additional obstacles to apply them. The inherent difficulty is due to that we have 3 players whereas the cute-and-paste property applies to 2-player protocols. Therefore we have to group 2 players together.

Given some parameters (x, k, a) for an input of WeakIndex$(n, 1)$, we denote by $\Pi(x, k, a)$ the random variable describing the transcript Π of our protocol. We start by two lemmas exploiting the average encoding theorem (proofs omitted).

▶ **Lemma 10.** *Let P be a randomized protocol for* WeakIndex$(n, 1)$ *with transcript Π satisfying $|\Pi| \leq \alpha n$ and $\mathrm{I}(K, D : \Pi_C | X) \leq \alpha$. Then*

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \mathrm{h}^2(\Pi(x[1,l-1]0X[l+1,n],l,1), \Pi(x[1,l-1]1X[l+1,n],l,1)) \leq 28\alpha,$$

where $l \in [\frac{n}{2} + 1, n]$ and $x[1, l-1]$ are uniformly distributed.

▶ **Lemma 11.** *Let P be a randomized protocol for* WeakIndex$(n, 1)$ *with transcript Π satisfying $\mathrm{I}(D : \Pi_B | X, K) \leq \alpha$. Then*

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \mathrm{h}^2(\Pi(x[1,l-1]0X[l+1,n],l,0), \Pi(x[1,l-1]0X[l+1,n],l,1)) \leq 12\alpha,$$

where $l \in [\frac{n}{2} + 1, n]$ and $x[1, l-1]$ are uniformly distributed.

We now end with the main lemma which combines both previous ones and applies the cut-and-paste property, where Players A, C are grouped.

▶ **Lemma 12.** *Let P be a randomized protocol for* WeakIndex$(n, 1)$ *with bounded error ϵ, and transcript Π satisfying $|\Pi| \leq \alpha n$ and $\max\{I(D : \Pi_B | X, K), I(K, D : \Pi_C | X)\} \leq \alpha$. Then $\alpha \geq (1 - 2\varepsilon)/10$.*

Proof. Let L be a uniform integer random variable in $[\frac{n}{2} + 1, n]$. Remind that we enforce the output of P to be part of Π. Therefore, any player, and in particular B, can compute f with bounded error ε given Π. Since $f(x[1, l-1]0X[l+1, n], l, 0) = 0$ and $f(x[1, l-1]1X[l+1, n], l, 1) = 1$, the error parameter ε must satisfies

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \|\Pi(x[1,l-1]0X[l+1,n],l,0) - \Pi(x[1,l-1]1X[l+1,n],l,0)\|_1 \geq 2(1-2\varepsilon).$$

The rest of the proof consists in upper bounding the LHS by 19α.

Applying the triangle inequality and that $(u+v)^2 \leq 2(u^2+v^2)$ on the inequalities of Lemmas 10 and 11 gives

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \mathrm{h}^2(\Pi(x[1,l-1]0X[l+1,n],l,0), \Pi(x[1,l-1]1X[l+1,n],l,1)) \leq 30\alpha.$$

We then apply the cut-and-paste property by considering (A, C) as a single player with transcript $\Pi_{A,C}$. Therefore

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \mathrm{h}^2(\Pi(x[1,l-1]0X[l+1,n],l,1), \Pi(x[1,l-1]1X[l+1,n],l,0)) \leq 30\alpha.$$

Combining again with the inequality from Lemma 11 gives

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \mathrm{h}^2(\Pi(x[1,l-1]0X[l+1,n],l,0),\Pi(x[1,l-1]1X[l+1,n],l,0)) \leq 42\alpha.$$

Last, we get the requested upper bound by using the connexion between the Hellinger distance and the ℓ_1-distance, and the convexity of the square function. ◀

4 Bidirectional streaming algorithm for PQ

Remember that in this section our stream is given without any timestamps. Therefore we consider in this section only streams w of $\mathtt{ins}(a), \mathtt{ext}(a)$, where $a \in [0,U]$. For the sake of clarity, we assume for now that the stream has no duplicate. Our algorithms can be extended to the general case, but the technical difficulties shadow the main ideas.

Up to padding we can assume that N is a power of 2: we append a sequence of $\mathtt{ins}(a)\mathtt{ext}(a)\mathtt{ins}(a+1)\mathtt{ext}(a+1)\ldots$ of suitable length, where a is large enough so that there is no duplicate (assuming that w is of even size, otherwise $w \notin \mathrm{PQ}(U)$). We use $\mathrm{O}(\log N)$ bits of memory to store, after the first pass, the number of letters padded.

We use a hash function based on the one used by the Karp-Rabin algorithm for pattern matching. For all this section, let p be a prime number in $\{\max(2U+1, N^{c+1}), \ldots, 2\max(2U+1, N^{c+1})\}$, for some fixed constant $c \geq 1$. Since our hash function is linear we only define it for single insertion/extraction as

$$\mathrm{hash}(\mathtt{ins}(a)) = \alpha^a \mod p, \quad \text{and} \quad \mathrm{hash}(\mathtt{ext}(a)) = -\alpha^a \mod p,$$

where α is a randomly chosen integer in $[0, p-1]$. This is the unique source of randomness of our algorithm. A hashcode h *encodes* a sequence w if $h = \mathrm{hash}(w)$ as a formal polynomial in α. In that case we say that h *includes* $w[i]$, for all i. Moreover w is *balanced* if the same integers have been inserted and extracted. In that case it must be that $h = 0$. We also say that h is balanced it it encodes a balanced sequence w. The converse is also true with high probability by the Schwartz-Zippel lemma.

▶ **Fact 13.** *Let w be some unbalanced sequence. Then* $\Pr(\mathrm{hash}(w) = 0) \leq \frac{N}{p} \leq \frac{1}{N^c}$.

The forward-pass algorithm was introduced in [7], but the reverse-pass one is even simpler. As a warming up, we start by introducing the later algorithm. In order to keep it simple to understand, we do not optimize it fully. Last define the instruction $\mathtt{Update}(h, v)$ that returns $(h + \mathrm{hash}(v) \mod p)$ *and* updates h to that value.

4.1 One-reverse-pass algorithm for PQ

Our reverse-pass algorithm decomposes the stream w into blocks. We call a valley an extraction $w[t] = \mathtt{ext}(a)$ with $w[t+1] = \mathtt{ins}(b)$. A new block starts at each valley. To the i-th block we associate a hashcode h_i and an integer m_i. Hashcode h_i encodes all extractions within the block and matching insertions. Integer m_i is the minimum of extractions in the block. With the values $(m_i)_i$, one can encode insertions in the correct h_i if $w \in \mathrm{PQ}$. Observe that we use index notations for block indices and bracket notations for stream positions.

Algorithm 1 uses memory space $\mathrm{O}(r)$, where r is the number of valleys in w. We could make it run with memory space $\mathrm{O}(\sqrt{N \log N})$ by reducing the number of valleys as in [7]. We do not need to as we use another compression in the two-pass algorithm.

We first state a crucial property of Algorithm 1, and then show that it satisfies Theorem 15, when there is no duplicate. We remind that we process the stream from right to left.

Algorithm 1 One-reverse-pass algorithm for PQ

```
1  m_0 ← −∞; h_0 ← 0; t ← N; i ← 0 // i is called the block index
2  While t > 0
3      If w[t] = ins(a)
4          k ← max{j ≤ i : m_j ≤ a}; //Compute the hashcode index of a
5          Update(h_k, w[t])
6      Else w[t] = ext(a)
7          If w[t+1] = ins(b) //This is a valley. We start a new block
8              i ← i+1; m_i ← a; h_i ← 0 //Create a new hashcode
9          Else w[t+1] = ext(b)
10             Check(a ≥ b)  //Check that extractions are well-ordered
11         Update(h_i, w[t])
12     t ← t − 1
13 For j = 0 to i: Check(h_j = 0) //Check that hashcodes are balanced w.h.p.
14 Accept   // w succeeded to all checks
```

▶ **Lemma 14.** *Consider Algorithm 1 right after processing* $\texttt{ins}(a)$*. Assume that* $\texttt{ext}(a)$ *has been already processed. Let* $h_k, h_{k'}$ *be the respective hashcodes including* $\texttt{ext}(a), \texttt{ins}(a)$*. Then* $k = k'$ *if and only if all* $\texttt{ext}(b)$ *occurring between* $\texttt{ext}(a)$ *and* $\texttt{ins}(a)$ *satisfy* $b > a$*.*

▶ **Theorem 15.** *There is a* 1*-reverse-pass randomized streaming algorithm for* $\mathrm{PQ}(U)$ *with memory space* $\mathrm{O}(r(\log N + \log U))$ *and one-sided bounded error* N^{-c}*, for inputs of length* N *with* r *valleys, and any constant* $c > 0$*.*

Proof. We show that Algorithm 1 suits the conditions, assuming there is no duplicate. Let $w \in \mathrm{PQ}(U)$. Then w always passes the test at line 10. Moreover, by Lemma 14, each insertion $\texttt{ins}(a)$ is necessarily in the same hashcode than its matching extraction $\texttt{ext}(a)$. Therefore, all hashcodes equal 0 at line 13 since they are balanced. In conclusion, the algorithm accepts w with probability 1.

Assume now that $w \notin \mathrm{PQ}$. First we show that unbalanced w are rejected with high probability, that is at least $1 - N^{-c}$, at line 13, if they are not rejected before. Indeed, since each $w[t]$ is encoded in some h_j, at least one h_j must be unbalanced. Then by Fact 13, the algorithm rejects w.h.p. We end the proof assuming w balanced. We remind that we process the stream from right to left. The two remaining possible errors are: (1) $\texttt{ins}(a)$ is processed before $\texttt{ext}(a)$, for some a; and (2) $\texttt{ext}(a), \texttt{ext}(b), \texttt{ins}(a)$ are processed in this order with $b < a$ and possibly intermediate insertions/extractions. In both cases, we show that some hashcodes are unbalanced at line 13, and therefore fail the test w.h.p by Fact 13, except if the algorithm rejects before.

Consider case (1). Since $\texttt{ins}(a)$ is processed before $\texttt{ext}(a)$, there is at least one valley between $\texttt{ins}(a)$ and $\texttt{ext}(a)$. Therefore $\texttt{ins}(a)$ and $\texttt{ext}(a)$ are encoded into different hashcodes, that are unbalanced at line 13. Consider now case (2). Lemma 14 gives that $\texttt{ext}(a)$ and $\texttt{ins}(a)$ are encoded in different hashcodes, that are again unbalanced at line 13. ◀

4.2 Bidirectional two-pass algorithm

Our algorithm performs one pass in each direction using Algorithms 2 and 2. We use the hierarchical data structure of [15] in order to reduce the number of blocks. A block of size 2^i is of the form $[(q-1)2^i + 1, q2^i]$, for $1 \leq q \leq N/2^i$. Observe that, given two such blocks, either they are disjoint or one is included in the other. We decompose dynamically the letters of w, that have been already processed, into nested blocks of 2^i letters as follows. Each new

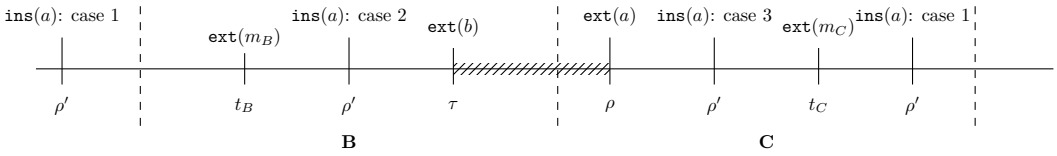

Figure 2 Relative positions of insertions and extractions used in the proof of Theorem 4

Algorithm 2 Pass from left to right

```
1  S ← [(0, −∞, 0)] // Initialization of S
2  While stream is not empty
3      Read(next letter v on stream) // See below
4      While the 2 topmost elements of S have same block size ℓ
5          (h1, m1, ℓ) ←Pop(S); (h2, m2, ℓ) ←Pop(S)
6          Push(S,(h1 + h2 mod p, min(m1, m2), 2ℓ)) // Merge of 2 blocks
7  Check(S = [(0, −∞, 0), (0, 0, N)])
8  Return
9
10 Function Read(v):
11 Case v = ins(a) // When reading an insertion
12     Let (h, m, ℓ) be the first item of S from top such that a ≥ m
13     Replace (h, m, ℓ) by (Update(h, v), m, ℓ)
14     Push (S, (0, +∞, 1))
15 Case v = ext(a) // When reading an extraction
16     For all items (h, m, ℓ) on S such that m > a: Check(h = 0)
17     Let (h, m, ℓ) be the first item of S from top such that a > m
18     Replace (h, m, ℓ) by (Update(h, v), m, ℓ)
19     Push(S,(0, a, 1))
```

processed letter of w defines a new block. When two blocks have same size, they merge. All processed blocks are pushed on a stack. Therefore, only the two topmost blocks of the stack may potentially merge. Because the size of each block is a power of 2 and at most two blocks have the same size (before merging), there are at most $\log N + 1$ blocks at any time.

Moreover, since our stream size is a power of 2, all blocks eventually appear in the hierarchical decomposition, whether we read the stream from left to right or from right to left. In fact, if two same-sized blocks appear simultaneously in one decomposition before merging, the same is true in the other decomposition. This point is crucial for our analysis.

Our algorithm uses the following description of a block B: its hashcode h_B, the minimum m_B of its extractions, and its size ℓ_B. For the analysis, let t_B be such that $w[t_B] = \texttt{ext}(m_B)$. Only h_B can change without B being merged with another block. On the pass from right to left, all extractions from the block and matching insertions are included in h_B. On the pass from left to right, insertions are included in the hashcode of the earliest possible block where they could have been, and extractions are included with their matching insertions. The minimums $(m_B)_B$ are used to decide where to include values. Observe the importance of checking $h_B = 0$ during the execution and not at the end, when only one block is left.

When there is some ambiguity, we denote by $h_B^{\rightarrow}$ and $h_B^{\leftarrow}$ the hashcodes for the left-to-right and right-to-left passes. Observe that m_B, t_B, ℓ_B are identical in both directions.

Proof of Theorem 4. We show that execution of both Algorithms 2 and 3 suits the conditions, assuming no duplicates. The space constraints are satisfied because elements of S have size $\mathrm{O}(\log N + \log U)$ and S has size $\mathrm{O}(\log N)$. The processing time is from inspection.

Algorithm 3 Pass from right to left

```
1  S ← []; // Initialization of S
2  While stream is not empty
3      Read(next letter v on stream) // See below
4      While the 2 topmost elements of S have same block size ℓ
5          (h1, m1, ℓ) ←Pop(S); (h2, m2, ℓ) ←Pop(S)
6          Push(S,(h1 + h2 mod p, min(m1, m2), 2ℓ)) // Merge of 2 blocks
7  Check(S = [(0, 0, N)])}
8  Return
9
10 Function Read(v):
11 Case v = ins(a) // When reading an insertion
12     Let (h, m, ℓ) be the first item of S from top such that a ≥ m
13     Replace (h, m, ℓ) by (Update(h, v), m, ℓ)
14     Push (S, (0, +∞, 1))
15 Case v = ext(a) // When reading an extraction
16     For all items (h, m, ℓ) on S such that m > a: Check(h = 0)
17     Push(S,(hash(v), a, 1))
```

As with Theorem 15, inputs in $\mathrm{PQ}(U)$ are accepted with probability 1, and unbalanced inputs are rejected with high probability (at least $1 - N^{-c}$). Let $w \notin \mathrm{PQ}$ be balanced. For ease of notations, let $w[-1] = \mathtt{ins}(-\infty)$ and $w[0] = \mathtt{ext}(-\infty)$. Then, there are $\tau < \rho$ such that $w[\tau] = \mathtt{ext}(b)$, $w[\rho] = \mathtt{ext}(a)$, with $a > b$ and $w[t] \neq \mathtt{ins}(a)$ for all $\tau < t < \rho$.

Among the pairs (τ, ρ), consider the ones with the smallest ρ. From those, select the one with the smallest b, with $w[\tau] = \mathtt{ext}(b)$. Let B, C be the largest possible disjoint blocks such that τ is in B and ρ in C. Then B and C have same size, are contiguous, and appear simultaneously in each direction before they merge. Let ρ' and τ' be such that $w[\rho'] = \mathtt{ins}(a)$ and $w[\tau'] = \mathtt{ins}(b)$. Then $w[t]$ is an insertion for all $\tau < t < \rho$ by minimality of ρ and b. Indeed if $w[t] = \mathtt{ext}(c)$ either $b > c$, contradicting the minimality of b, or $c > b$ and (τ, t), contradicting the minimality of ρ. In particular, $t_C \geq \rho$ and $t_B \leq \tau$. Similarly, $\tau' < \tau$.

We distinguish three cases based on the position ρ' of $\mathtt{ins}(a)$ (see Figure 2): $\rho' \notin [t_B, t_C]$, $t_B < \rho' < \tau$, and $\rho < \rho' < t_C$. These cases determine in which hashcode $\mathtt{ins}(a)$ is included. We analyze Algorithms 2 and 3 when some letter is processed before blocks potentially merge.

Case 1: $\rho' \notin [t_B, t_C]$. Then $h_B^{\rightarrow}$ and $h_C^{\leftarrow}$ are unbalanced respectively when $w[t_C]$ and $w[t_B]$ are processed; therefore w.h.p. Algorithm 2 detects $h_B^{\rightarrow} \neq 0$ or Algorithm 3 detects $h_C^{\leftarrow} \neq 0$, depending on whether $m_B > m_C$. The full proof is omitted because of space constraints.

Case 2: $t_B < \rho' < \tau$. We show that when Algorithm 3 processes $w[t_B] = \mathtt{ext}(m_B)$, it checks $h_D^{\leftarrow} = 0$ at line 16 for some $h_D^{\leftarrow}$ including $\mathtt{ins}(a)$ but not $\mathtt{ext}(a)$. Thus it rejects w.h.p.

When $w[\rho'] = \mathtt{ins}(a)$ is processed on the right-to-left pass, $\tau \in B_1$ with B_1 a block in the stack. $\tau \in B$, therefore B_1 intersects B. Because $B_1 \not\subseteq B$, we have $B_1 \subseteq B$. Because $w[\tau] = \mathtt{ext}(b)$, we have $a > b \geq m_{B_1}$, and block B_1 is eligible at line 12 of Algorithm 3, meaning that $w[\rho'] = \mathtt{ins}(a)$ is included in either $h_{B_1}^{\leftarrow}$ or a more recent hashcode $h_{B_2}^{\leftarrow}$. Since $\rho' \in B$, again $B_2 \subseteq B$. Last, when Algorithm 3 processes $w[t_B] = \mathtt{ext}(m_B)$, since we are still within B, some hashcode h_{B_3}, with $B_3 \subseteq B$, includes $w[\rho']$. Moreover, $h_{B_3}^{\leftarrow}$ does not include $w[\rho] = \mathtt{ext}(a)$ since $\rho \in C$ and C comes before B. Last, $m_{B_3} > m_B$, by definition of m_B. Hence, Algorithm 3 checks $h_{B_3}^{\leftarrow} = 0$ at line 16 when processing $w[t_B]$. B_3 satisfies the conditions for D when $w[t_B]$ is processed, and Algorithm 3 rejects w.h.p.

Case 3: $\rho < \rho' < t_C$. The proof is the same as case 2, replacing τ, B, B_1, etc.. with ρ, C, C_1, etc... and Algorithm 2 with Algorithm 3. Note that we only have $a \geq m_{C_1}$ this time. ◀

4.3 Generalization when duplicates occur

We maintain two additional parameters δ_B and C_B for each block B. The difference between the number of insertions and extractions included in h_B is stored in δ_B. Whenever $\delta_B = 0$, we check $h_B = 0$. The number of unmatched occurrences of $\texttt{ins}(m_B)$ for the left-to-right pass (resp. $\texttt{ext}(m_B)$ for the right-to-left pass) is stored in C_B. We can then appropriately determine whether each $\texttt{ext}(m_B)$ (resp. $\texttt{ins}(m_B)$) should be included in h_B.

The inequality at line 12 of Algorithm 2 has to become strict instead of large, which the proof of case 3 of the theorem longer and breaks the symmetry.

References

1 N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

2 Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004.

3 M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2):225–244, 1994.

4 M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

5 M. Braverman. Interactive information complexity. In *Proc. of ACM Symp. on Theory of Computing*, pages 505–524, 2012.

6 M. Braverman and A. Rao. Information equals amortized communication. In 748-757, editor, *Proc. of IEEE Symp. on Foundations of Computer Science*, 2011.

7 A. Chakrabarti, G. Cormode, R. Kondapally, and A. McGregor. Information cost tradeoffs for augmented index and streaming language recognition. In *Proc. of IEEE Symp. on Foundations of Computer Science*, pages 387–396, 2010.

8 A. Chakrabarti, Y. Shi, A. Wirth, and A. C.-C. Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *Proc. of IEEE Symp. on Foundations of Computer Science*, pages 270–278, 2001.

9 M. Chu, S. Kannan, and A. McGregor. Checking and spot-checking the correctness of priority queues. In *Proc. of Int. Colloquium on Automata, Languages and Programming*, pages 728–739, 2007.

10 R. Jain and A. Nayak. The space complexity of recognizing well-parenthesized expressions in the streaming model: the index function revisited, 2010. ECCC Tech. Rep. TR10-071.

11 R. Jain, J. Radhakrishnan, and P. Sen. A lower bound for the bounded round quantum communication complexity of Set Disjointness. In *Proc. of IEEE Symp. on Foundations of Computer Science*, pages 220–229, 2003.

12 T. S. Jayram, Ravi Kumar, and D.Sivakumar. Two applications of information complexity. In *Proc. of ACM Symp. on Theory of Computing*, pages 673–682, 2003.

13 I. Kerenidis, S. Laplante, V. Lerays, J. Roland, and D. Xiao. Lower bounds on information complexity via zero-communication protocols and applications. In *Proc. of IEEE Symp. on Foundations of Computer Science*, 2012. To appear.

14 C. Konrad and F. Magniez. Validating XML documents in the streaming model with external memory. In *Proc. of Int. Conf. on Database Theory*, pages 34–45, 2012.

15 F. Magniez, C. Mathieu, and A. Nayak. Recognizing well-parenthesized expressions in the streaming model. In *Proc. of ACM Symp. on Theory of Computing*, pages 261–270, 2010.

16 S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers Inc., 2005.

17 M. Saks and X. Sun. Space lower bounds for distance approximation in the data stream model. In *Proc. of ACM Symp. on Theory of Computing*, pages 360–369, 2002.