Unidirectional Input/Output Streaming Complexity of Reversal and Sorting*

Nathanaël François[1], Rahul Jain[2], and Frédéric Magniez[3]

1 **Univ Paris Diderot, Sorbonne Paris-Cité, LIAFA, CNRS, 75205 Paris, France**
nathanael.francois@liafa.univ-paris-diderot.fr
2 **CQT and CS Department, National University of Singapore**
rahul@comp.nus.edu.sg
3 **CNRS, LIAFA, Univ Paris Diderot, Sorbonne Paris-Cité, 75205 Paris, France**
frederic.magniez@cnrs.fr

Abstract

We consider unidirectional data streams with restricted access, such as read-only and write-only streams. For read-write streams, we also introduce a new complexity measure called expansion, the ratio between the space used on the stream and the input size.

We give tight bounds for the complexity of reversing a stream of length n in several of the possible models. In the read-only and write-only model, we show that p-pass algorithms need memory space $\Theta(n/p)$. But if either the output stream or the input stream is read-write, then the complexity falls to $\Theta(n/p^2)$. It becomes polylog(n) if $p = \mathrm{O}(\log n)$ and both streams are read-write.

We also study the complexity of sorting a stream and give two algorithms with small expansion. Our main sorting algorithm is randomized and has $\mathrm{O}(1)$ expansion, $\mathrm{O}(\log n)$ passes and $\mathrm{O}(\log n)$ memory.

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

Keywords and phrases Streaming Algorithms, Multiple Streams, Reversal, Sorting

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Background and Motivations. Streaming algorithms have been studied for estimating statistics, checking properties and computing functions (more often with sublinear outputs) on massive inputs for several years. However, less is known on computing functions that also have a massive output, and therefore require two data streams, one for the input and one for the output. The notion of reversal complexity on a multi-tape machine, which can be related to streaming complexity with multiple streams, was first introduced in 1970 by Kameda and Vollmar [12] for decision problems. The model however was not explored again until 1991 when Chen and Yap [5] considered the computable functions in this model, and gave an algorithm for sorting using two streams with $\mathrm{O}(\log n)$ passes and (internal) memory space of size $\mathrm{O}(\log n)$. This was a significant improvement over the lower bound for the case of a single stream which requires $\Omega(n/s)$ passes, where s is the size of the memory space, as proved by Munro and Paterson [15].

* Partially supported by the French ANR Blanc project ANR-12-BS02-005 (RDAM), the Singapore Ministry of Education Tier 3 Grant and also the Core Grants of the Centre for Quantum Technologies, Singapore.

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–15

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

Recently the interest for complexity in models with multiple streams has been renewed. Indeed, streams are now considered as a model of external storage allowing multiple sequential passes on them. Hernich and Schweikardt [10] gave reductions from classical complexity classes to the classes of problems decidable with $O(1)$ streams. Grohe, Koch, Hernich and Scwheikardt [9, 8] also gave several tight lower bounds for multi-stream algorithms using $o(\log n)$ passes. Gagie [7] showed that two streams achieve perfect compression in polylogarithmic memory space and a polylogarithmic number of passes. Beame, Huynh, Jayram and Rudra [3, 2] also proved several lower bounds on approximating frequency moments with multiple streams and $o(\log n)$ passes. However, there is a distinct lack of lower bounds for multi-stream algorithms with $\Omega(\log n)$ passes. Indeed, most classical tools (such as reduction to communication complexity) for proving lower bounds on streaming algorithms fail for multiple streams. One exception is Ruhl's [17] W-Stream and StreamSort models. In W-Stream each (unidirectional) stream is alternatively read-only and write-only, and passes are synchronized; while this model seems similar to ours, it is actually much less powerful and most lower bounds match the naive linear algorithms. StreamSort is the W-Stream model augmented with a sorting oracle and yields more interesting results, but of course trivializes the problem we study in this paper. Indeed, most classical tools (such as reduction to communication complexity) for proving lower bounds on streaming algorithms fail for multiple streams.

We therefore take an opposite approach and restrict the number of streams to two, the *input stream* and the *output stream*. This leads to our notion of *input/output streaming algorithms*. For many scenarios, this model is in fact more realistic since, as an external storage, it comes at a cost to use several streams simultaneously. In fact, the idea of using only input and output and constrained secondary storage can also be found in other models, such as [1]. Additionally, we only allow passes in one direction on both streams. This may looks too restrictive since bidirectional passes provide exponential speedup in several decision problems, as exhibited by several works [14, 11, 4, 13, 6]. However, changing the direction of processing a stream can have a higher cost: most hard drives spin in only one direction and reading them in the other is not as practical. We also consider three types of stream accesses: *read-only* (for the input stream), *write-only* (for the output stream) and *read-write* (for any or both streams). We emphasise that by write-only stream we mean that once the algorithm has written in a cell, it can never overwrite it.

For read-write streams, we introduce the new complexity measure of *expansion* of a streaming algorithm, i.e. the ratio between the maximal size of the stream during the computation and the size of the input (this concept does not make sense for read-only and write-only stream, which cannot be used for intermediate computations). While space on external storage is by definition not as constrained as memory space, in the case of massive inputs requiring large hard drives, expanding them even by a factor $O(\log n)$ is not reasonable. To the best of our knowledge, while it has sometimes been implicitely limited to 1 or $O(1)$, this measure of complexity has never been studied before for potential trade-offs with space or time.

We then study two problems, Reverse and Sort, in the model of input/output streaming algorithms. Both problems consists in copying the content of the input stream to the output stream, in reverse order for the first problem, and in sorted order for the second one. Because of its importance in real applications, Sort has been extensively studied in many contexts. However, there is still a lot to say about this problem in face of new models of computation that massive data arise. Reverse can be seen as the simplest instance of Sort. It is natural to study its complexity since we forbid any reverse pass on streams. Moreover, any algorithm

for Reverse gives a way to implement each of the efficient bidirectional (and single stream) algorithms of [14, 13, 6] in our model, thus giving another example of a speed-up with multiple streams.

Our results. In Section 3, we give tight bounds for the complexity of Reverse. We provide deterministic algorithms that are optimal even against randomized ones. Using communication complexity to prove lower bounds with multiple streams is inherently difficult as soon as there are multiple passes on both. Indeed, it is possible to copy one stream on the other one, and since heads move separately, the algorithm can access at any part of the second stream while processing the first one. In the read-only/write-only model, we instead skip communication complexity and prove directly with information theory that a p-pass algorithm needs space $\Omega(n/p)$ (Theorem 3). In the read-only/read-write model, we similarly show that any p-pass algorithm needs space $\Omega(n/p^2)$ (Theorem 10). We provide an algorithm achieving that bound by copying large (unreversed) blocks from the input, and then placing correctly one element from each of the blocks during each pass (Theorem 8). Similar results hold in the read-write/write-only model (Theorems 9 and 12). Last, we consider the read-write/read-write model and give a $\mathrm{O}(\log n)$-pass algorithm with polylogarithmic memory space (Theorem 13). All our algorithms presented make extensive use of the ability to only write part of a stream during a pass. In the more restrictive W-Stream model where this is not possible, as shown by [17], a p-pass algorithm requires memory $\Omega(n/p)$.

In Section 4, we consider problem Sort. Even with two streams only, it appears to have a tight bound, as the algorithm by Chen and Yap [5] matches the lower bound proved by Grohe, Hernich and Scwheikardt [8]. However, Chen and Yap's algorithm was not designed with expansion in mind: it works by replicating the input n times, thus using $\mathrm{O}(n^2)$ cells on the input and output streams. In this context, linear expansion of the input is not reasonable. We therefore give two algorithms with two streams that improve on it. The first one is deterministic. Similarly to the former algorithm, it is based on Merge Sort, but has expansion $\mathrm{O}(\log n)$ instead of $\Omega(n)$. It also uses $\mathrm{O}(\log n)$ passes and space $\mathrm{O}(\log n)$ (Theorem 14). The second one is randomized and based on Quick Sort. It has expansion $\mathrm{O}(1)$, $\mathrm{O}(\log n)$ passes and memory $\mathrm{O}(\log n)$ (Theorem 15).

2 Preliminaries

In streaming algorithms (see [16] for an introduction), a *pass* on a string $w \in \Sigma^n$, for some finite alphabet Σ, means that w is given as a *stream* $w[1], w[2], \ldots, w[n]$, which arrives sequentially, i.e., letter by letter in this order. Depending on the model, w may or may not be modified. For simplicity, we always assume $|\Sigma| = \mathrm{O}(n)$, except when explicitly stated. Otherwise, one should transpose our algorithms such that the letters are also read sequentially, say bit by bit. We now fix in the rest of the paper such a finite alphabet Σ equipped with a total order.

We consider *input/output streaming algorithms* with two data streams X and Y. Stream X initially contains the input, and stream Y is initially empty and will contain the output at the end of the execution of the algorithm. We denote by $X[i]$ the i-th cell of stream X, and similarly for Y. The size of a stream is the number of its cells containing some data. Our algorithms have also access to a random access memory M, usually of sublinear size in the input size.

We then parameterize such algorithms by the operations allowed on the input stream X (read-only or read-write), on the output stream Y (write-only or read-write), the number of passes, the bit-size of the memory M and the expansion of the streams with regard to the size

of the input. Whenever we mention the memory of an algorithm, we mean a random-access memory.

▶ **Definition 1** (Input/Output streaming algorithm)**.** Let I be either RO or RW, and let O be either WO or RW. Then a $p(n)$-*pass I/O streaming algorithm* with space $s(n)$ is an algorithm that, given $w \in \Sigma^n$ as a stream X, produces its output on an initially empty stream Y and such that:

- It performs $p(n)$ sequential passes in total on X and Y;
- It maintains a memory M of size at most $s(n)$ bits while processing X, Y;
- If I is RO, X cannot be modified;
- If O is WO, Y cannot be read and each cell of Y can be written only once.

Moreover, the algorithm has expansion $\lambda(n)$ if streams X, Y have length (in number of cells) at most $\lambda(n) \times n$ during its execution, for all input $w \in \Sigma^n$.

Observe that we do not mention any running time in our definition. Indeed, all our lower bounds will be stated independently of it. Moreover, all our algorithms process each letter from each stream in polylogarithmic time. They also have also polylogarithmic preprocessing and postprocessing times.

A usual way to get an algorithm with expansion > 1 consists in annotating streams with a larger alphabet Σ', while keeping (artificially) the same number of annotated cells. In that case, one can simulate one annotated cell using $(\log|\Sigma'|)/\log|\Sigma|$ non-annotated cells.

For simplicity, we assume further 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. Moreover, this assumption only makes our lower bounds stronger.

We will use the two usual notions of randomized computing.

Monte Carlo: An algorithm computes a function f on Σ^n *with error* $\varepsilon \leq 1/3$ if for all inputs $w \in \Sigma^n$, it outputs $f(x)$ with probability at least $1 - \varepsilon$.

Las Vegas: An algorithm computes a function f on Σ^n *with failure* $\varepsilon \leq 1/2$ if for all inputs $w \in \Sigma^n$, it outputs $f(x)$ with probability at least $1 - \varepsilon$, otherwise it gives no output.

The two functions we study in this paper in term of streaming complexity are now formally defined.

▶ **Definition 2** (Reverse and Sort)**.** For $w = w[1]w[2]\dots w[n] \in \Sigma^n$, let us define Reverse$(w)$ as $w[n]w[n-1]\dots w[1]$. When Σ has a total order, also define Sort(w) as the sorted permutation of w.

For simplicity when proving lower bounds, for the proofs of Theorems 3, 10 and 12, instead of considering algorithms for Reverse writing $Reverse(X)$ on Y with passes from left to right, we will have them write X on Y with passes from right to left, which is equivalent with a relabeling of Y. The algorithms presented for upper bounds, however, read better in the model where we write Reverse(X) on Y using a pass from left tor right.

When we describe a streaming algorithm using pseudo-code, a 'For' loop will always correspond to a single pass on each stream, except when explicitly mentioned otherwise. Most algorithms will consist of a 'While' loop including a constant number of 'For' loops, i.e. passes. Therefore the number of iterations of the 'While' loop will give us the number of passes.

Throughout the paper, we always give randomized lower bounds. While communication complexity arguments fail in the context of mutliple streams, we use information theory arguments. They have recently been proved to be a powerful tool for proving lower bounds

in communication complexity, and therefore for streaming algorithms. Here we not only use them directly, but also there is no direct interpretation in term of communication complexity.

Let us remind now the notions of entropy H and mutual information I. Let X, Y, Z be three random variables. Then :

$$\begin{aligned} \mathrm{H}(X) &= -\mathbb{E}_{x\leftarrow X} \log \Pr(X = x), \\ \mathrm{H}(X|Y = y) &= -\mathbb{E}_{x\leftarrow X} \log \Pr(X = x|Y = y), \\ \mathrm{H}(X|Y) &= \mathbb{E}_{y\leftarrow Y} \mathrm{H}(X|Y = y), \\ \mathrm{I}(X : Y|Z) &= \mathrm{H}(X|Z) - \mathrm{H}(X|Y, Z). \end{aligned}$$

The entropy and the mutual information are non negative and satisfy $\mathrm{I}(X : Y|Z) = \mathrm{I}(Y : X|Z)$. For $0 \leq \varepsilon \leq 1$, we denote by $\mathrm{H}(\varepsilon)$ the entropy of the Bernoulli variable that takes value 1 with probability ε, and 0 with probability $1 - \varepsilon$.

3 Reversal with two streams

3.1 Complexity in the Read-only/Write-only model

When $\Sigma = \{0, 1\}$, the naive algorithm, that copies at each pass s bits of the input in memory and then to the output in reverse order, requires memory space $s = n/p$ when performing p passes. When processing each stream for the first time, it is in fact obvious that $Y[i]$ cannot be written until $X[n - i + 1]$ has been read. In particular, for $p = 2$ we have $s \geq n$ for any algorithm. With multiple passes on each stream, the proof gets more technical especially since the point where the streams cross, i.e. where we can write on the second stream what was read on the first one during the current pass depends on the pass, the input and the randomness. However the $s \geq \mathrm{O}(n/p)$ bound still applies for all p, when the error probability of the algorithm is $\mathrm{O}(1/p)$.

Observe that a constant error probability can be reduced to $\mathrm{O}(1/p)$ using $\mathrm{O}(\log p)$ parallel repetitions, leading to an $\mathrm{O}(\log p)$ factor in the size of the memory space.

▶ **Theorem 3.** *Let $0 \leq \varepsilon \leq 1/10$. Every p-pass randomized RO/WO algorithm for* Reverse *on $\{0, 1\}^n$ with error ε requires space $\Omega(n/p)$.*

Expansion is not mentionned in this theorem as it does not make sense on streams that are not read-write: the algorithm either cannot modify the stream if it is read-only, or has no use for the extra data if it is write-only.

Before proving Theorem 3, we begin with two useful facts.

▶ **Fact 4.** *Let X be uniformly distributed over $\{0, 1\}^n$, and let J be some random variable on $\{0, 1, \ldots, n\}$ that may depend on X. Then :*

$$\mathrm{H}(X[1, J]|J) \leq \mathbb{E}(J) \text{ and } \mathrm{H}(X[1, J]|X[J + 1, n]) \geq \mathbb{E}(J) - \mathrm{H}(J).$$

Similarly,

$$\mathrm{H}(X[J + 1, n]|J) \leq n - \mathbb{E}(J) \text{ and } \mathrm{H}(X[J + 1, n]|X[1, J]) \geq n - \mathbb{E}(J) - \mathrm{H}(J).$$

Proof. $\mathrm{H}(X[1, J]|J) \leq \mathbb{E}(J)$ and $\mathrm{H}(X[J + 1, n]|J) \leq n - \mathbb{E}(J)$ are direct. The second part uses the first one as follows:

$$\begin{aligned} \mathrm{H}(X[1, J]|X[J + 1, n]) &= \mathrm{H}(X|J, X[J + 1, n]) - \mathrm{H}(X[J + 1, n]|J, X[J + 1, n]) \\ &= \mathrm{H}(X|J) - \mathrm{H}(X[J + 1, n]|J) \\ &\geq \mathrm{H}(X) - \mathrm{H}(J) - n + \mathbb{E}(J) = \mathbb{E}(J) - \mathrm{H}(J). \end{aligned}$$

◀

▶ **Fact 5** (Data processing inequality). *Let X, Y, Z, R be random variables such that R is independent from X, Y, Z. Then for every function f*

$$\mathrm{H}(X|Y,Z) \leq \mathrm{H}(f(X,R)|Y,Z) \quad \textit{and} \quad \mathrm{I}(X:Y|Z) \geq \mathrm{I}(f(X,R):Y|Z).$$

Note that the previous property is usually stated with no variable R, then f is defined as a *probabilistic function*.

Proof of Theorem 3. In this proof, we use the equivalent model of the algorithm copying X (and not Reverse(X)) on Y, but processing Y only with passes from right to left. This means that the two heads start on opposite ends and meet once each pass.

Consider a p-pass randomized RO/WO algorithm for Reverse on $\{0,1\}^n$ with error ε and space $s \geq \log n$. For simplicity, we assume that passes are synchronized : whenever a pass on one stream ends, the head on the other stream ends its own pass, and then eventually moves back to its original position. This costs us at most a factor 2 in the number of passes.

Let the input stream X be uniformly distributed in $\{0,1\}^n$. For each pass $1 \leq t \leq p$, let $Z^t \in \{0,1,\perp\}^n$ be the reverse of the data written on the output stream Y: if nothing is written at pass t and index i, then $Z^t[i] = \perp$. This model corresponds to writing the same string on the output stream as on the input stream, but going in different directions, an makes the notations simpler as we want $X[i] = Z^t[i]$. Note that because the algorithm cannot overwrite a letter, for each i there is at most one t such that $Z^t[i] \neq \perp$. Last, let L^t be the index where the reading head and the writing head meet during pass t. Since passes are synchronized, L^t is unique (but possibly depends on the input and random choices). For $1 \leq i \leq n$, $1 \leq t \leq p$, let M_i^t be the state of the memory after the algorithm reads $X[i]$ on pass t.

For $1 \leq t \leq p$, since s bounds the size of memory, we have :

$$s \geq \mathrm{I}(X[1,L^t] : M_{L^t}^t | X[L^t+1,n])) \text{ and } s \geq \mathrm{I}(X[L^t+1,n] : M_n^{t+1} | X[1,L^t])).$$

Using the definition of mutual information, we get the following inequalities:

$$s \geq \mathrm{H}(X[1,L^t]|X[L^t+1,n]) - \mathrm{H}(X[1,L^t]|M_{L^t}^t, X[L^t+1,n]),$$

$$s \geq \mathrm{H}(X[L^t+1,n]|X[1,L^t]) - \mathrm{H}(X[L^t+1,n]|M_n^{t-1}, X[1,L^t]).$$

We define the following probabilities : $q_i^t(l) = \Pr(Z^t[i] \neq \perp | L^t = l)$, $q_i^t = \Pr(Z^t[i] \neq \perp)$, $\varepsilon_i^t(l) = \Pr(Z^t[i] \neq \perp, Z^t[i] \neq X[i] | L^t = l)$ and $\varepsilon_i^t = \Pr(Z^t[i] \neq \perp, Z^t[i] \neq X[i])$. By definition, they also satisfy $\varepsilon_i^t = \mathbb{E}_{l \sim L^t}(\varepsilon_i^t(l))$ and $q_i^t = \mathbb{E}_{l \sim L^t}(q_i^t(l))$. Note that by hypothesis[1] and because there is no rewriting, $\sum_{t=1}^p \varepsilon_i^t \leq \Pr(\exists t, Z^t[i] \neq \perp, Z^t[i] \neq X[i]) \leq \varepsilon$. Lemmas 6 and 7 give us these inequalities:

$$2s \geq n - \sum_{i=1}^{n} \mathrm{H}(X[i]|Z^t[i], L^t) - O(\log n),$$

$$\mathrm{H}(X[i]|Z^t[i], L^t) \leq 1 - q_i^t \left(1 - \mathrm{H}(\varepsilon_i^t/q_i^t)\right).$$

Combining them yields :

$$2s \geq \sum_{i=1}^{n} q_i^t \left(1 - \mathrm{H}\left((\varepsilon_i^t/q_i^t)\right)\right) - O(\log n).$$

[1] Here we only need the hypothesis that each bit of Y is wrong with probability at most ε, and not the stronger hypothesis that $Y \neq \text{Reverse}(X)$ with probability at most ε.

Let $\alpha_i = \sum_{t=1}^{p} q_i^t$. Then $\alpha_i = \Pr[Y[i] \neq \perp]$ satisfies $\alpha_i \geq 1 - \varepsilon$ by hypothesis. Now summing over all passes leads to $2ps \geq \sum_{i=1}^{n} \alpha_i \sum_{t=1}^{p} (q_i^t/\alpha_i)(1 - \mathrm{H}(\varepsilon_i^t/q_i^t)) - \mathrm{O}(p \log n)$.

The concavity of H gives us $\sum_{t=1}^{p} (q_i^t/\alpha_i)\mathrm{H}(\varepsilon_i^t/q_i^t) \leq \mathrm{H}(\varepsilon/(1-\varepsilon))$. This means, replacing α_i and ε_i^t by their upper bounds, that $2ps \geq n(1-\varepsilon)(1 - \mathrm{H}(\varepsilon/(1-\varepsilon))) - \mathrm{O}(p \log n)$. Since Theorem 3 has $\varepsilon \leq 0.1$ as an hypothesis, our algorithm verifies $ps \geq \Omega(n)$. ◀

▶ **Lemma 6.** *Assuming the hypotheses of Theorem 3, at any given pass t,*

$$2s \geq n - \sum_{i=1}^{n} \mathrm{H}(X[i]|Z^t[i], L) - O(\log n).$$

Proof. In this proof, we write $Z[i]$ for $Z^t[i]$ since there is generally no ambiguity. We similarly omit the t on other notations.

The data processing inequality (Fact 5) gives us the following inequality : $\mathrm{H}(X[1,L]|M_L, X[L+1,n], L) \leq \mathrm{H}(X[1,L]|Z[1,L], L)$. We can rewrite this as

$$\mathrm{H}(X[1,L]|M_L, X[L+1,n], L) \leq \mathbb{E}_{l \sim L}(\mathrm{H}(X[1,l]|Z[1,l], L = l)).$$

Using the chain rule and removing conditioning, we get

$$\mathrm{H}(X[1,L]|M_L, X[L+1,n], L) \leq \mathbb{E}_{l \sim L}(\sum_{i=1}^{l} \mathrm{H}(X[i]|Z[i], L = l)).$$

Similarly,

$$\mathrm{H}(X[L+1,n]|M_n^{t-1}, X[1,L], L) \leq \mathbb{E}_{l \sim L}(\sum_{i=1}^{l} \mathrm{H}(X[i]|Z[i], L = l)).$$

Using that both M_L (where $M_L = M_{L^t}^t$) and M_n^{t-1} are of size at most s bits, we get

$$2s \geq \mathrm{I}(X[1,L] : M_{L^t}^t | X[L+1,n]) + \mathrm{I}(X[L+1,n] : M_n^{t-1} | X[1,L]).$$

Then we conclude by combining the above inequalities and using Fact 4 as follows:

$$\begin{aligned}
2s &\geq \mathbb{E}(L) - \mathrm{H}(L) + n - \mathbb{E}(L) - \mathrm{H}(L) \\
&\quad - \mathrm{H}(X[1,L]|M_L^t, X[L+1,n]) - \mathrm{H}(X[L+1,n]|M_n^{t-1}, X[1,L]) \\
&\geq n - \mathbb{E}_{l \sim L}\left(\sum_{i=1}^{l} \mathrm{H}(X[i]|Z[i], L = l) + \sum_{i=l+1}^{n} \mathrm{H}(X[i]|Z[i], L = l)\right) - \mathrm{O}(\log n) \\
&= n - \sum_{i=1}^{n} \mathrm{H}(X[i]|Z[i], L) - O(\log n).
\end{aligned}$$

◀

▶ **Lemma 7.** *Assuming the hypotheses of Theorem 3, for any pass t,* $\mathrm{H}(X[i]|Z^t[i], L_t) \leq 1 - q_i^t(1 - \mathrm{H}(\varepsilon_i^t/q_i^t))$

Proof. As above, we omit the t in the proof, as there is no ambiguity.

The statement has some similarities with Fano's inequality. Due to the specificities of our context, we have to revisit its proof as follows. First we write

$$\begin{aligned}
\mathrm{H}(X[i]|Z[i],L) &\leq \mathbb{E}_{l\sim L}(\mathrm{H}(X[i]|Z[i],L=l)) \\
&\leq \mathbb{E}_{l\sim L}(q_i(l)\mathrm{H}(X[i]|Z[i],L=l,Z[i]\neq\perp) \\
&\quad +(1-q_i(l))\mathrm{H}(X[i]|L=l,Z[i]=\perp) \\
&\leq \mathbb{E}_{l\sim L}\left(q_i(l)\mathrm{H}\left(\frac{\varepsilon_i(l)}{q_i(l)}\right)+1-q_i(l)\right) \\
&= 1-q_i+\mathbb{E}_{l\sim L}\left(q_i(l)\mathrm{H}\left(\frac{\varepsilon_i(l)}{q_i(l)}\right)\right). \qquad (1)
\end{aligned}$$

By replacing the entropy with its definition, we can see that for any $1\geq q\geq\varepsilon>0$, we have $q\mathrm{H}\left(\frac{\varepsilon}{q}\right)=\mathrm{H}(q-\varepsilon,\varepsilon,1-q)-\mathrm{H}(q)$, where $\mathrm{H}(x,y,z)$ is the entropy of a random variable R in $\{0,1,2\}$ with $\Pr(R=0)=x$, $\Pr(R=1)=y$ and $\Pr(R=2)=z$. Let R_i be such that $R_i=0$ if $X[i]=Z[i]$, $R_i=2$ if $Z[i]=\perp$ and $R_i=1$ otherwise. Note that $(Z[i]=\perp)$ is a function of R_i. Therefore:

$$\begin{aligned}
\mathbb{E}_{l\sim L}\left(q_i(l)\mathrm{H}\left(\frac{\varepsilon_i(l)}{q_i(l)}\right)\right) &= \mathbb{E}_{l\sim L}(\mathrm{H}(R_i|L=l)-\mathrm{H}((Z[i]=\perp)|L=l)) \\
&= \mathrm{H}(R_i|L)-\mathrm{H}((Z[i]=\perp)|L) \\
&= \mathrm{H}(R_i)-\mathrm{H}((Z[i]=\perp))+\mathrm{I}((Z[i]=\perp)|L)-\mathrm{I}(R_i|L).
\end{aligned}$$

By the data processing inequality, $\mathrm{I}((Z[i]=\perp):L)\leq\mathrm{I}(R_i:L)$, so

$$\mathbb{E}_{l\sim L}(q_i(l)\mathrm{H}(\varepsilon_i(l)/q_i(l))))\leq q_i\mathrm{H}(\varepsilon_i/q_i)).$$

Combining this with inequality 1 gives us the lemma. ◀

3.2 Complexity of Read-only/Read-write and Read-Write/Write-Only

In this section, we prove tight bounds in the Read-only/Read-Write and Read-Write/Write-Only model. These models are more complex than the previous one since an algorithm may now modify $Y[i]$ or $X[i]$ several times, and use that as additional memory.

Algorithm. As a subroutine we use Algorithm 1, which performs $\mathrm{O}(\sqrt{n})$ passes and uses space $\mathrm{O}(\log|\Sigma|)$. It works by copying blocks of size $\mathrm{O}(\sqrt{n})$ from the input stream to the output stream without reversing them (otherwise there would not be enough space in the memory), but putting them in the correct order pairwise. In addition, during each pass on the output, it moves one element from each block already copied to the correct place. Since blocks have as many elements as there are passes left after they are copied, the output stream is in the correct order at the end of the execution .

▶ **Theorem 8.** *There is a deterministic algorithm such that, given n and $p\leq\sqrt{n}$, it is a p-pass RO/RW streaming algorithm for* Reverse *on Σ^n with space* $\mathrm{O}(\log n+(n\log|\Sigma|)/p^2)$ *and expansion* 1.

Proof. We will prove that Algorithm 1 satisfies the theorem when $p=2\sqrt{n}$. For the general case, the algorithm treats groups of $m=4n/p^2$ letters as though they were just one letter of the new alphabet Σ^m, and then runs Algorithm 1. This uses spaces $\mathrm{O}(\log(n/m)+(n/m)(\log|\Sigma^m|)/p^2)=\mathrm{O}(\log n+(n\log|\Sigma|)/p^2)$.

Algorithm 1 RO/RW streaming algorithm for Reverse

```
1  p ← √(2n), t ← 1, i_1 ← p;
2  While {t ≤ p}
3    If t > 1 then (R, l) ← (Y[n − i_t], n − i_t)
4    If i_t < n then
5      Y[n − i_t − (p − t), n − i_t] ← X[i_t − (p − t), i_t] // Order is unchanged
6      i_{t+1} ← i_t + p − t
7    For m = t − 1 to 1
8      Put R in the right place Y[l'] // l' computed from l, m, p, n
9      (R, l) ← (Y[l'], l')
10   t ← t + 1
```

We now prove the theorem for $p = 2\sqrt{n}$. First, observe that every element of the input is copied on the output, as $\sum_{t=1}^{p}(p-t) = p(p+1)/2 = n + \sqrt{n} \geq n$.

Let $1 \leq t < p$. The subword $X[i_t - (p-t) - 1, i_t]$ is initially copied at line 5 on $Y[n - i_t - (p-t) - 1, n - i_t]$. Therefore only $Y[n - i_t]$ is correctly placed. Let B_t be $\{n - i_t - (p-t), n - i_t - 1\} = \{n - i_{t+1} + 1, n - i_t - 1\}$. Then B_t denotes the indices (on Y) of elements copied during the t-th iteration of the While loop that are incorrectly placed. For each $l \in B_t$, the correct place for $Y[l]$ is in $\{n - i_t + 1, n - i_t + p - t\} = \{n - i_t + 1, n - (i_{t-1} + 1)\} = B_{t-1}$, where by convention B_0 is defined with $i_0 = 0$. Therefore, in the $(t+1)$-th iteration of the while loop, the first $Y[l]$ we place correctly goes from $l = n - i_{t+1} + 1 = n - i_t - (p-t) \in B_t$ to some $l' \in B_{t-1}$. Then recursively the previous value of $Y[l']$ goes in B_{t-2}, and so on until we reach B_0 where nothing was written initially.

Thus, the For loop places correctly one element of each of B_{t-1}, $B_{t-2}, \ldots, B_1$. Observe that B_m has at most $p - m$ elements incorrectly placed. Moreover, there are $(p - m)$ remaining iterations of the While loop after B_m is written. Therfore all elements are places correctly when Algorithm 1 ends.

Now we prove that Algorithm 1 has the claimed complexity. It only starts a new pass when t increases, at each execution of the While loop at line 2. Indeed, each execution of line 8 can be performed within the current pass since $Y[l]$ moves forward to a new index in B_{m-1} as explained before. Therefore Algorithm 1 runs in p passes. Since it keeps at most two elements in memory at any time, and only needs to keep track of the current position, n, p, t and m, it uses $O(\log n + \log |\Sigma|)$ memory. ◀

▶ **Theorem 9.** *There is a deterministic algorithm such that, given n and $p \leq \sqrt{n}$, it is a p-pass RW/WO streaming algorithm for* Reverse *on Σ^n with space $\mathrm{O}(\log n + (n \log |\Sigma|)/p^2)$ and expansion 1.*

We omit the proof of this theorem, as the algorithm is extremely similar to the one used in the Read-Only/Read-Write model. The main difference is that the For loop that moves one element of each block to its correct place is applied before a block is copied on Y and not after like in Algorithm 1. Similarly, the new algorithm starts with blocks of size s and ends with size $\Theta(\sqrt{n/s})$ instead of having blocks of decreasing size.

Lower bound. We employ techniques similar to the ones we used in the proof of Theorem 3. However, here we will not consider individual cells on the stream but instead blocks of size $k = \sqrt{ns}$, where s is the memory space. This allows us to easily bound the amount of information each block receives.

▶ **Theorem 10.** *Let $0 < \varepsilon \leq 1/3$. Every p-pass λ-expansion RO/RW streaming algorithm for* Reverse *on $\{0,1\}^n$ with error ε requires space $\Omega(n/p^2)$.*

Proof. Consider a p-pass λ-expansion randomized RO/RW algorithm for Reverse on $\{0,1\}^n$ with error ε and space s, with $s \geq \log n$. Like with Theorem 3, we consider the model where we want $Y = X$, but Y is processed from right to left.

As in the proof of Theorem 3, we assume passes are synchronized at the cost of a factor at most 2 in p. We also keep similar notations : X is the input stream uniformly distributed in $\{0,1\}^n$, and for each $1 \leq t \leq p$, $Y^t \in \{0,1,\perp\}^n$ is the data currently on output stream Y at pass t. Unlike with Z^t in the previous section, this includes the data previously written, as in this model we can read it and modify it. Let $1 \leq k \leq n$ be some parameter. We now think on X, Y^t as sequences of k blocks of size n/k, and consider each block as a symbol. If $\lambda > 1$, then everything written on the output stream (the only one that can grow) after the n-th bit is considered to be part of the Y_k^t. For instance X_i denotes the i-th block of X, which is of size n/k bits. We write X_{-i} for X without its i-th block, and $X_{>i}$ (resp. $X_{<i}$) for the last $(k-i)$ blocks of X (resp. the first $(i-1)$ blocks). For each $1 \leq t \leq p$, let $L_t \in \{0,\dots,k-1\}$ be the block where the input head and the output head meet during the t-th pass. Since passes are synchronised, L_t is unique and is the only block where both heads can be simultaneously during the t-th pass. Let M_i^t be the memory state as the output head enters the i-th block during t-th pass.

Consider a pass t and a block i. We would like to have an upper bound on the amount of mutual information between Y_i^t and X_i that is gained during pass t (with regards to information known at pass $t-1$). Let $\Delta_{i,j}^t = \mathrm{I}(X_i : Y_i^t | L_t = j, X_{-i}) - \mathrm{I}(X_i : Y_i^{t-1} | L_t = j, X_{-i})$ for some i and j. Of course, if $i = j$, without looking inside the block structure we only have the trivial bound $\Delta_{i,i}^t \leq H(X_i) = n/k$. It is however easier to bound other blocks. Assume without loss of generality that $i < j$. We use the data processing inequality $I(f(A) : B|C) \leq I(A : B|C)$ with Y_i^t as a function of M_i^t and Y_i^{t-1}. This gives us

$$\Delta_{i,j}^t \leq \mathrm{I}(X_i : X_{>i}, M_i^t, Y_i^{t-1} | L_t = j, X_{-i}) - \mathrm{I}(X_i : Y_i^{t-1} | L_t = j, X_{-i}).$$

We can remove $X_{>i}$ which is contained in the conditioning. Applying the chain rule, we cancel out the second term and are left with

$$\Delta_{i,j}^t \leq \mathrm{I}(X_i : M_i^t | L_t = j, X_{-i}, Y_i^{t-1}) \leq \mathrm{H}(M_i^t) \leq s.$$

The same holds if $j < i$ instead. If $j = i$, then $\Delta_{i,j}^t \leq n/k$ because $\mathrm{H}(X_i) = n/k$.

We fix j, sum over i and get $\sum_{i=0}^{k-1} \Delta_{i,j}^t \leq n/k + ks$. The expectation over $j \sim L_t$ is

$$\sum_{i=0}^{k-1} \mathrm{I}(X_i : Y_i^t | L_t, X_{-i}) - \mathrm{I}(X_i : Y_i^{t-1} | L_t, X_{-i}) \leq n/k + ks.$$

From Fact 11, we get the following inequalities:

$$\mathrm{I}(X_i : Y_i^t | L_t, X_{-i}) \geq \mathrm{I}(X_i : Y_i^t | X_{-i}) - \mathrm{H}(L_t),$$

$$\mathrm{I}(X_i : Y_i^{t-1} | L_t, X_{-i}) \leq \mathrm{I}(X_i : Y_i^{t-1} | X_{-i}) + \mathrm{H}(L_t).$$

Therefore,

$$\sum_{i=0}^{k-1} \mathrm{I}(X_i : Y_i^t | X_{-i}) - \mathrm{I}(X_i : Y_i^{t-1} | X_{-i}) \leq n/k + ks + k \log k.$$

Summing over t yields

$$\sum_{i=0}^{k-1} \mathrm{I}(X_i : Y_i^p | X_{-i}) \leq p(n/k + ks + k \log k).$$

By hypothesis $s \geq \log n$, $\varepsilon \leq 1/3$ and $\mathrm{I}(X_i : Y_i^p | X_{-i}) \geq (1 - \mathrm{H}(\varepsilon))n/k$. If $k = \sqrt{n/s}$, then $s = \Omega(n/p^2)$. ◀

▶ **Fact 11.** *Let A, B, C, D be random variables. Then*

$$\mathrm{I}(A : B|D) - \mathrm{H}(C|D) \leq \mathrm{I}(A : B|C, D) \leq \mathrm{I}(A : B|D) + \mathrm{H}(C|D).$$

▶ **Theorem 12.** *Let $0 < \varepsilon \leq 1/3$. Every p-pass λ-expansion RW/WO streaming algorithm for* Reverse *on $\{0,1\}^n$ with error ε requires space $\Omega(n/p^2)$.*

Proof of Theorem 12. Consider a p-pass λ-expansion randomized RW/WO algorithm for Reverse on $\{0,1\}^n$ with error ε and space s, with $s \geq \log n$. As before, we consider the model where the algorithm writes X on Y, using passes from right to left. This prrof is similar to the proof of Theorem 10.

We proceed as before: we assume passes are synchronized at the cost of a factor at most 2 in p. We also keep similar notations : X is the input stream uniformly distributed in $\{0,1\}^n$, and for each $1 \leq t \leq p$, $X^t \in \{0,1\}^n$ is the content of the input stream and $Y^t \in \{0,1,\perp\}^n$ is the content of the output stream Y at pass t. Let $1 \leq k \leq n$ be some parameter. As before, whe think of X, X^t, Y^t as sequences of k blocks of size n/k, and X_i denotes the i-th block of X. If $\lambda > 1$, then everithing written on the inpu stream after the n-th bit is considered part of X_k^t. We write X_{-i}, $X_{>i}$ and $X_{<i}$ as before. We also define $X_i^{\leq t}$ as the $(t+1)$-uple $(X_i, X_i^1, \ldots, X_i^t)$, i.e. the history of the i-th block until pass t.

As in prevoius proofs, for each $1 \leq t \leq p$, let $L_t \in \{0, \ldots, k-1\}$ be the block where the input head and the output head meet during the t-th pass. L_t is unique and is the only block where both heads can be simultaneously during the t-th pass. Let M_i^t be the memory state as the output head enters the i-th block during t-th pass.

Consider a pass t and a block i. As with Theorem 10, we would like to have an upper bound on the amount of mutual information between Y_i^t and X_i that is gained during pass t, assuming $L_t \neq i$. Let $\Delta_{i,j}^t = \mathrm{I}(X_i : Y_i^t | L_t = j, X_{-i}^{\leq t-1}) - \mathrm{I}(X_i : Y_i^{t-1} | L_t = j, X_{-i}^{\leq t-1})$ for some $j > i$. By the data processing inequality, we have $I(f(A) : B|C) \leq I(A : B|C)$. Therefore,

$$\Delta_{i,j}^t \leq \mathrm{I}(X_i : X_{>i}^{t-1}, M_i^t, Y_i^{t-1} | L_t = j, X_{-i}^{\leq t-1}) - \mathrm{I}(X_i : Y_i^{t-1} | L_t = j, X_{-i}^{\leq t-1}).$$

We can remove $X_{>i}^{t-1}$ which is contained in the conditioning. Applying the chain rule, we cancel out the second term and are left with

$$\Delta_{i,j}^t \leq \mathrm{I}(X_i : M_i^t | L_t = j, X_{-i}^{\leq t-1}, Y_i^{t-1}) \leq \mathrm{H}(M_i^t) \leq s.$$

The same holds if $j < i$ instead. If $j = i$, then $\Delta_{i,j}^t \leq n/k$ because $\mathrm{H}(X_i) = n/k$.

We fix j, sum over i and get $\sum_{i=0}^{k-1} \Delta_{i,j}^t \leq n/k + ks$. As before, by taking the expectation over $j \sim L_t$ and then using Fact 11, we can remove the condition $L_t = j$. This gives us

$$\sum_{i=0}^{k-1} \mathrm{I}(X_i : Y_i^t | X_{-i}^{\leq t-1}) - \mathrm{I}(X_i : Y_i^{t-1} | X_{-i}^{\leq t-1}) \leq n/k + ks + k \log k.$$

We cannot sum over t yet because the conditionning $X_{-i}^{\leq t-1}$ depends on t. However, because X_{-i}^t is a function of X_{-i}^{t-1}, the memory state at the beginning of the pass and the memory as the head on the input tape leaves the i-th block, applying Fact 11 again yields

$$\mathrm{I}(X_i : Y_i^t | X_{-i}^{\leq t}) - \mathrm{I}(X_i : Y_i^t | X_{-i}^{\leq t-1}) \leq \mathrm{H}(X_{-i}^t | X_{-i}^{\leq t-1}) \leq 2s.$$

This is a consequence of the output stream being write-only, which we had not used until now.

Algorithm 2 RW/RW streaming algorithm for Reverse

```
1  W_0 ← X ; W_1 ← Y ; α ← 0 ; // Rename the streams
2  k ← n ; // Size of current blocks
3  While k > 1
4    k ← k/2
5    For i = 1 to n/2k − 1
6      W_{1−α}[(2i + 1)k + 1, (2i + 2)k] ← W_α[2ik + 1, (2i + 1)k] // One pass
7    For i = 0 to n/2k − 1
8      W_{1−α}[2ik + 1, (2i + 1)k] ← W_α[(2i + 1)k + 1, (2i + 2)k] // Another pass
9    α ← 1 − α ; Erase W_{1−α} ; // Exchange the roles of the streams
10 Y ← W_α // Copy the final result on output tape
```

Therefore $\sum_{i=0}^{k-1} \mathrm{I}(X_i : Y_i^t | X_{-i}^{\leq t-1}) - \mathrm{I}(X_i : Y_i^{t-1} | X_{-i}^{\leq t-1}) \leq n/k + 3ks + k \log k$. Summing over t yields

$$\sum_{i=0}^{k-1} \mathrm{I}(X_i : Y_i^p | X_{-i}) \leq p(n/k + 3ks + k \log k).$$

By hypothesis $s \geq \log n$, $\varepsilon \leq 1/3$ and $\mathrm{I}(X_i : Y_i^p | X_{-i}) \geq (1 - \mathrm{H}(\varepsilon))n/k$. If $k = \sqrt{n/s}$, then $s = \Omega(n/p^2)$. ◀

Note that the proof still works if we relax the write-only model by allowing the algorithm to rewrite over data that was previously written on the output stream.

3.3 Complexity of Read-write/Read-write

Algorithm 2 proceeds by dichotomy. For simplicity, we assume that n is a power of 2, but the algorithm can easily be adapted while keeping $\lambda = 1$. At each step, it splits the input in two, copies one half to its correct place on the stream, then makes another pass to copy the other half, effectively exchanging them.

▶ **Theorem 13.** *Algorithm 2 is a deterministic $O(\log n)$-passes RW/RW streaming algorithm for* Reverse *on Σ^n with space $\mathrm{O}(\log n)$ and expansion 1.*

Proof. Since the algorithm can read and write on both tapes, they perform very similar roles. We rename the input stream W_0 and the output stream W_1. By a simple recursion, we see that whenever a block $W_{1-\alpha}[tk, (t+1)k]$ is moved, it is moved in the place that $\mathrm{Reverse}(W_{1-\alpha}[tk, (t+1)k])$ will occupy. Therefore, the algorithm is correct.

Now we prove the bounds on s and p. Algorithm 2 never needs to remember a value, only the current index and current pass, so $s = \mathrm{O}(\log n)$. Since the length of blocks copied is divided by 2 at each execution of line 4, it ends after a logarithmic number of executions of the While loop. Each iteration of the While loop requires two passes, one for each of the two For loops (lines 5 and 7). Therefore the total number of passes is in $\mathrm{O}(\log n)$. ◀

4 Sorting with two streams

Sort is generally more complex than Reverse. Even in the RW/RW model, we are not able to present a deterministic algorithm as efficient as Algorithm 2 for Reverse. With three streams, the problem becomes easy since we can Merge Sort two streams, and write the result of each step on the third one.

Algorithm 3 RW/RW streaming algorithm implementing Merge Sort

```
1  W_0 ← X ; W_1 ← Y ; α ← 0;
2  t ← 1; k ← 1 // Size of the sorted blocks
3  Expand the input : each element has a label of size log n.
4  While k < n
5    For i = 1 to n/2k {Copy B^t_{2i} on W_{1−α}}
6    For i = 1 to n/2k {For each W_α[j] ∈ B^t_{2i−1} ∪ B^t_{2i}
7                           {W_α[j] ← index of W_α[j] in B^{t+1}_i}}
8    For i = 1 to n/2k {Copy B^t_{2i} at the end of W_α after B^t_{2i−1}}
9    For i = 1 to n/2k {For each W_α[j] ∈ B^t_{2i−1}
10                          {Write W_α[j] at its position on W_{1−α}}}
11   For i = 1 to n/2k {For each W_α[j] ∈ B^t_{2i}
12                          {Write W_α[j] at its position on W_{1−α}}}
13   α ← 1 − α; Erase W_{1−α}; t ← t + 1; k ← 2k;
14 Y ← W_α
```

4.1 Merge Sort

We begin with an algorithm inspired from [5]. The algorithm works as a Merge Sort. We call B_i^t the i-th sorted block at the t-th iteration of the While loop, consisting of the sorted values of $X[2^t i + 1, 2^t(i+1)]$. Since there is no third stream to write on when two blocks B_{2i-1}^t and B_{2i}^t are merged into B_i^{t+1}, we label each element with its position in the new block. Then both halves are copied on the same stream again so that they can be merged with the help of the labels. This improves the expansion of [5] from n to $\log n$. However it is somewhat unsatisfying because when Σ is of constant size, our algorithm still has $\Omega(\log n)$ expansion.

▶ **Theorem 14.** *Algorithm 3 is a deterministic* $\mathrm{O}(\log n)$*-pass RW/RW streaming algorithm for* Sort *on* Σ^n *with space* $\mathrm{O}(\log n)$ *and expansion* $\mathrm{O}((\log n)/\log|\Sigma|)$.

Proof. Since it is an implementation of the Merge Sort algorithm, Algorithm 3 is correct. Each iteration of the While loop corresponds to five passes on each tape, and therefore the total number of passes is in $\mathrm{O}(\log n)$. Since the algorithm only needs to remember the position of the heads, current elements and the counters k, t, it only uses memory $\mathrm{O}(\log n)$. Finally, since the label for each element is at most n, we only use space $\log n$ on the stream to write it, and therefore the expansion is at most $(\log|\Sigma| + \log n)/\log|\Sigma| = \mathrm{O}((\log n)/\log|\Sigma|)$. ◀

4.2 Quick Sort

With a Quick Sort algorithm instead of a Merge Sort, we only need to store the current pivot (of size at most $\log n$), without labeling elements. However, Quick Sort comes with its own issues: the expected number of executions of the While loop is $\mathrm{O}(\log n)$, but unless we can select a good pivot it is $\Omega(n)$ in the worst case. For this reason, we use a randomized Las vegas algorithm.

A block in Algorithm 4 is a set of elements that are still pairwise unsorted, i.e. elements that have the same relative positions to all pivots so far. The block B_i^t is the i-th lower one during the t-th iteration of the While loop, and P_i^t is its pivot. The block B_{2i-1}^{t+1} consists of all elements in B_i^t lower than P_i^t and all elements equal P_i^t with a lower index. The block B_{2i}^{t+1} is the complementary. Algorithm 4 marks the borders of blocks with the symbol $\sharp$.

Algorithm 4 selects each pivot P_i^t at random among the elements of B_i^t. While it may not do so uniformly with only one pass because $|B_i^t|$ is unknown, it has an upper bound

Algorithm 4 RW/RW streaming algorithm implementing Quick Sort

```
1  W_0 ← X ; W_1 ← Y ; α ← 0 ;
2  t ← 1 ; K ← 1 // Number of unsorted blocks
3  While K > 0
4    Abort if the total number of passes is Ω((log n)/ε)
5    Expand W_0 adding O(K) space for ♯ and pivots
6    For i = 1 to K
7      Find P_i^t at random
8      W_{1−α}[i] ← P_i^t
9      Replace P_i^t with a ⊥ on W_α
10   For i = 1 to K
11     Copy W_{1−α}[i] = P_i^t at the start of B_i^t on W_α
12   For i = 1 to K
13     Write all elements in B_{2i−1}^{t+1} on W_{1−α}
14     Write ♯P_i♯ on W_{1−α}.
15     Leave space for the rest of B_{2i}^{t+1}
16   For i = 1 to K
17     Write all elements in B_{2i}^{t+1} in the space left on W_{1−α}
18   α ← 1 − α; Erase W_{1−α}; t ← t + 1
19   K ← new number of unsorted blocks // using an additional pass
20 Y ← W_α
```

$k \geq |B_i^t|$. Algorithm 4 selects $l \in \{1, \ldots, k\}$ uniformly at random, then picks l_i the remainder modulo $2^{\lceil \log |B_i^t| \rceil}$ of l. This can be computed in one pass with $\mathrm{O}(\log n)$ space by updating l_i as the lower bound on $|B_i^t|$ grows. It uses $P_i^t = B_i^t[l_i]$. P_i^t is selected uniformly at random from a subset of size at least half of B_i^t, which guarantees that with high probability $min(|B_{2i-1}^{t+1}|, |B_{2i}^t|) = \Omega(|B_i^t|)$.

▶ **Theorem 15.** *For all $0 < \varepsilon \leq 1/2$, Algorithm 4 is a randomized $\mathrm{O}((\log n)/\varepsilon)$-pass RW/RW Las Vegas streaming algorithm for Σ^n with failure ε, space $\mathrm{O}(\log n)$ and expansion $\mathrm{O}(1)$.*

Proof. Algorithm 4 is an implementation of the Quick Sort algorithm, and is therefore correct. Its correctness does not depend on the quality of the pivots. The bound on the memory and the expansion are direct. While it uses additional symbols $\perp$ and $\sharp$, they can be easily replaced by an encoding with symbols of Σ appearing in the input with only $\mathrm{O}(1)$ expansion. Because for each i and t, with high probability $min(|B_{2i-1}^{t+1}|, |B_{2i}^t|) = \Omega(|B_i^t|)$, with high probability Algorithm 4 terminates in $\mathrm{O}(\log n)$ passes. ◀

Open problems

The first open problem is whether our lower bounds still hold if multiple rewrites are allowed in the read-only/write-only model.

Another one is whether there exists a deterministic $\mathrm{O}(\log n)$-pass RW/RW streaming algorithm for Sort with space $\mathrm{O}(\log n)$ and expansion $\mathrm{O}(1)$. We can derandomize Algorithm 4 using any deterministic algorithm that finds a good approximation of the median. The best algorithm we obtained that way has $\mathrm{O}(\alpha^{-1} \log n)$ passes, $\mathrm{O}(n^\alpha)$ memory and $\mathrm{O}(1)$ expansion, for any $\alpha > 0$. We conjecture that there is no such algorithm and that having constant expansion algorithm for Sort requires a tradeoff in number of passes, memory space, number of streams or determinism.

Last, combining the algorithm from Theorem 8 with the results in [14], we obtain

a $O(\sqrt{n}/\log n)$-pass RO/RW streaming algorithm with space $O((\log n)^2)$ for recognizing well-parenthesized expressions with two parentheses types. We do not know if this is optimal.

References

1. Alok Aggarwal and Jeffrey Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
2. Paul Beame and Trinh Huynh. The value of multiple read/write streams for approximating frequency moments. *ACM Transactions on Computation Theory*, 3(2):6, 2012.
3. Paul Beame, T.S. Jayram, and Atri Rudra. Lower bounds for randomized read/write stream algorithms. In *Proceedings of the 39th annual ACM symposium on Theory of computing*, pages 689–698, 2007.
4. Amit Chakrabarti, Graham Cormode, Ranganath Kondapally, and Andrew McGregor. Information cost tradeoffs for augmented index and streaming language recognition. *SIAM Journal on Computing*, 42(1):61–83, 2013.
5. Jianer Chen and Chee-Keng Yap. Reversal complexity. *SIAM Journal on Computing*, 20(4):622–638, 1991.
6. Nathanaël François and Frédéric Magniez. Streaming complexity of checking priority queues. In *Proceeding of the 30th International Symposium on Theoretical Aspects of Computer Science*, page 454. Citeseer, 2013.
7. Travis Gagie. On the value of multiple read/write streams for data compression. In *Information Theory, Combinatorics, and Search Theory*, pages 284–297. Springer, 2013.
8. Martin Grohe, André Hernich, and Nicole Schweikardt. Lower bounds for processing data with few random accesses to external memory. *Journal of the ACM*, 56(3):12, 2009.
9. Martin Grohe, Christoph Koch, and Nicole Schweikardt. Tight lower bounds for query processing on streaming and external memory data. *Theoretical Computer Science*, 380(1):199–217, 2007.
10. André Hernich and Nicole Schweikardt. Reversal complexity revisited. *Theoretical Computer Science*, 401(1):191–205, 2008.
11. Rahul Jain and Ashwin Nayak. The space complexity of recognizing well-parenthesized expressions. Technical Report 71, Electronic Colloquium on Computational Complexity, 2010.
12. Tiko Kameda and Roland Vollmar. Note on tape reversal complexity of languages. *Information and Control*, 17(2):203–215, 1970.
13. Christian Konrad and Frédéric Magniez. Validating xml documents in the streaming model with external memory. *ACM Transactions on Database Systems*, 38(4):27, 2013.
14. Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 261–270, 2010.
15. James I. Munro and Mike S. Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.
16. S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.
17. Jan Matthias Ruhl. *Efficient algorithms for new computational models*. PhD thesis, Citeseer, 2003.