Counting monotonic subsequences in a data stream

Ashish Bora, Abhishek Sinha, Harshal Priyadarshi

1 Abstract

We study the space complexity of algorithms for counting the number of k-length monotonic subsequences in a data stream. Without loss of generality, we will focus on the monotonically decreasing case (k-dec-count). Our work is related to, and is a generalization of the well studied problem of counting inversions in the streaming model [1].

We prove $\Omega(n)$ deterministic lower bounds for any streaming algorithm that exactly computes the number of inversions (2-dec-count) in a data stream of t elements where each element comes from $[n]$ and $t \geq \Omega(n)$. The proof uses a reduction argument and utilizes communication lower bounds for computation of disjointness. Our second result is $\Omega(n)$ lower bound on any algorithm that correctly computes k-dec-count in a data stream. Once again, we use a reduction argument to compute inversions using any algorithm that computes k-dec-count.

On the other hand, highly efficient approximation algorithms are known for 2-dec-count [1]. We present our attempts at obtaining similar improvements for k-dec-count with $k \geq 3$.

2 Introduction

Recently, there has been a massive growth in the total amount of data available to us. This has necessitated the creation of space efficient algorithms to process data. Streaming Algorithms are a natural fit to this setting. The streaming model was first introduced in [2] and the paper studies problem of computing moments of the frequency vector of data items. This paper also introduces techniques to prove lower bounds on the space complexity of streaming algorithms by reduction from problems in communication complexity.

Many of the well known streaming algorithms attempt to compute permutation invariant functions (functions whose value remains the same irrespective of the order of elements in the stream). A string of papers have obtained tight bounds on a class of such functions. Estimating k^{th} moment of the frequency vector over a data stream is perhaps the most well studied problem in this domain. [3] provides algorithms with the best known space complexity for this problem. [4] proves matching lower bounds using problems in communication complexity. Another interesting approach is explored in [5] where a single linear sketch of the data stream can be used for approximate computation of a large class of permutation invariant functions.

In contrast, there has been relatively less work on permutation dependent functions. [1] studies space complexity of the problem of counting inversions in a streaming model. Note that this function is not permutation invariant and this paper was among the first to study computation of such functions in the streaming model. Another example of such a work is [6], where the authors study the problem of finding the length of Largest Increasing Subsequence (LIS) and Longest Common Subsequence (LCS) in the streaming model.

In this work, we attempt to study the space complexity of algorithms for counting the number of k-length monotonic subsequences in a data stream. We will focus on the monotonically decreasing case (k-dec-count), but note that the problem is essentially equivalent for the monotonically increasing case. This problem is a generalization of the problem of counting inversions (obtained by setting $k = 2$).

We obtain the following two results:

Theorem 2.1. Let $0 < c$ be a constant. Let $cn \leq t$. Then any zero error randomized algorithm that counts the number of inversions in a data stream of length t, where each element is from $[n]$, must have $\Omega(n)$ space complexity.

A proof is presented in the appendix. We note that our proof uses a reduction argument and uses the communication complexity lower bound for the disjointness function. A similar theorem is presented in [1], but unlike their setting, our proof does not require the input to be a permutation and in particular also works for $t < n$.

Theorem 2.2. Let $0 < c$ be a constant. Let $cn \leq t$. Then any zero error randomized algorithm that counts the number of k-length monotonically decreasing subsequences in a data stream of length t, where each element is from $[n]$, must have space complexity $\Omega(n)$.

This is a generalization of the previous theorem. A proof can be found in the appendix. Our main technique is to reduce 2-dec-count problem to the k-dec-count problem.

3 Related Work

The work most closely related to ours is [1] which is also the basis for many of the approaches we describe. This paper describes a number of deterministic, randomized as well as approximate techniques for counting the number of inversions in different kinds of streams. There are two very important results in this paper. The first is the $\mathcal{O}\left(\sqrt{\frac{n}{\epsilon}}\log(n)\right)$ space algorithm for counting the number of inversions in a stream with ϵ approximation error. The other is the $\mathcal{O}\left(\epsilon^{-1}\sqrt{n}\log(n)\log(mn)\right)$ space algorithm for counting the number of inversions in a list of size n with elements chosen from $[m]$ and approximation error of ϵ.

This paper also describes some other efficient approaches for counting inversions in permutations. It describes a $\mathcal{O}\left(\log^2(n)\right)$ deterministic algorithm for computing inversions by maintaining a constant number of counters at $\log(n)$ different levels. It converts this to a $\mathcal{O}\left(\log(n)\log\log(n)\right)$ randomized algorithm by using a similar randomization trick as was used in [2] to reduce the number of bits per counter. Finally, it also gives a $\mathcal{O}\left(\log(n)\log\log(n)\right)$ space deterministic algorithm for computing the number of inversions in a permutation.

The paper also derives space lower bounds for a variety of settings. This is done by reducing from the Disjointness problem in communication complexity. For the exact computation setting, the authors show that any randomized (or deterministic) algorithm has a space lower bound of $\Omega(n)$. For the approximate computation setting, they show that any randomized algorithm for computing inversions requires $\Omega(\log(n))$ space.

4 $\mathcal{O}\left(\sqrt{\frac{n}{\epsilon}}\log(n)\right)$ algorithm for computing inversions

We first describe an $\mathcal{O}\left(\sqrt{\frac{n}{\epsilon}}\log(n)\right)$ algorithm from [1] for computing inversions in a permutation (presented as a stream) with an ϵ approximation ratio. This algorithm is the foundation for our attempts at solving the 3-dec-count problem described later. At a high level, this algorithm divides the set of integers from 1 to n into $\mathcal{O}\left(\sqrt{n}\right)$ buckets. It then divides the problem of computing inversions into two parts – computing *global inversions* and computing *local inversions*. Global inversions are those where the two elements belong to different buckets while local inversions consist of all the remaining inversions. Global inversions are computed exactly whereas local inversions are computed approximately using an $\mathcal{O}\left(\sqrt{n}\right)$ sized sliding window buffer. We now describe the details of the algorithm.

- Let ϵ be the desired approximation ratio and define s to be $\sqrt{\epsilon n}$.

- The algorithm divides the set of integers from 1 to n into $\frac{n}{s}$ buckets each of size s. At any step in the algorithm, we maintain the exact count of the number of integers that have arrived in each bucket. This requires $\mathcal{O}\left(\frac{n}{s}\log(n)\right) = \mathcal{O}\left(\sqrt{\frac{n}{\epsilon}}\log(n)\right)$ space.
- In addition, we also create a buffer to store the cs most recently arriving elements where $c = \lceil 1 + \frac{4}{\epsilon} \rceil$. This require $\mathcal{O}\left(cs\log(n)\right)$ bits.
- Finally, we maintain two counters of $\log(n)$ bits each which maintain the count of global inversions and local inversions respectively.
- Whenever a new element p arrives, we first determine the bucket d to which it belongs. We then do the following updates
  - We update the count for bucket d by 1.
  - We update the global elements counter by adding to it the counter values for the buckets to the left of d.
  - We pop an element from the local buffer if it is already of cs size and add p to it. We update the local inversions counter by adding to it the number of elements in the local buffer which also belong to bucket d and are greater than p.
- At the end of the stream, the global and local inversions counters are added to get the final count.

4.1 Metrics on the Space of Permutations

Before we delve into the proof sketch, we first describe an interesting property of inversions that is also described in [1] for the purpose of this proof.

Firstly we give a more general definition of inversions. Given two arbitrary permutations σ and τ, the number of inversions of τ with respect to σ is defined as the number of pairs (u, v) such that $\sigma(u) < \sigma(v)$ and $\tau(u) > \tau(v)$. This is known as the *Kendall-Tau* distance [7][1] between the permutations and is denoted as $K(\sigma, \tau)$. It is a metric on the space of permutations. The common notion of inversions basically refers to the number of inversions of a permutation σ with respect to the identity permutation.

Another metric that can be defined on the space of permutations is the Spearman's footrule metric [7][1]. For two permutations σ and τ, this is represented as $F(\sigma, \tau)$ and defined as $F(\sigma, \tau) = \sum_{u=1}^{u=n} |\sigma(u) - \tau(u)|$. If τ is the identity permutation ($\tau = I$), then $F(\sigma, I) = \sum_{u=1}^{u=n} |\sigma(u) - u|$.

Diaconis and Graham [7] proved a theorem that relates the two metrics. This theorem states that for any two permutations σ and τ, $K(\sigma, \tau) \leq F(\sigma, \tau) \leq 2K(\sigma, \tau)$. If we consider the special case of $\tau = I$, this implies that the Spearman's footrule metric of a permutation with respect to the identity permutation is both upper and lower bounded by some multiple of the number of inversions (common notion) in the permutation.

4.2 Proof Sketch

We now give a brief sketch of why the algorithm described above gives ϵ approximation factor for computing the number of inversions. Again, this is taken from [1].

- One of the first things to observe is that we only err while counting local inversions. The count of global inversions is exact since we maintain the exact count of elements in every bucket at every step of the algorithm.
- Let us try to analyze the properties of a missed local inversion. It corresponds to a pair of integers u, v such that $u < v$ but $\sigma(u) > \sigma(v)$. Since the inversion is local, u and v must be close, i.e. $|u - v| \leq s$ and since the inversion was not counted using the local buffer, $|\sigma(u) - \sigma(v)| \geq cs$.
- The two inequalities above, (along with triangle inequality) allow us to get a lower bound on $|u - \sigma(u)|$ or $|v - \sigma(v)|$.

- Thus, each missed inversion contributes to the Spearman's footrule metric, which allows us to bound number of missed inversions in terms of this distance.
- Then using the result of Diaconis and Graham described above, we can bound the total number of missed local inversions in terms of the total number of inversions in the stream.

5 Our approach

We now focus our attention on solving 3-dec-count for permutations, *i.e.* Given a permutation σ of $[n]$, we wish to count the number of monotonically decreasing subsequences of length 3. For any two permutations σ and τ we define $K_3(\sigma, \tau)$ to be the total number of subsequences of length 3 that are monotonically decreasing in σ but monotonically increasing in τ. Thus,

$$K_3(\sigma, \tau) = |\{(i, j, k) : \sigma(i) > \sigma(j) > \sigma(k) \text{ and } \tau(i) < \tau(j) < \tau(k)\}|$$

The quantity we wish to compute is $K_3(\sigma) = K_3(\sigma, I)$, where I is the identity permutation.

Similar to the $\mathcal{O}\left(\sqrt{n/\epsilon}\log(n)\right)$ space algorithm presented above, in this section we outline a strategy to solve 3-dec-count using local and global counters.

We divide $[n]$ into n/s intervals each of size s. Let us denote these intervals by $\{I_1, I_2, \cdots I_{n/s}\}$. Let $I(u)$ denote the interval that u belongs to. *i.e.* $I(u) = t$ iff $u \in I_t$. Then all 3-dec subsequences (u, v, w) can be classified into the following four categories:

1. $I(u) > I(v) > I(w)$
2. $I(u) = I(v) > I(w)$
3. $I(u) > I(v) = I(w)$
4. $I(u) = I(v) = I(w)$

Type 1 inversions are called global. Type 2 and 3 are called semi-local and type 4 are called local.

To count 3-dec of type 1, it is sufficient to maintain counters for each interval which count the number of elements in that interval seen till now, i.e.

$$N_k^t = |\{j : \sigma(j) \leq t, j \in I_k\}|.$$

This takes $\mathcal{O}\left(\frac{n}{s}\log(n)\right)$ space.

To count 3-dec of type 2 and 3, it is sufficient to maintain

$$M_k^t = |\{(i, j) : i \leq j, t \geq \sigma(i) \geq \sigma(j), i \in I_k, j \in I_k\}|$$

5.1 One pass algorithm for approximate counting of inversions in a list

We note that getting an efficient approximation for M_k^t is essentially equivalent to finding an efficient one pass algorithm to approximate the number of inversions in a list of n distinct elements where each element comes from $[m]$. Given such an algorithm we can apply one instance of each to our buckets and use the resulting counts to approximate 3-dec of Type 2 and 3.

We first study a solution presented in [1] for a related problem. This is a modification of the algorithm used for the case of permutations. Recall that the set $[n]$ was split into intervals of size $\sqrt{\epsilon n}$. The modified algorithm uses two passes over the data. The first pass is used to construct the intervals by randomly sampling endpoints from the list. Then, in the second pass the algorithm proceeds exactly as in the case of permutations. This algorithm has a space complexity of $\mathcal{O}\left(\epsilon^{-1}\sqrt{n}\log(n)\log(mn)\right)$.

However, for our setting, we need a one-pass algorithm that can additionally gives good approximation at each step. Thus, we attempted to modify the two pass algorithm to make it one pass.

The first part is constructing interval boundaries to be uniformly distributed from the given list. This can be easily done by reservoir sampling by fixing the number of intervals apriori. Instead of $\sqrt{n}$ intervals, we construct $\sqrt{m}$ intervals. We are okay with paying $\mathcal{O}\left(\epsilon^{-1}\sqrt{m}\log(n)\log(mn)\right)$ cost since our interval sizes (m) will be small. Thus, we maintain a set of $\sqrt{m}$ entries from the list and add new elements probabilistically as in reservoir sampling. This involves maintaining an additional counter of size $\mathcal{O}\left(\log(n)\right)$.

When the intervals change, some local inversions become global or vice versa. So the second challenge is to ensure that the buffer correctly counts the local inversions. Fortunately, the notion of local and global inversions for all inversions ending at a given element is exactly the same as the one when the element is inserted. Thus, the buffer counts are computed as intended.

The third challenge is to correctly update the counters with the changing interval boundaries. Removing an element from the interval set just requires merging two intervals and thus their counts. The harder case is when we insert an element in the interval set. It is not clear how the count for the original interval should be split for the two newly created intervals. If the distribution of the elements in the list was uniform, a sensible strategy would be to split according to interval lengths. However, this strategy is not robust – we show by a counterexample that one can get very large additive/multiplicative errors using this approximation scheme. The counterexample is as follows. Say we have got the first $\frac{m}{100}$ integers from $[m]$ in the stream till now. At this stage, the rightmost interval will roughly have a size of $\frac{\sqrt{m}}{100}$. In addition, suppose that the next element that we get is $\frac{m}{2}$. Also suppose that reservoir sampling results in this new element being chosen as a pivot replacing the endpoint to the left of it. Now, since both the intervals to the right and left of $\frac{m}{2}$ have similar sizes, they will end up getting similar counts. In reality, the interval to the right should actually have gotten 0 count and the interval to the left $\frac{\sqrt{m}}{100}$.

Thus, we do not have a provably correct one pass algorithm as required in our setting.

5.2 Recursive counting for local 3-dec-count

To count 3-dec of Type 4, one can simply invoke recursion. i.e. on each of the subintervals, we can use the algorithm to compute local 3-dec sequences.

Let $f_3(n)$ be the space complexity of counting 3-dec sequences in a permutation of $[n]$. Let $g_2(n, m)$ be the space complexity of approximately counting the number of inversions in a list of length $[n]$, where each element is a distinct element from $[m]$, such that the approximation holds at every step of the algorithm.

Let us split into n/s intervals, each of size s. This gives the following recursion:

$$f_3(n) = \frac{n}{s}\log(n) + g_3(\frac{n}{s}, \frac{n}{s}) + \frac{n}{s}f_3(s)$$

By using Master Theorem, $f(n) = \Omega(n)$. Thus, this method does not give us any advantage.

5.3 Buffer for counting local 3-dec-count

As in the algorithm described in Section 4, one can follow the same strategy. *i.e.* we maintain a buffer of size B and estimate local 3-dec-count within the buffer. This will miss some of the local 3-dec sequences and we would like to bound those in terms of the total 3-dec-count.

As we saw in Section 4, the proof for the 2-dec case proceeds by identifying that for any missed local inversion, (u, v), it must be that u and v are close, i.e. $|u - v| < s$ since it is a local inversion. Further they must occur in the permuted sequence far apart, i.e. $|\sigma(u) - \sigma(v)| > B$, since this inversion is missed. Through these and the triangle inequality, one gets an upper bound on missed inversions in terms of the Spearman's

footrule metric $F(\sigma)$ (see Section 4.1). Now since $F(\sigma)$ is within a factor of 2 of $K(\sigma)$, we get the desired bound.

The same proof technique has significant hurdles when used for 3-dec-count. For any missed 3-dec sequence (u, v, w), we have that $|u - w| < s$, and $|\sigma(u) - \sigma(w)| > B$. This is bounded in terms of Spearman's footrule metric as before, but this connection is not very helpful since there is no good way to upper bound it in terms of $K_3(\sigma)$. We show this by constructing examples where $K_3(\sigma)$ is very small, yet $K(\sigma)$ is large. Suppose we have a permutation σ where the order of the elements is as follows $\frac{n}{2}+1, \frac{n}{2}+2, n, 1, 2, \ldots, \frac{n}{2}$. We can see that the given permutation has $\frac{n^2}{4}$ inversions since we can pair any element from the first $\frac{n}{2}$ elements with any element from the last $\frac{n}{2}$ elements. At the same time, we can observe that there are no length 3 decreasing subsequences. Thus $K_3(\sigma) = 0$ while $K(\sigma) = \frac{n^2}{4}$.

Thus, this technique does not readily yield an approximation bound.

5.4 Zero or not

In this section we consider a simplified version of the the problem. Instead of trying to find the number of decreasing subsequences of length k, we just aim to find if there exists at least one decreasing subsequence of length k or not.

There is a deterministic algorithm that solves this problem is $\mathcal{O}(k \log(n))$ space. We simply maintain k numbers. The i^{th} number denotes the largest number yet found such that there is a decreasing sequence of length i ending at that number. We shall call this the best-i-sequence. When a new number arrives in a stream, we update these counters using dynamic programming – either the new element does is not part of the best-i-sequence, in which case, the i-th number remains unchanged. Otherwise it is the last element in which case, we can simply use the best $i-1$ length sequence before the current number. We store k numbers, and each requires $\mathcal{O}(\log n)$ bits.

Computing k-dec-count exactly is space inefficient for any $k \geq 2$. Computing if k-dec-count > 0 can be thought of as a very crude approximation of k-dec-count. As shown here, this can be efficiently done for small k with zero error. Our hope is that better approximations to k-dec-count (such as within a constant factor) are also similarly efficiently computable, but we have not been able to prove this for any $k \geq 3$.

6 Properties of $K_3(\sigma, \tau)$

In this section we aim to study some properties of $K_3(\sigma, \tau)$, *i.e.* the count of total number of 3-dec sequences between two permutations. Note that $K(\sigma, \tau)$ is the Kendall-Tau distance.

6.1 K_3 is not a metric

$K(\cdot, \cdot)$ is known to be a metric on the space of permutations. Similarly, the Spearman's footrule metric $F(\cdot, \cdot)$ is also a metric. As we saw in the 4.1 section, these two can be bounded in terms of each other.

In contrast the quantity we are trying to approximate (K_3) is not a metric. This is because $K_3(\sigma, \tau) = 0$ does not imply $\sigma = \tau$. We can consider the following simple counterexample. Suppose we have a permutation with the order of elements being $2, 1, 3, 4, 5, \ldots, n$. This permutation has zero 3-dec sequences with respect to the identity permutation but is clearly not the same as the identity permutation.

6.2 Relation to Kendall-Tau Metric

A simple bound is the following: $K_3 \leq nK_2$. This is simply because each inversion can participate in at most n 3-dec sequences. Thus, K_3 cannot be larger than nK_2.

We saw from example in Section 5.3 that it is possible to get $K_3 = 0$, but $K = O(n^2)$. Thus there is no hope of getting a multiplicative upper bound on K in terms of K_3. The additive bound is the worst possible, i.e. $\mathcal{O}\left(n^2\right)$ and thus not very helpful.

7 Discussion and Future work

In this work, we attempted to solve the k-dec-count problem. Our main contribution is a $\Omega(n)$ lower bound on the space complexity of any randomized algorithm for k-dec-count based on the communication complexity lower bound of zero error randomized algorithm for the disjointness function. We also presented our attempts at solving the approximate version of the 3-dec-count problem based on the approach presented for the 2-dec-count problem in [1]. Solving this problem for larger values also remains an open problem.

References

[1] M. Ajtai, T. Jayram, R. Kumar, and D. Sivakumar, "Approximate counting of inversions in a data stream," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 370–379, ACM, 2002.

[2] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 20–29, ACM, 1996.

[3] S. Ganguly, "Polynomial estimators for high frequency moments," *arXiv preprint arXiv:1104.4552*, 2011.

[4] Y. Li and D. P. Woodruff, "A tight lower bound for high frequency moment estimation with small error," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pp. 623–638, Springer, 2013.

[5] O. R. Braverman V. and R. A., "Universal streaming," in *arXiv preprint arXiv:1408.2604*, 2014.

[6] X. Sun and D. P. Woodruff, "The communication and streaming complexity of computing the longest common and increasing subsequences," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 336–345, Society for Industrial and Applied Mathematics, 2007.

[7] P. Diaconis and R. L. Graham, "Spearman's footrule as a measure of disarray," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 262–268, 1977.

[8] J. Håstad and A. Wigderson, "The randomized communication complexity of set disjointness.," *Theory of Computing*, vol. 3, no. 1, pp. 211–219, 2007.

Appendix A

In this section, we present proofs of theorems stated in Section 2.

7.1 Proof: Theorem 2.1

For a fixed c, and fixed $t > cn$, assume that there is a zero error randomized streaming algorithm to compute the number of inversions in any stream of length t (where each element comes from $[n]$) using at most M bits of memory. To argue about lower bound on M, without loss of generality, we can assume that $c < 1$ and $t \leq n$, since any algorithm for counting inversions in a large stream must necessarily correctly compute inversions in a smaller stream. This can be seen by a simple reduction whereby we append large numbers to the end of a small stream. So for the rest of the proof, we can take $c < 1$ and $t \leq n$.

The disjointness function is as follows: Let Alice (A) and Bob (B) have strings $x, y \in \{0,1\}^n$. These are interpreted as subsets of $[n]$. Then $DISJ(x, y) = \mathbb{I}(x \cap y = \phi)$. The k disjointness problem is the same as disjointness problem except that we restrict the sets x and y so that they have exactly k elements each. The communication complexity of k-disjointness problem in this setting is denoted by $R_0\left(DISJ_k^n\right)$.

We solve the private coin zero error randomized version of the $t/2$ disjointness problem using the given streaming algorithm as follows:

Alice and Bob both sort their inputs. Alice runs the algorithm to compute inversions by giving one element of x at a time in a streaming fashion. Once all the elements are given, she sends the contents of the memory to Bob. Bob then continues the algorithm by giving one element of y at a time. This computes the number of inversions in the stream (x, y). Let the answer to this be N_1.

Similarly, they can compute number of inversions in the stream (y, x). Let the answer to this be N_2.

The main insight is that x and y are disjoint iff

$$N_1 + N_2 + \frac{(|x|-1)|x|}{2} + \frac{(|y|-1)|y|}{2} = \frac{(|x|+|y|-1)(|x|+|y|)}{2}$$

To see this, note that N_1 counts the sum over all elements in x, the number of elements in y that are smaller than that element, and $(|x|-1)|x|)/2$ is the sum over all elements in x, the number of elements in x that are greater than that element. Thus, the first and third term together is the sum over all elements in x of the number of elements in x and y that are greater than that number. Similarly, second and fourth term together is the sum over all elements in y of the number of elements in x and y that are greater than that number. Thus, the total is the sum over all elements of the number of elements that are greater than that element. This should precisely give the RHS if the inputs are disjoint. If they are not disjoint, some of the pairs will not be counted (since inversion requires strict inequality). This will thus yield a smaller LHS.

So, we can distinguish between the two cases by two runs of the stream counting algorithm. Thus,

$$R_0(DISJ_{t/2}^n) \leq 2M + \mathcal{O}\left(\log(n)\right),$$

where we need $\log(n)$ communication for sending $|x|, |y|, N_1$ and N_2.

It is known that for $k \leq n/2$, $R_0\left(DISJ_k^n\right) = \Omega(k)$ [8]. Thus, in our setting, since $t/2 < n/2$,

$$2M + \mathcal{O}\left(\log(n)\right) \geq R_0(DISJ_{t/2}^n) = \Omega(t/2) \geq \Omega(cn/2) = \Omega(n)$$

Thus, $M = \Omega(n)$.

7.2 Proof: Theorem 2.2

For ease of exposition, we give the proof of this result for the special case of $k = 3$. The proof for larger values of k follows along similar lines giving $\Omega(n)$ bound.

We know that there exists a $\Omega(n)$ lower bound for 2-dec-count. We try to find a reduction from 2-dec-count to 3-dec-count.

In this regard, we assume that we have an algorithm for the 3-dec-count problem (called 3-Alg from now). We can construct an algorithm for the 2-dec-count problem as follows.

- We keep forwarding the stream elements we receive to 3-Alg. At the end of the stream, the output of 3-Alg is the solution of the 3-dec-count problem.
- After this, we forward 1 more element to 3-Alg which is smaller than all the other elements in the stream (we choose this to be 0 although it could be any number smaller than 1). As a result, the output of 3-Alg will be updated to include all the length 3 decreasing sequences that end at this newly added element.
- These set of sequences correspond in a one-to-one fashion to the inversions in the streams. Let (a, b) be an inversion in the original stream. Then $a > b$ and both of them by construction are greater than 0. Thus $a > b > 0$ .Hence the inversion corresponds to a length 3 sequence ending at the appended 0. The other direction is trivial.
- We can now subtract off the value we had before appending the element to get the number of inversions in the stream. Hence, we have found an algorithm for the 2-dec-count that requires only $\mathcal{O}(\log(n))$ additional space. Since the lower bound for 2-dec-count is $\Omega(n)$, the lower bound for 3-dec-count will also be $\Omega(n)$.