Near-Optimal Relative Error Streaming Quantile Estimation via Elastic Compactors

Elena Gribelyuk* Pachara Sawettamalya† Hongxun Wu‡ Huacheng Yu§

Abstract

Computing the approximate quantiles or ranks of a stream is a fundamental task in data monitoring. Given a stream of elements $x_1, x_2, \ldots, x_n$ and a query x, a relative-error quantile estimation algorithm can estimate the rank of x with respect to the stream, up to a multiplicative $\pm\epsilon \cdot \mathrm{rank}(x)$ error. Notably, this requires the sketch to obtain more precise estimates for the ranks of elements on the tails of the distribution, as compared to the additive $\pm\epsilon n$ error regime. This is particularly favorable for some practical applications, such as anomaly detection.

Previously, the best known algorithms for relative error achieved space $\tilde{O}(\epsilon^{-1}\log^{1.5}(\epsilon n))$ (Cormode, Karnin, Liberty, Thaler, Veselỳ, 2021) and $\tilde{O}(\epsilon^{-2}\log(\epsilon n))$ (Zhang, Lin, Xu, Korn, Wang, 2006). In this work, we present a nearly-optimal streaming algorithm for the relative-error quantile estimation problem using $\tilde{O}(\epsilon^{-1}\log(\epsilon n))$ space, which almost matches the trivial $\Omega(\epsilon^{-1}\log(\epsilon n))$ space lower bound.

To surpass the $\Omega(\epsilon^{-1}\log^{1.5}(\epsilon n))$ barrier of the previous approach, our algorithm crucially relies on a new data structure, called an *elastic compactor*, which can be dynamically resized over the course of the stream. Interestingly, we design a space allocation scheme which adaptively allocates space to each compactor based on the "hardness" of the input stream. This approach allows us to avoid using the maximal space *simultaneously* for every compactor and facilitates the improvement in the total space complexity.

Along the way, we also propose and study a new problem called the Top Quantiles Problem, which only requires the sketch to provide estimates for the ranks of elements in a fixed-length tail of the distribution. This problem serves as an important subproblem in our algorithm, though it is also an interesting problem of its own right.

1 Introduction

Learning the distribution of data that are represented as a stream is an important task in streaming data analysis. A concrete problem that captures this task is the *streaming quantile estimation* problem.

Given a stream of elements $\pi = (x_1, x_2, \ldots, x_n)$, the quantile estimation problem asks us to process the stream, while maintaining a *small memory* that stores a few input elements, such that at the end of the stream, for any given query y, the algorithm must output an approximation of the *rank* of y in π with high probability, i.e., an approximation of $\mathrm{rank}_\pi(y) := |\{i \in [n] : x_i < y\}|$.[1]

The problem has been extensively studied [9, 13, 1, 8, 14, 2, 16] when we allow *additive error*, i.e., the algorithm outputs an estimate $\widehat{\mathrm{rank}}_\pi(y) = \mathrm{rank}_\pi(y) \pm \epsilon n$ with high probability. Optimal bounds are known in this setting: Karnin, Lang, and Liberty proposed an algorithm with $O((1/\epsilon)\log\log(1/\delta))$ space, which matches the best space one can hope for even for *offline* algorithms when the failure probability δ is a constant [13].[2]

*Department of Computer Science, Princeton University, Email: eg5539@princeton.edu. Supported in part by an NSF CAREER award CCF-2339942.

†Department of Computer Science, Princeton University, Email: pachara@princeton.edu. Supported in part by an NSF CAREER award CCF-2339942.

‡Department of EECS, UC Berkeley. Email: wuhx@berkeley.edu. Supported by Avishay Tal's Sloan Research Fellowship, NSF CAREER Award CCF-2145474, and Jelani Nelson's NSF award CCF-2427808.

§Department of Computer Science, Princeton University, Email: yuhch123@gmail.com. Supported in part by an NSF CAREER award CCF-2339942.

[1] In this work, we consider algorithms in *comparison-based model*, wherein stream elements are drawn from a universe equipped with a total-ordering. At any time, the algorithm may only performed comparisons between any two elements stored in memory, and does not depend on the true value of each element.

[2] An offline algorithm sees the elements all at once and computes a small sketch that can answer rank queries approximately.

On the other hand, oftentimes, the application needs to accurately learn the tail distribution of the data stream. For instance, this need arises when monitoring network latencies: the distribution of response times is often very long-tailed, and understanding the occasional, yet problematic, high response times is a key purpose of the task [4]. For such applications, algorithms that guarantee *relative errors* give high accuracy on the tail distribution, and are thus more aligned with this stricter requirement. That is, the algorithm must return $\widehat{\text{rank}}_\pi(y) = (1 \pm \epsilon)\text{rank}_\pi(y)$.[3] The relative-error quantile estimation task also arises when approximately counting the inversions in a stream [11].

The optimal bound for offline algorithms with relative error is $\Theta(\epsilon^{-1}\log(\epsilon n))$, by simply storing elements with ranks $\{1, 2, \ldots, \epsilon^{-1}\}$ and $\{\epsilon^{-1}(1+\epsilon), \epsilon^{-1}(1+\epsilon)^2, \ldots\}$. Best-known streaming algorithms are Multi-Layer Randomization ("MR" algorithm) by Zhang, Lin, Xu, Korn and Wang [20] with $O(\epsilon^{-2}\log(\epsilon^2 n))$ space, and a recent breakthrough [3] by Cormode, Karnin, Liberty, Thaler and Veselỳ with $O(\epsilon^{-1}\log^{1.5}(\epsilon n))$ space (we will refer to it as the CKLTV algorithm below).

The MR algorithm [20] maintains logarithmically many sketches of size $O(\epsilon^{-2})$. Each sketch is responsible for queries with rank in $[\epsilon^{-2}2^i, \epsilon^{-2}2^{i+1})$ for some i. More recently, Cormode et al. [3] introduced "relative compactors". Roughly speaking, a relative compactor takes a stream of elements as input and outputs a shorter stream such that the rank of any query in the *input* stream can be approximated with *small relative error* based on its rank in the *output* stream (see Section 5 for a more detailed overview). Then, the algorithm of [3] "connects" logarithmically many relative compactors, i.e., the output stream of the previous relative compactor is fed (online, in the streaming sense) to the next relative compactor as its input stream. However, both of the aforementioned algorithms ([3], [20]) have the optimal dependence on one of the parameters ϵ and n, while are suboptimal by a polynomial factor on the other.

A natural question is whether we can improve the sketch size in the MR algorithm, or improve the relative compactor space in CKLTV, so that the offline optimal space for this problem can also be achieved in streaming. It turns out that the answer to this question is *yes and no*. Neither of the two subroutines can be improved in general, due to a lower bound that we prove in Section 7. On the other hand, the "bad" streams that lead to such lower bounds inherently cannot be combined – the algorithm maintains logarithmically many sketches of relative compactors, there are bad streams that would force one of them to consume large space, but not all of them simultaneously.

Based on this observation, we propose a new streaming algorithm for quantile estimation with relative errors using nearly optimal space $\tilde{O}(\epsilon^{-1}\log\epsilon n)$[4], nearly matching the trivial $\Omega(\epsilon^{-1}\log(\epsilon n))$ offline lower bound[5]. We use the framework of MR, and maintain logarithmically many sketches such that each sketch is responsible for answering queries with rank in $[\epsilon^{-1}2^i, \epsilon^{-1}2^{i+1})$. Now, each sketch is implemented using a collection of new data structures, which we call *elastic compactors*. Elastic compactors are inspired by relative compactors, but they have one crucial additional feature: they are resizable. Since our sketch for each of the (logarithmically-many) scales will be built using these elastic compactors, we will actually be able to resize the entire sketch for each scale as needed. Depending on how "difficult" the input stream for each sketch is, our algorithm dynamically allocates space to the sketches, and resizes them to the current space on-the-fly as the stream is observed. Whenever a piece of "bad input stream" targeting a specific sketch occurs (i.e. the one that we construct in Section 7), the algorithm automatically allocates more space to that sketch temporarily, while still guaranteeing that the total space of all sketches is always bounded by $\tilde{O}(\epsilon^{-1}\log\epsilon n)$ with high probability.

THEOREM 1.1. *Let $0 < \delta \le 0.5$ and $0 < \epsilon \le 1$. There is a randomized, comparison-based, one-pass streaming algorithm that, when processing a stream π consisting of n elements, produces a sketch satisfying the following: for any query $x \in \mathcal{U}$, the sketch returns an estimate $\widehat{\text{rank}}_\pi(x)$ for $\text{rank}_\pi(x)$ such that with probability $1-\delta$,*

$$|\widehat{\text{rank}}_\pi(x) - \text{rank}_\pi(x)| \le \epsilon \cdot \text{rank}_\pi(x),$$

[3]The definition as-is gives higher accuracy for queries with small ranks. By running the algorithm with a reversed total-ordering of stream elements, we can obtain high accuracy at the tail of the distribution.

[4]The $\tilde{O}$ hides the dominated $\log(1/\epsilon)$, $\log\log n$ and $\log\log(1/\delta)$ terms. For a precise space bound, see Theorem 1.1.

[5]The $\Omega(\epsilon^{-1}\log(\epsilon n))$ lower bound can be shown by inserting $\epsilon^{-1}\log(\epsilon n)$ many distinct elements $x_1 < x_2 < \cdots < x_{\epsilon^{-1}\cdot\log(\epsilon n)}$ where for any $1 \le i \le \log(\epsilon n)$, the elements $x_{\epsilon^{-1}(i-1)+1}, \ldots, x_{\epsilon^{-1}i}$ are inserted 2^i times each. Any algorithm, even offline one that can see the entire stream, must keep all elements $x_1, x_2, \ldots, x_{\epsilon^{-1}\cdot\log(\epsilon n)}$ in memory to satisfy the error guarantee.

where the probability is over the internal randomness of the streaming algorithm. Moreover, the total space used by the sketch is

$$O(\epsilon^{-1}\log(\epsilon n)\cdot\log(1/\epsilon)\cdot(\log\log n+\log(1/\epsilon))\cdot(\log\log 1/\delta)^3).$$

There are several consequences of our construction, which to recall, consists of logarithmically-many resizable sketches for each scale $[\epsilon^{-1}2^i, \epsilon^{-1}2^{i+1})$. First, since the sketch can be easily resized, our algorithm actually does not need to know the stream length n in advance, and the same algorithm works as the stream length increases. Another important feature in practice is *mergeability*, i.e. it is useful to be able to summarize two substreams π_1 and π_2 separately into sketches $\mathcal{M}_1, \mathcal{M}_2$, and then create a *merged* sketch $\mathcal{M}$ which applies to the combined stream $\pi = \pi_1 \bigsqcup \pi_2$ with similar error and space guarantees. Currently, it is not clear whether our relative-error quantiles sketch is fully-mergeable (See Section 8 for more discussion).

All-quantiles estimation. As a straight-forward corollary of Theorem 1.1, we obtain a sketch that satisfies the *all-quantiles guarantee*, meaning that for all queries $x \in \mathcal{U}$ *simultaneously*, the sketch provides an accurate estimate with high probability. The proof proceeds by a standard union bound over an ϵ-net, and is nearly identical to argument given in Appendix B of [3].

COROLLARY 1.2. *Let $0 < \delta \leq 0.5$ and $0 < \epsilon \leq 1$. There is a randomized, comparison-based, one pass streaming algorithm that, when processing a stream π consisting of n elements, produces a sketch satisfying the all-quantiles guarantee: for all queries $x \in \mathcal{U}$ simultaneously, the sketch returns an estimate $\widehat{\text{rank}}_\pi(x)$ such that with probability $1-\delta$,*

$$|\widehat{\text{rank}}_\pi(x) - \text{rank}_\pi(x)| \geq \epsilon \cdot \text{rank}_\pi(x),$$

where the probability is over the internal randomness of the streaming algorithm. The total space used by the sketch is

$$O\left(\epsilon^{-1}\log(\epsilon n)\cdot\log(1/\epsilon)\cdot(\log\log n+\log(1/\epsilon))\cdot\left(\log\log\left(\frac{\log(\epsilon n)}{\delta\epsilon}\right)\right)^3\right)$$

1.1 Further Related Works

Deterministic Sketches. In the deterministic additive-error setting, Greenwald and Khanna constructed the GK sketch that stores $O(\epsilon^{-1}\log(\epsilon n))$ elements [9]; more recently, [6] showed that the GK sketch is optimal and [10] gave a simplification of the GK sketch which still achieves optimal space. In the relative-error case, Zhang et al. [19] proposed a deterministic algorithm that uses $O(\epsilon^{-1}\log^3(\epsilon n))$ space. This algorithm maintains logarithmically many sketches based on the *chronological order* of the elements, and keeps merging sketches with similar sizes. Currently, the best known lower bound is $\Omega\left(\frac{\log^2(\epsilon n)}{\epsilon}\right)$ [6].

Sketches with Known Universe. Additionally, some works focus on the case when the universe $\mathcal{U}$ is known in advance to the streaming algorithm [5, 17]. In the additive error regime, the classical *q-digest* algorithm gave an optimal deterministic quantile summary using $O(\epsilon^{-1}\log|\mathcal{U}|)$ words of memory. A recent work of [12] improved this bound to $O(\epsilon^{-1})$ words, achieving an optimal space if the stream length $n \leq \text{poly}(|\mathcal{U}|)$. For the relative error setting, [5] designed a deterministic bq-summary algorithm using $O\left(\frac{\log(\epsilon n)\log|\mathcal{U}|}{\epsilon}\right)$ words of memory, while the offline lower bound is only $\Omega\left(\frac{\log \epsilon n}{\epsilon}\right)$ words.

Resizable Sketches. During the past ten years, there has been a flurry of works on "resizable sketches," wherein the goal is to design sketches that provide a fixed guarantee on the accuracy while allowing the space allocation to be dynamically adjusted throughout the runtime of the algorithm. This is especially important in practice, where the sketch size may start out being very small, but may need to grow sublinearly until it reaches some fixed maximum size. In particular, resizable sketches have been designed for filters [15, 7] [6]. Recently in [18], it was posed as an open question to design resizable sketches for the quantile estimation problem. In fact, we actually answer this question since our final sketch for the relative-error quantile estimation problem is actually resizable. Thus, we show that our sketch can achieve near-optimal space $\tilde{O}(\epsilon^{-1}\log(\epsilon n))$ while also having this practically-favorable "resizability" feature.

[6] In a talk on "Resizable Sketches" at the Simons Institute Workshop on "Sketching and Algorithm Design" in October 2023, it was mentioned that there are also expandable sketches for the k-minimum values problem and the well-known Misra-Gries sketch for deterministic heavy-hitter detection [18].

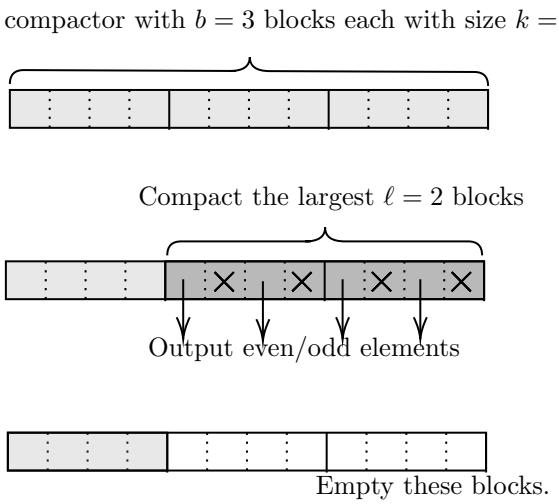

Figure 1: Compacting the largest ℓ blocks in a relative compactor.

2 Technical Overview

2.1 Relative Compactors Before describing our algorithm, it is helpful to have a quick overview of the (relative) compactors, which form the main building block for the recent relative-error quantile estimation algorithm of [3]. A compactor C takes an input stream π, keeps a small number of elements in memory, and outputs a substream π' of length at most $|\pi|/2$, while "preserving" the rank of any query x approximately:

$$\text{rank}_\pi(x) \approx 2 \cdot \text{rank}_{\pi'}(x) + \text{rank}_C(x). \tag{2.1}$$

That is, except for the few elements kept in the memory at the end of the stream, the compactor reduces the length of the stream by half. If we view the elements in the output as having *weight two*, then the rank of any x is preserved approximately. For *relative compactors*, we want the approximation in this building block to eventually lead to an overall "small relative error," i.e., the difference of the two sides of (2.1) needs to be smaller for x with a smaller rank.

To implement such a reduction in the stream length, a relative compactor maintains a *sorted array*. The array is divided into b blocks of size k, hence, of total size $s = k \cdot b$. The compactor keeps inserting elements from the stream π to the array. When the array is full, the algorithm performs a *compaction*, which empties a few blocks. To perform a compaction,

- the algorithm first decides the number of *blocks* ℓ to compact;
- then for the *largest* ℓ *blocks* (i.e., the largest $\ell \cdot k$ elements), it outputs either *all even-indexed elements* or *all odd-indexed elements* in those blocks, each with probability 1/2;
- finally, it removes all elements in those blocks.

See Figure 1 for an example. These compactions are crucial in order to make room for future insertions, yet they are the only source of errors in rank estimation. Consider any fixed query x and the error incurred to (2.1) during a compaction. If all ℓ blocks only contain elements *greater than* x, then clearly, this operation does not incur any error. Otherwise, since the compaction outputs every other element, it makes an error of at most 1 (in its absolute value). In fact, this only happens when an *odd* number of elements smaller than x is involved in the compaction, and the error is 1 or -1 with probability 1/2. At the very least, when this happens, the compaction has to involve at least one element that is smaller than x. We adopt the terminology of [3] and call the elements smaller than x, the *important elements*, and we call the compactions which involve important elements the *important compactions*. Note all these definitions are with respect to a fixed query x.

The crucial piece of the algorithm is to set the number of compaction blocks ℓ each time, so that for any fixed query x, the total error is small, or equivalently, there are only few important compactions. One way to do this

is to *sample ℓ from a geometric distribution* with $p = 1/2$ independently each time.[7] It is not hard to see that if we set the number of blocks $b \gg \log n$, then for each compaction, *conditioned on* it involving at least one element smaller than x, there is a constant probability that the compaction involves (at least) one entire block of elements smaller than x, due to the geometric distribution.[8] Intuitively, this implies that if a compaction incurs error, then it involves $\Omega(k)$ elements smaller than x on average. Since there are only $\mathrm{rank}_\pi(x)$ elements smaller than x, this intuition leads to an $O(\mathrm{rank}_\pi(x)/k)$ upper bound on the number of important compactions on average.[9] Since each such compaction incurs a ± 1 error independently, one can conclude that the difference on the two sides of (2.1) is typically

$$O(\sqrt{\mathrm{rank}_\pi(x)/k}). \tag{2.2}$$

Furthermore, for x with a very small rank, there is *no* important compaction with high probability, in which case, the error is zero.

Their final streaming algorithm simply maintains a chain of $O(\log n)$ relative compactors. The first compactor reads the input stream directly; the output of the i-th compactor is passed to the $(i+1)$-st compactor; the input to the last compactor is sufficiently short so that it never needs to perform compactions, hence, it has no output.

By (2.1), the rank of a query x is approximately reduced by half between the input and output of a relative compactor. The error analysis divides the chain of compactors into two parts based on the rank of x in the input of the compactor:

(a) for the first few compactors, when the query still has large rank, the *relative error* is tiny due to the square root in (2.2);

(b) after these compactors, the rank gets small enough so that there are no more important compactions – the error is zero.

It turns out that by setting $b = 2\log n$ and $k = \Theta\left(\frac{1}{\epsilon\cdot\sqrt{\log n}}\right)$, the space is $O(\epsilon^{-1}\log^{1.5} n)$, and the accumulated *relative error* from all compactors is at most $O(\epsilon)$.

To see this, consider a query of rank r, the relative error from the first compactor is $O(\sqrt{r/k}/r) = O(\sqrt{1/rk})$ by (2.2), the relative error from j-th compactor is $O(\sqrt{2^j/rk})$ due to the reduction in r. The errors form a geometric series, and the sum is dominated by the last term. Note because $b \gg \log n$, when the rank becomes smaller than $\frac{b}{2}\cdot k$, there is no important compactions with respect to this query with high probability. Hence, the last term in the geometric series has $r/2^j = \Theta(bk)$, the sum gives the total relative error $O(\sqrt{1/bk^2}) = O(\epsilon)$.

2.2 Our approach Our algorithm maintains $O(\log n)$ sketches, such that the i-th sketch is maintained for elements with rank roughly $\epsilon^{-1}\cdot 2^i$ with respect to the *input stream so far*. In particular, we will ensure that the sketches are in order: All elements in sketch i are smaller than those in sketch $i+1$. For each element from the input stream, we insert it into the sketch so that the ordering is preserved, i.e., we insert it into the largest sketch where the smallest element is smaller than the new element. We will also ensure that the sketches always store approximately the right number of elements: When a total number of $\Omega(\epsilon^{-1}\cdot 2^i)$ elements have been inserted into sketch i, i.e., sketch i "overflows," we will move the largest half of its elements to sketch $i+1$, so that sketch i only contains elements of current rank roughly $\epsilon^{-1}\cdot 2^i$ and all sketches are still in order.[10]

Now let us focus on one such sketch, and see what properties we will need from it. First, we can insert an element into the sketch, delete the largest elements, and can make approximate rank queries at the end. Since all sketches are in order, sketch i is only relevant for queries with rank $\Theta(\epsilon^{-1}\cdot 2^i)$. Hence, we can afford an additive error of $O(2^i)$ from sketch i. Note that while it looks similar to additive error quantile estimation, there is in fact

[7] This means we have $\Pr[\ell = t] = 2^{-t}$ for integer $t \geq 1$.

[8] When there is less than one entire block of important elements, the statement is technically not true. However, in this case, all these elements will be in the smallest block, and when $b \gg \log n$, there is a negligible probability that a compaction involves this block, i.e., we may assume the condition never happens.

[9] The work of [3] has another deterministic strategy for choosing ℓ with the same bound on the number of important compactions. The algorithm fixes a sequence of integers, called the *compaction schedule*, and chooses deterministically the next integer from the sequence as ℓ each time. We in fact build on this deterministic version. See Section 5.1 for an overview of the deterministic version.

[10] Note that the *sketches* will necessarily have to only store a subset of the inserted elements to save space, so some elements may have already been deleted when the time comes for them to be moved. We will make this step more explicit later in the overview.

a key difference – we may insert up to $n \gg \epsilon^{-1} \cdot 2^i$ elements to a single sketch. Even though we only need to focus on queries of rank $O(\epsilon^{-1} \cdot 2^i)$, there may actually be much more elements inserted in total. This key difference makes the sketches with $O(1/\epsilon)$ space [13] with additive error inapplicable to this setting.

Consider the i-th sketch, observe that we can always first sample each insertion to it with probability $\Theta(\epsilon^{-1} \cdot 2^{-i})$. This is because for a query of (final) rank $\Theta(\epsilon^{-1} \cdot 2^i)$, its rank after the sampling would be roughly $\Theta(1/\epsilon^2) \pm O(1/\epsilon)$, i.e., the relative error incurred due to sampling is $O(\epsilon)$. By maintaining the *smallest* $O(1/\epsilon^2)$ elements that survive the sampling, one could answer all such queries with small required error. This is the main idea of [20] with space $O(\epsilon^{-2} \cdot \log n)$, but it has suboptimal dependence on ϵ.

To further improve the space, we could maintain a chain of $O(\log 1/\epsilon)$ relative compactors for all *sampled elements* with $k = \Theta\left(\frac{1}{\epsilon \cdot \sqrt{\log n}}\right)$ and $b = O(\log n)$. That is, we feed all sampled elements to the first compactor, and connect the output of compactor j to the input of compactor $j+1$. By the same error analysis as in the last subsection, for a query of rank roughly $r = \Theta(1/\epsilon^2)$ (after sampling), the total relative error incurred by the compactors is

$$\sqrt{1/rk} + \sqrt{2/rk} + \sqrt{2^2/rk} + \cdots + \sqrt{1/bk^2} = O(\epsilon).$$

Note that when the sketch reaches its capacity, we can simply move the largest elements from all compactors to the next sketch. This allows us to delete the largest elements. However, since we set the parameters of each compactor same as before (this is necessary, since otherwise the relative error just from the last compactor would already be more than ϵ), and there will be more than one relative compactor for each of the $\log n$ sketches, this does not give an improvement as is.

Although this data structure still uses $\tilde{\Omega}(\epsilon^{-1} \log^{1.5} n)$ space, this reformulation of the algorithm in [3] facilitates the space improvement in our work. In particular, we observe the following.

- Let n_i be the number of insertions to the i-th sketch. Inside the relative compactors of the i-th sketch, we need really $b = \Theta(\log n_i)$ instead of $\Theta(\log n)$ blocks. In order to force the i-th sketch to actually use *all* $\Theta(\epsilon^{-1} \cdot \sqrt{\log n})$ space, we must insert $n_i = n^{\Theta(1)}$ elements into it.

- Suppose we indeed frequently insert many elements to sketch i, then this makes the task of later sketches easier. This is because for every $O(\epsilon^{-1} \cdot 2^i)$ insertions to sketch i, we will have to move the largest half to the next sketch $i+1$.[11] Note that these elements are smaller than any element in sketch $i+1$. Also, when this happens $O(1)$ times, sketch $i+1$ must have reached its capacity, and will have to move the largest half of its elements to sketch $i+2$.

  In particular, this means that all elements that were previous stored in sketch $i+1$ will be "pushed away" to later sketches after sketch i overflows a constant number of times. As original elements in sketch $i+1$ are pushed away, we can now *effectively start the count n_{i+1} from zero again* and use a smaller space for sketch $i+1$.

Combining these two observations, we argue that not *all sketches need as much as $\Theta(\epsilon^{-1} \cdot \sqrt{\log n})$ space.* If there is a large number and very frequent insertions to sketch i, then for a large number of sketches after i, they would only need $b = O(1)$ and $O(1/\epsilon)$ space. As a proof-of-concept, consider an input stream such that between any two adjacent insertions to sketch i, there are roughly β_{i+1} insertions to sketch $i+1$. Then by the above argument, before all current elements in sketch $i+1$ are pushed away to $i+2$ due to the overflows from sketch i, there will be only roughly $\beta_{i+1} \cdot \epsilon^{-1} \cdot 2^{i+1}$ new insertions to sketch $i+1$. It turns out that we can maintain a smaller sketch $i+1$ by setting the number of blocks b to $\Theta(\log \beta_{i+1})$, and a sketch of size $O(\epsilon^{-1} \cdot \sqrt{\log \beta_{i+1}})$ would be sufficient. Since the stream length is n, implying that $\prod_i \beta_i \approx n$, the total sketch size is $\sum_i O(\epsilon^{-1} \cdot \sqrt{\log \beta_i}) \le \sum_i O(\epsilon^{-1} \cdot \log \beta_i) \le O(\epsilon^{-1} \cdot \log n)$.

To carry out the full details for general streams, we face the following concrete challenges.

- The stream may insert new elements to a sketch at different rates at different times. It may even temporarily stop inserting into a sketch, and resume at a later time. This means that the sketch sizes need to actively and dynamically change over time based on how frequent the insertions to the sketches currently are.

[11] As sketch i only has a bounded memory, we cannot remember exactly the largest half of elements inserted and move them to the next sketch $i+1$. However, we can move our current sketch of the largest half from sketch i to sketch $i+1$. In fact, it turns out to be easy to integrate this sketch into sketch $i+1$, as it only has error 2^i, smaller than that of sketch $i+1$.

- We have no way to predict the insertion frequencies to different sketches, yet we need to allocate the *fixed amount of total space* to different sketches so that each of them has small error as required.

In the following two subsections, we give an overview of how we tackle the challenges.

2.3 Elastic Compactors It turns out that the relative compactors can naturally be made resizable: when we need to downsize it to make space for other sketches, we simply do a compaction to empty the largest blocks, and then resize the array. The main question then is under what conditions we can have the same error bound as before.

To be more specific, we introduce elastic compactors. In addition to what a relative compactor can do, the user may request an elastic compactor to resize to a given size, possibly after every element from the input stream. Elastic compactors are constructed based on relative compactors. We fix the block size k, and let the array resize by only adjusting the number of blocks b (this turns out to be sufficient for our application). Let us assume after every insertion, the user may specify a new number of blocks b_j that the array has to resize to.

- To resize, if the array has at most kb_j elements, then the new size is large enough to contain all elements, we simply set the number of blocks to b_j.
- Otherwise, the algorithm samples the number of blocks to do compaction from a geometric distribution *conditioned on* having at most b_j nonempty blocks, i.e., it compacts the largest $b_{j-1} - b_j$ blocks with probability $1/2$, the largest $b_{j-1} - b_j + 1$ blocks with probability $1/4$, etc.

Note that when b_j remains the same for all j, this is the same as relative compactors. The error analysis is also similar. Consider any fixed query x, and consider a compaction. Conditioned on the compaction involving at least one element smaller than x, there is a constant probability that it involves at least one entire block of k elements. The only exception is when the compaction decides to empty the entire array. We discussed in the review of relative compactors that when $b \gg \log n$, this does not happen except with negligible probability. A simple calculation shows that as long as $\sum_j 2^{-b_j} \ll 1$, we have the same guarantee. Hence, the same bound on the number of important compactions holds, and so does the error bound.

In other words, throughout our algorithm, suppose we require the compactor to resize to size $s_1, s_2, \ldots, s_\ell$, then as long as

$$\sum_{j=1}^{\ell} 2^{-s_j/k} \ll 1, \tag{2.3}$$

the error bound of (2.2) holds. Such elastic compactors can be used to construct the $O(\log n)$ sketches we discussed above. It turns out that by replacing the $O(\log 1/\epsilon)$ relative compactors in each sketch by elastic compactors and setting $k = O(1/\epsilon)$, the sketches can also be resized (with a factor of $O(\log 1/\epsilon)$ more space than an elastic compactor) while having the same error guarantee as before.[12]

2.4 Allocating space As we discussed earlier, the stream may insert elements to each sketch at different frequencies at different times. Therefore, we may have to allocate a different amount of space to each sketch at different times. To demonstrate the space allocation strategy, let us first consider the following special case.

A tree instance. Suppose the input stream inserts elements in the following way.

- It inserts a batch of $n^{0.1}$ elements to the 1st sketch almost consecutively, except that it may "pause" at any 100 time points in the middle.
- During each "pause," the stream inserts a batch of $n^{0.1}$ elements to the 2nd sketch almost consecutively, and it may again "pause" 100 times during each batch.
- During each "pause" in inserting elements to sketch i, the stream inserts $n^{0.1}$ elements to sketch $i+1$ and later sketches recursively. We recurse for $i = 1, 2, \ldots, 0.1 \log n$. At the end of the recursion when $i = 0.1 \log n$, we just inserts $n^{0.1}$ elements to sketch $i+1$ but not later sketches.

[12] A single elastic compactor may use more space than before by setting $k = O(1/\epsilon)$. In the worst case, it may use space as much as $O(\epsilon^{-1} \log n)$, but we will show that the total space is bounded.

Let us also assume that *there are sufficiently many insertions to sketch i between adjacent pauses of sketch i*. In this case, the stream has a tree structure: the root corresponds to the insertion (sub)stream to sketch 1, each child of the root corresponds to an insertion substream to sketch 2 during each pause in the root, etc. When there are sufficiently many insertions between adjacent pauses, different subtrees do not interfere with each other.

For such streams, we must drastically resize its sketches during each pause. This is because during each pause in inserting to sketch i, the stream only inserts elements to larger sketches. Hence, when the pause ends, the elements that sketch i are supposed to maintain remain unchanged, and the insertions to sketch i will continue from there. Effectively, we are inserting $n^{0.1}$ elements to it, which forces the sketch to use $\Omega(\epsilon^{-1} \cdot \sqrt{\log n})$ space *if we do not resize it* during the pause. It leads to a total space of $\Omega(\epsilon^{-1} \cdot \log^{1.5} n)$ at the bottom of the recursion, when we have $0.1 \log n$ levels of pauses happening at the same time.

Our solution to this special case is to downsize the sketches during its pauses according to its length and the length of its current child. That is, in the above tree view of the stream, let us consider the length of the stream corresponding to each node. For a node u with length l_u and its child v with length l_v, we will resize the sketch maintained for u to $\tilde{O}(k \log(l_u/l_v))$ during the pause at v (for now, let us assume that we know these lengths in advance). This guarantees that the requested space sequence satisfies (2.3), since $\sum_{v:\text{child of } u} l_v < l_u$. Hence, the sketches provide the same error guarantee as before. Importantly, the total space of all sketches is always bounded by $O(k \log n) = O(\epsilon^{-1} \log n)$, because the spaces allocated to the sketches form a telescoping sum and the length of the root is n.

General streams. Our final algorithm, which works for general streams, uses a similar idea. In the above tree example, adjacent children of a node are essentially independent because we assumed that sufficiently many *small* elements are inserted in between. We first make an analogue of it for general streams. Consider sketch i and a time t_1. Suppose sketch $i-1$ overflows a constant number of times between t_1 and some later time t_2, then all elements in sketch i at t_1 will have been pushed away to later sketches. This effectively *resets* sketch i, making the sketch at time t_2 "independent" of the sketch at time t_1. Hence, (for the analysis) we will divide the timeline of sketch i into intervals of various lengths based on its "resets," i.e., we start a new interval when sketch i resets. These intervals will correspond to (the substreams of) the nodes in the tree instance, but they do not form an exact tree structure as before. However, we observe that each interval of sketch i only intersects $O(1)$ intervals of sketch $i-1$ (this is because when sketch $i-1$ resets $O(1)$ times due to the overflow of sketch $i-2$, it must also have overflown $O(1)$ times itself, causing sketch i to reset). Hence, we can view each interval of sketch i as having $O(1)$ "parents," i.e., the intervals of sketch $i-1$ it intersects with. We inductively assign a weight to each interval, generalizing the length, such that the weight of an interval I is the sum of all weights of intervals that have I as one parent, and the weight of an interval of the last sketch is one. Now suppose we know the weights of all intervals in advance. Then when sketch i is currently in an interval u with weight w_u and sketch $i+1$ is in interval v with weight w_v (note that then, u is a parent of v), we allocate space $O(k \log(w_u/w_v))$ to sketch i. A similar analysis gives the same error bound. By using the fact that the weight of any interval is at most $n^{O(1)}$, since each interval has $O(1)$ parents, the total space is again $O(k \log n)$ by a telescoping sum.

To calculate the weights, we need full information about the stream, including how the intervals intersect in the future. We show that this is not necessary, the same algorithm works (up to factors of $\log \log n$ and $\log 1/\epsilon$) even if we simply use the interval intersection information so far to calculate the weights. See Section 4.4 for more details.

3 Preliminaries

All logarithms in this paper are base 2. Without loss of generality, we will also assume that $1/\epsilon, n$, and $1/\delta$ are all powers of 2.

The Comparison-based Model. In this work, we focus on the *comparison-based model*: this means that at any time t, the memory of the streaming algorithm is a tuple $(\mathcal{M}_t, I_t)$ where $\mathcal{M}_t$ is a subset of the stream elements and at any time, the algorithm can only compare two elements in $\mathcal{M}_t$. I_t contains auxiliary information which is stored by the algorithm (e.g. information about previous comparisons, etc). Note that the memory usage of the algorithm is measured by the size of $|\mathcal{M}_t|$ only.

Probability. We will use the following probability fact in our analysis.

FACT 3.1. *For any two (not necessarily independent) random variables X and Y, when we add them, their*

standard deviation at most adds, that is,

$$\mathsf{E}[(X+Y)^2]^{1/2} \le \mathsf{E}[X^2]^{1/2} + \mathsf{E}[Y^2]^{1/2}.$$

Proof.

$$\begin{aligned}\mathsf{E}[(X+Y)^2] &= \mathsf{E}[X^2] + \mathsf{E}[Y^2] + 2\mathsf{E}[XY] \\ \text{(by Cauchy-Schwarz)} \qquad &\le \mathsf{E}[X^2] + \mathsf{E}[Y^2] + 2\mathsf{E}[X^2]^{1/2}\mathsf{E}[Y^2]^{1/2} \\ &= \left(\mathsf{E}[X^2]^{1/2} + \mathsf{E}[Y^2]^{1/2}\right)^2.\end{aligned}$$

□

We will also need properties of subgaussian random variables. A mean-zero variable X is σ^2-subgaussian if $\mathsf{E}[e^{\lambda X}] \le e^{\frac{\lambda^2\sigma^2}{2}}$ for all $\lambda \in \mathbb{R}$. If a random variable X is not mean-zero, we say it is σ^2-subgaussian if the mean zero variable $(X - \mathsf{E}[X])$ is σ^2-subgaussian.

FACT 3.2. *If X is σ_1^2-subgaussian and random variable Y is an independent σ_2^2-subgaussian. Then $X+Y$ is $\sigma_1^2+\sigma_2^2$-subgaussian.*

4 Description of the Algorithm

Before we present our full algorithm, we first provide a high-level overview of our strategy.

High-level Overview. Recall the definition of an ϵ-relative-error quantile sketch: for any query $x \in \mathcal{U}$, after processing a stream π the sketch must return an estimate $\widehat{\text{rank}}_\pi(x)$ such that

$$|\widehat{\text{rank}}_\pi(x) - \text{rank}_\pi(x)| \le \epsilon \cdot \text{rank}_\pi(x)$$

holds with probability at least $1-\delta$. Importantly, observe that (up to a factor of 2) the relative error guarantee is equivalent to the following: for all queries x such that $\text{rank}(x) \in [\epsilon^{-1}\cdot 2^{i-1}, \epsilon^{-1}\cdot 2^i]$, the answer of the sketch can tolerate an *absolute* error of at most 2^i.

The general strategy of our algorithm naturally follows from the above observation: we decompose the relative-error quantile estimation problem into $\lceil \log_2(\epsilon n)\rceil$-many absolute-error quantile problems. In particular, we partition the rank-space into $\lceil \log_2(\epsilon n)\rceil$ *ranges*, where range i contains all inserted elements with rank (roughly) between $[\epsilon^{-1}\cdot 2^{i-1}, \epsilon^{-1}\cdot 2^i]$ with respect to all elements inserted so far. Then, we maintain a separate sub-sketch H_i for each range i and maintain the guarantee that each sub-sketch H_i will have absolute error of at most 2^i (See Figure 2). For each new element x_t that appears in the stream, we find the range i such that x_t is smaller than all elements in H_{i+1} and larger than or equal to all elements in H_{i-1}, and we insert x_t into that sub-sketch H_i. At any point in the stream, if H_i contains many more than $O(\epsilon^{-1}\cdot 2^i)$ elements, we remove the largest elements from H_i and insert them into the next sub-sketch H_{i+1}.

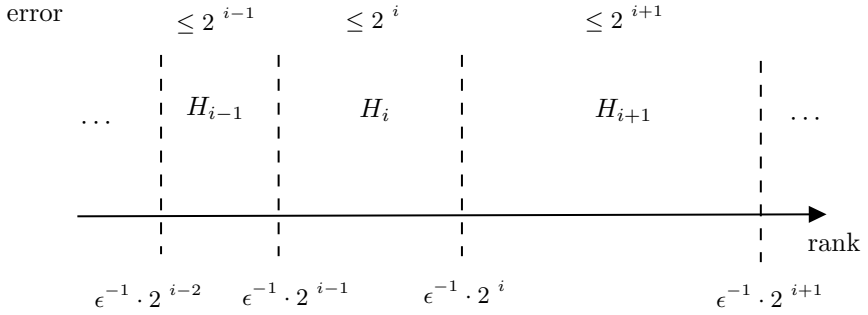

Figure 2: Our basic strategy.

Observe that in this construction, each sub-sketch H_i is different from an (insertion-only) additive-error quantile sketch, as we are periodically deleting the largest elements in H_i and moving them into H_{i+1}. We call

such a sub-sketch H_i a *top-quantiles sketch.*[13][14] Moreover, to achieve a total space of $\widetilde{O}(\epsilon^{-1}\log(\epsilon n))$, on average we can only afford to allocate $\widetilde{O}(1/\epsilon)$ space per sub-sketch H_i. However, as we show in Lemma 4.3, there is a $\Omega\left(\epsilon^{-1}\cdot\sqrt{\log(\epsilon n)}\right)$ space lower bound for any such top-quantiles sketch H_i. This becomes the main technical barrier of our work.

To get around this barrier, we make the following key observation: this lower bound says that for each sub-sketch, there *exists* an input stream that forces the sketch to use maximum space *at some time point* during the execution of the algorithm. However, the lower bound in Lemma 4.3 does not imply the existence of a single hard input stream that simultaneously forces *every sub-sketch* H_i for rank interval $[\epsilon^{-1}\cdot 2^{i-1}, \epsilon^{-1}\cdot 2^i]$ to achieve its maximum space usage. As a result, there is still hope to design a near-optimal $\widetilde{O}(\epsilon^{-1}\log(\epsilon n))$ algorithm.

Our data structure H_i is based on the relative compactor from the previous relative error quantile sketch of [3]. There are two main deviations of our algorithm from previous work: (1) as discussed earlier, we maintain a separate data structure (or "sub-sketch") for stream elements that are roughly in rank range $[\epsilon^{-1}\cdot 2^{i-1}, \epsilon^{-1}\cdot 2^i]$ for each $i = 1, ..., \lceil\log(\epsilon n)\rceil$, and (2) our sketch is made up of *elastic* compactors which are *dynamically resizable*. In particular, our sub-sketches can change their space usage dynamically based on "demand": for example, if the number of insertions into range i is very high, we may choose to strategically allocate more space to H_i while allocating less space to data structures for other ranges. Using this idea, the main task becomes to define a scheme to dynamically allocate space to the sub-sketches and constantly resize them. Surprisingly, this approach results in a reduced total space of $\widetilde{O}\left(\epsilon^{-1}\cdot\log(\epsilon n)\right)$ without contradicting the aforementioned lower bound.

The rest of this section is organized as follows.

- In Section 4.1, we will introduce the most basic component of our resizable sketch, which constitutes our *elastic compactor*. However, we defer the detailed implementation of this data structure to Section 5.
- Then, in Section 4.2, we will explain how to construct each sub-sketch H_i from these elastic compactors.
- Finally, in Section 4.3, we describe the strategy for dynamically allocating space to different sub-sketches H_i, which finishes the high-level overview of our algorithm.

Throughout this section, we only provide bounds on the expected squared error, and some proofs are deferred to Section 6.1. For the high probability bounds, we refer the reader to Section 6.2.

4.1 Elastic Compactors The most basic building block of our algorithm is what we call an *elastic compactor*. Specifically, this is a compactor whose size can be dynamically adjusted. For our case, we will eventually choose (and dynamically adjust) the size of the elastic compactor in an online manner, based on the input stream. See Table 1 for the definition.

Error Guarantee. For any query $x \in \mathcal{U}$, we define the error associated to x by

$$\Delta(x) \coloneqq \left(\textsc{Rank}_C(x) + 2\cdot\text{rank}_{\pi'}(x)\right) - \text{rank}_\pi(x) \tag{4.4}$$

We enforce the following guarantee for elastic compactors.

LEMMA 4.1. *Let π be the input stream and $s_1, s_2, \ldots, s_\ell$ be the sequence of space parameters after each resize/insert operation. There is a randomized elastic compactor C such that, if we condition on*

$$\sum_{t=1}^{\ell} 2^{-s_t/k} \le 0.5, \tag{4.5}$$

for any query $x \in \mathcal{U}$, C achieves expected squared error

$$\mathsf{E}\left[\Delta(x)^2\right] \le \frac{\text{rank}_\pi(x)}{k},$$

where the expectation is taken over the randomness of the compactor.

[13] In fact, we formalize this as a separate problem of independent interest (See Section 4.2).

[14] Note that the name "Top Quantiles" refers to the task of estimating the *bottom* ranks. This is generally in contrast with the rest of our work, in which we consider $\text{Quantile}(x) = \text{rank}(x)/n$.

Elastic Compactor

Parameters. Let C be an (s, k)-elastic compactor: C may store at most s elements in memory, and let k be a parameter related to the total error allowed for C.

Input: A stream π of elements in $\mathcal{U}$.
Output: As a result of each *compaction*, C can output elements to a *compacted* stream π'. In the end, π' contains at most $|\pi|/2$ elements.

Operations:

- $\textsc{Resize}(C, s')$: Expand or compress the space of the compactor C to s' and set $s \leftarrow s'$.

  Whenever a resize operation compresses the space to a smaller size (i.e. $s' < s$), we will perform compaction, and some elements stored in C will be outputted to the output stream π'.

- $\textsc{Insert}(C, x_1, x_2, \ldots, x_m)$: Add $x_1, x_2, \ldots, x_m$ to the input stream π of compactor C, where $m \leq s$. We will implement it using $\textsc{Resize}(C, s')$:

  – First, we call $\textsc{Resize}(C, 2s)$, insert $x_1, x_2, \ldots, x_m$ into the new open space of size s in memory.

  – Then, we call $\textsc{Resize}(C, s)$ to compress the space back to the original size.

  Note that some elements may be outputted to the output stream π' during this last compression.

- $\textsc{Rank}_C(x)$: Return the rank of x among the elements π that are still in the memory of compactor C.

  Let $\widehat{\text{rank}_\pi}(x)$ denote the estimated rank of x with respect to stream π. Then, we define $\widehat{\text{rank}_\pi}(x) = \textsc{Rank}_C(x) + 2 \cdot \text{rank}_{\pi'}(x)$.

Table 1: Elastic Compactor.

Intuitively, Equation (4.5) ensures that the space used by the compactor cannot be too small. Note that we modify the implementation of the relative compactor of [3] in our definition of the elastic compactor. We defer the proof of Lemma 4.1 to Section 5.

Additional operations. Finally, we define two more operations for our elastic compactor, which we call "reset" and "remove" operations. These will be required in Section 5.

- $\textsc{Reset}(C)$: This operation resets the sum in Equation (4.5) that we have accumulated so far to zero, while maintaining all the elements that are stored in C. Essentially, by resetting the sum, we allow the compactor to handle more operations, but at the cost of introducing more error into the rank estimate.
- $\textsc{Remove}_{\max}(C)$: This operation removes the largest element in the compactor C and outputs it to another stream π_{remove}. When this operation is present, we define the error attributed to $x \in \mathcal{U}$ to be

$$\Delta(x) \coloneqq \left(\textsc{Rank}_C(x) + 2 \cdot \text{rank}_{\pi'}(x) + \text{rank}_{\pi_{\text{remove}}}(x)\right) - \text{rank}_{\pi}(x). \tag{4.6}$$

Suppose we fix a query $x \in \mathcal{U}$. Then, we call a reset *important* if at the time of reset, there is at least one element in the compactor C that is smaller than x. Formally, we have the following guarantee.

Lemma 4.2. *Suppose between any two adjacent resets, the sequence of space constraints in the resize operations satisfies Equation* (4.5)*. Let S be an upper bound on the number of elements in the compactor at any time. Then for any query $x \in \mathcal{U}$, if there are t important resets, the randomized compactor achieves expected squared error*

$$\mathsf{E}\left[\Delta(x)^2\right] \le \frac{\text{rank}_{\pi}(x) + t \cdot S}{k}.$$

Here $\Delta(x)$ is defined as in Equation (4.6)*.*

The proof of Lemma 4.2 is also deferred to Section 5.[15]

4.2 The Top Quantiles Sketch As mentioned before, our basic strategy is to maintain sub-sketches H_i's that each handles elements of rank roughly around $R_i \coloneqq \epsilon^{-1} \cdot 2^i$ and has absolute error less than 2^i. The main challenge of this task can be formalized as the following natural problem.

The Top-R Quantiles Problem.

Input. A stream π of length $|\pi| \gg R$.
Output. For any query $x \in \mathcal{U}$ (given at the end of the stream) with the promise that $\text{rank}_{\pi}(x) \le R$, the sketch has to output an estimate $\widehat{\text{rank}}_{\pi}(x)$ such that,

$$\left|\text{rank}_{\pi}(x) - \widehat{\text{rank}}_{\pi}(x)\right| \le \epsilon \cdot R.$$

Table 2: The Top Quantiles Problem.

Note when input stream length $|\pi| \le R$, this problem can be solved by a standard additive-error quantile sketch, because the additive-error sketch will incur error at most $\epsilon \cdot |\pi| \le \epsilon \cdot R$. However, this becomes a different problem when $|\pi| \gg R$. In fact, this is evidenced by the fact that the additive-error quantile sketch takes only $O(\epsilon^{-1} \log\log \delta)$ space (KLL sketch [13]), while for this problem, as mentioned before, we have the following lower bound saying $\Omega(\epsilon^{-1}\sqrt{\log n})$ space is necessary (for a specific ϵ) to achieve failure probability $\delta < 0.1$.

Lemma 4.3. *Let $n = |\pi|$ be the stream length. When $\epsilon = 1/\sqrt{\log n}$, any comparison-based algorithm that solves the top quantiles problem for $R = \log n$ with $\delta < 0.1$ failure probability requires at least $\Omega(\log n)$ space.*

[15] In fact, Lemma 4.1 is a direct application of Lemma 4.2 when $t = 0$.

The proof of Lemma 4.3 is deferred to Section 7. Note that there are some natural motivations for studying this problem; e.g. when monitoring system latency, we may only care about an abnormal tail of the data and wish to construct sketches that incur the smaller error $\epsilon \cdot R$ instead of $\epsilon \cdot |\pi|$.

Outline of our approach. First, we design a resizable sketch H which solves the top-R quantiles problem. Later on, we will augment this sketch H with additional features in order to obtain our final sub-sketches H_i's, which will be used to solve the overall relative error quantile estimation problem.

Compactor Hierarchy. Similar to the KLL sketch [13], our sketch H is a hierarchy that consists of a sampler, a chain of elastic compactors $C_1, ..., C_{\log(1/\epsilon)-1}$, and finally a buffer B which stores $O(1/\epsilon)$ elements, as shown in Figure 3 below.

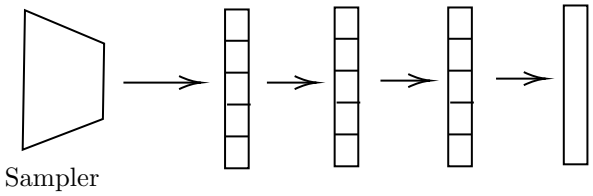

Figure 3: Structure of H

Note that our resizable sketch H has two tuneable parameters: an error parameter $\epsilon > 0$, and a space parameter s_H (eventually, the total space used by sketch H will be $s_H \cdot \lceil \log_2(1/\epsilon) \rceil$). Next, we define the same set of operations for H (which consists of a sampler followed by a hierarchy of elastic compactors), just as we defined earlier for each elastic compactor C:

- $\textsc{Insert}(H, x)$: Add x into the input stream π of H.
- $\textsc{Resize}(H, s')$: Expand or compress the space of H so that we set the space parameter to be $s_H \leftarrow s'$.
- $\textsc{Rank}_H(x)$: Return an estimate for the rank of x. To compute the estimate $\textsc{Rank}_H(x)$, we output the total weight of the elements in H that are smaller than x. That is,

$$\textsc{Rank}_H(x) \coloneqq \sum_{j=0}^{\log(1/\epsilon)-1} 2^j \cdot \epsilon^2 R \cdot \textsc{Rank}_{C_j}(x) + \epsilon R \cdot \textsc{Rank}_B(x) \tag{4.7}$$

where $\textsc{Rank}_{C_j}(x)$ and $\textsc{Rank}_B(x)$ denote the rank of x among all elements contained in compactor C_j and buffer B, respectively.

Recall that for all x with $\text{rank}_\pi(x) \le R$, we want

$$|\textsc{Rank}_H(x) - \text{rank}_\pi(x)| \le O(\epsilon) \cdot R.$$

We now describe the operation of each component as the stream travels through the sketch H from left to right. Additionally, for every element x stored in the memory of H, we define the *weight* of x to represent the number of stream elements that are "represented" by x.

1. **Sampler**: first, each element x in input stream π is passed through the sampler: with probability $\frac{1}{\epsilon^2 R}$, the sampler inserts x into the first elastic compactor C_0, and x is discarded otherwise. Note that if there are $|\pi| = n$ elements in the input stream, roughly $\ell = \frac{n}{\epsilon^2 R}$ elements will be inserted into C_0.

   Additionally, when element x first appears in the stream π, x has weight 1. After passing through the sampler, each sampled element carries weight $\epsilon^2 R$.

2. **Elastic compactors**: for each $0 \le j < \log_2(1/\epsilon)$, the elastic compactor C_j has block-size parameter $k = 1/\epsilon$ and space parameter $s = s_H$. After each operation to C_j, it might output m ($m \le s$) element $x_1, x_2, \ldots, x_m$. We call $\textsc{Insert}(C_{j+1}, x_1, x_2, \ldots, x_m)$ to insert them into the next compactor. Eventually, the output of the last compactor is inserted into the buffer.

   Note that for each $0 \le j < \log_2(1/\epsilon)$, elements in C_j all have weight $\epsilon^2 R \cdot 2^j$.

3. **Buffer**: the buffer B stores the smallest $1/\epsilon$ elements inserted to it, and throws away the largest element whenever it is full.

   Each element in the buffer has weight $\epsilon \cdot R$.

Intuitively, as the smallest $1/\epsilon$ elements in the buffer already have total weight R, and we only care about top-R quantiles, this explains why we can afford to throw away all other elements from the buffer.

REMARK 4.4. *For the corner case when $R < \frac{1}{\epsilon^2}$, we omit the sampler and directly insert stream elements into the compactor C_j such that $2^j \cdot \epsilon^2 R = 1$.*

Implementation of Operations. For INSERT(H, x), we feed x to the sampler, which inserts x into C_0 with probability $\frac{1}{\epsilon^2 R}$. (See Algorithm 1.) To simplify the notation, we will define $C_{\log(1/\epsilon)}$ to be the buffer B at the last level of the hierarchy.

Algorithm 1 Insert an element.

```
1: procedure INSERT(H, x)
2:     with probability 1/(ε²R) do
3:         INSERT(C_0, x).
4:         for i = 0, 1, ... log(1/ε) − 1 do
5:             Let x_1, x_2, ..., x_m be the elements that C_i outputs after insertion.
6:             INSERT(C_{i+1}, x_1, x_2, ..., x_m).
```

Likewise, we insert elements into the buffer B (i.e. the last compactor $C_{\log(1\epsilon)}$) as follows. Here, we note that if the new insertion causes there to be more than $1/\epsilon$ elements stored in the buffer, we remove the largest elements until there are only $1/\epsilon$ remaining.

Algorithm 2 Insert elements to buffer B.

```
1: procedure INSERT(B, x_1, x_2, ..., x_m)
2:     Add x_1, x_2, ..., x_m to the buffer B.
3:     while the buffer B has more than 1/ε elements do
4:         Remove the largest elements in the buffer.
```

Now, to implement RESIZE(H, s') on the full sub-sketch H, we simply call RESIZE(C_j, s') for each level j of the compactor hierarchy (See Algorithm 3).

Algorithm 3 Resize the hierarchy.

```
1: procedure RESIZE(H, s')
2:     RESIZE(C_0, s').
3:     for j = 0, 1, ... log(1/ε) − 1 do
4:         Let x_1, x_2, ..., x_m be the elements that C_i outputs after insertion.
5:         INSERT(C_{j+1}, x_1, x_2, ..., x_m).
6:         RESIZE(C_{j+1}, s') unless j = log(1/ε) − 1. (The buffer is always of fixed size 1/ε.)
```

Error Guarantee. In conclusion, we have the following lemma.

LEMMA 4.5. *Let π be the input stream and $s_1, s_2, \ldots, s_\ell$ be the sequence of space parameters after each resize / insertion from the sampler into C_0. Our randomized top-R quantiles sketch H, conditioning on*

$$\sum_{t=1}^{\ell} 2^{-\epsilon \cdot s_t} \leq 0.25, \tag{4.8}$$

for any query x with $\mathrm{rank}_\pi(x) \leq R$, achieves standard deviation

$$\mathsf{E}\left[|\mathrm{RANK}_H(x) - \mathrm{rank}_\pi(x)|^2\right]^{1/2} \leq O(\epsilon) \cdot R.$$

We defer the proof to Section 6.1. The main idea is to apply Lemma 4.1 to analyze the error introduced by each C_j.

REMARK 4.6. *Let $n = |\pi|$. Then there are in expectation $\ell = \frac{n}{\epsilon^2 R}$ many insertions into C_0. Setting $s_1 = s_2 = \cdots = s_\ell = \epsilon^{-1} \log \ell = O\left(\epsilon^{-1} \log n\right)$ gives an $O\left(\epsilon^{-1} \log n\right)$ space algorithm for the top-R quantiles problem. We will later improve it in Section 7. By setting $s_1 = s_2 = \cdots = s_\ell = O(\epsilon^{-1}\sqrt{\log n})$, we obtain an optimal algorithm for Top-R Quantiles problem, matching the lower bound in Lemma 4.3. For this section, the simpler bound in Lemma 4.5 suffices.*

Final Description of Sub-sketch H_i. Finally, we augment the top-R quantiles sketch H described above to obtain the final version of our sketches H_i, which we will use to build our final algorithm for the relative error quantile estimation problem. In particular, we add three additional features, which are described in detail below. Note that we denote $R_i \coloneqq \epsilon^{-1} \cdot 2^i$ and let π_{H_i} be the input stream of H_i.

1. **Limiting the maximum total weight in H.** In our algorithm, we will always ensure that the total weight of elements in each H_i is less than $3R_i$. Also, we will set the buffer size to $\frac{3}{\epsilon}$. Since each element in the buffer has weight ϵR_i, the buffer will never be full (as long as the total weight is bounded by $3R_i$). Thus, each subsketch H_i no longer needs to throw away elements when the buffer is full (Remove Line 4 from Algorithm 2).

   Instead, H_i operates as follows: whenever the total weight exceeds $3R_i$, the sub-sketch H_i outputs the largest element x stored in H_i into the output stream π'_{H_i}, with multiplicity equal to the *weight* of x. Then, we remove x from H_i (Note: the largest element x may not be in the final buffer B, rather it may be in one of the elastic compactors in the hierarchy). We repeat this removal process until the total weight of H becomes less than $2R_i$ again. See Algorithm 4 for the pseudocode. Observe that once the total weight (which is initially 0) exceeds R_i, it will always be within $[R_i, 3R_i]$.

Algorithm 4 Maintaining the total weight of H_i.

1: $R_i \leftarrow \epsilon^{-1} \cdot 2^i$
2: **if** total weight of elements in H_i exceeds $3R_i$ **then**
3: RESET(H_{i+1}). ▷ This line is for Section 4.3 and should now be ignored.
4: **repeat**
5: Suppose the largest element in H_i is in C_j $(0 \le j \le \log(1/\epsilon))$. ▷ Note this includes the buffer ($j = \log(1/\epsilon)$).
6: $\text{REMOVE}_{\max}(C_j)$ and let x be the largest element that C_j outputs. [16]
7: Output $2^j \cdot \epsilon^2 R_i$ many copies of x to an output stream π'_{H_i}. ▷ Later in Section 4.3, we will insert each element in π'_{H_i} to H_{i+1}.
8: **until** total weight of elements in H_i is less or equal to $2R_i$

   In the final algorithm, we will then insert the elements in π'_{H_i} into the next sub-sketch H_{i+1}. We observe that by repeatedly removing the largest elements from H_i and adding them into π'_{H_i}, we guarantee the following "ordering" property:

   OBSERVATION 4.7. *The elements in the output stream π'_{H_i} of H_i are naturally in non-decreasing order, and are always larger than all elements in H_i. Furthermore, it follows that all elements in H_{i+1} are always larger than all elements in H_i.*

2. **Error guarantee for all queries:** Instead of only those queries x with $\text{rank}_{\pi_{H_i}}(x) \le R_i$, we are going to define a notion of error for *arbitrary query*. We define the estimate of $\text{rank}_{\pi_{H_i}}(x)$ by H_i as

$$\widehat{\text{rank}_{\pi_{H_i}}}(x) \coloneqq \text{RANK}_{H_i}(x) + \text{rank}_{\pi'_{H_i}}(x).$$

[15] For the buffer B, $\text{REMOVE}_{\max}(B)$ is defined naturally as removing the largest element in the buffer and output it.

Here π'_{H_i} is the ordered output we defined above. The error is therefore defined as

$$\Delta_{H_i}(x) \coloneqq \widehat{\text{rank}_{\pi_{H_i}}}(x) - \text{rank}_{\pi_{H_i}}(x).$$

3. **Resets:** Just as we defined a reset operation for elastic compactors, we also define resets for our sub-sketch H_i.

   - RESET(H_i): This operation resets the sum in Equation (4.8) at the cost of introducing more error. Concretely, this operation simply calls RESET($C_{i,j}$) for every compactor $C_{i,j}$ ($0 \le j < \log(1/\epsilon)$) in the sampler/compactor hierarchy defined in Section 4.2.

   Fixing a query $x \in \mathcal{U}$, we say a reset is *important* if at the time of reset, there exists at least one element in H_i that is smaller than x.

To summarize, we have the following lemma.

LEMMA 4.8. *Consider an arbitrary query x. Suppose there are $t_i(x)$ important resets. Given the same condition as Lemma 4.5, we have*

$$\mathsf{E}\left[\Delta_{H_i}(x)^2\right]^{1/2} = O(\epsilon) \cdot \sqrt{R_i \cdot \text{rank}_{\pi_{H_i}}(x) + t_i(x) \cdot R_i^2}.$$

Notably, this lemma plays an important role in the proof of Lemma 4.11; we defer the proof to Section 6.1.

4.3 All Quantiles Relative Error Sketch Now, we are ready to describe the complete algorithm for the relative error quantile estimation problem in the streaming model.

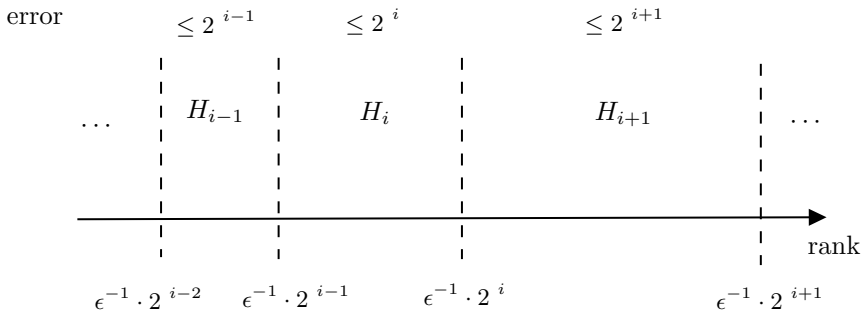

Figure 2: Our basic strategy. (repeated from page 9)

The Full Relative Error Quantile Sketch. As explained at the beginning of the section, for each $0 \le i \le \log_2(\epsilon n)$, our algorithm will maintain one sub-sketch H_i for elements of rank roughly $R_i \coloneqq \epsilon^{-1} \cdot 2^i$.[17] We now describe the operation of the algorithm on each inserted stream element x.

1. First, we insert x into the hierarchy H_i whose range contains x. More precisely, we feed x to the sampler of the first H_i such that H_{i+1} is either empty, or contains only elements that are greater than x.

2. As we mentioned earlier, H_i might have total weight exceeding $3R_i$ after this insertion; in this case, H_i will move at least R_i elements into the output steam π'_i; these elements will eventually be added to the input stream of the next hierarchy H_{i+1}.

3. Importantly, whenever Item 2 happens, we first *reset* the sketch H_{i+1} (Line 3, Algorithm 4) and then insert those newly added elements in π'_i into H_{i+1}.

[17] For the first $\log(1/\epsilon)$ many H_i's, we have $R_i \le \frac{1}{\epsilon^2}$, thus they have the structure described in Remark 4.4.

Our estimator is a natural extension of Equation (4.7): to estimate the rank of any query x, we output

$$\widehat{\text{rank}_\pi}(x) := \sum_{i=0}^{\log_2(\epsilon n)} \textsc{Rank}_{H_i}(x) \tag{4.9}$$

Before we proceed, we make a few observations about the structure of our overall sketch.

LEMMA 4.9. *Our all quantiles relative error sketch maintains the following properties:*

1. *Throughout the execution of the algorithm, all sub-sketches H_i maintain the invariant that the total weight of elements stored in H_i is within $[R_i, 3R_i]$ (excluding the last non-empty sub-sketch $H_{i'}$ which may have total weight less than $R_{i'}$).*
2. *Moreover, the ranges of the H_i's are disjoint, that is, all elements in H_{i-1} are always smaller than those in H_i.*

Proof. We note that the first property follows directly from the definition of H_i. So, we proceed to check the second property, which we prove by induction. Suppose that before an operation, the H_i's ranges are disjoint. Consider the following two cases:

- If this operation is an insertion (Item 1), because we insert x into the H_i whose range contains x. The disjointness is preserved.
- If this operation is moving elements from H_i to H_{i+1} (Item 2 and Item 3), by Observation 4.7, these elements we moved to H_{i+1} are larger than all elements in H_i. Thus the range of H_i and H_{i+1} are still disjoint.

□

At this point, it suffices for us to show that this sketch achieves the relative error guarantee, assuming that we can allocate an appropriate amount of space to each sub-sketch H_i (we will address our space allocation strategy next, in Section 4.4).

CLAIM 4.10. *For any sub-sketch H_i, the number of important reset $t_i(x) \le 2 \cdot \frac{\text{rank}_{\pi_{H_i}}(x)}{R_i}$.*

Proof. This is because whenever we have a important reset for H_i, there must be at least $R_{i-1} = R_i/2$ many elements that are smaller than x being inserted into H_i. □

LEMMA 4.11. *For any input stream π, assuming that the space we allocate to each H_i satisfies the premise of Lemma 4.5 (Equation (4.8)), we always have that for any query x,*

$$\mathsf{E}\left[\left|\text{rank}_\pi(x) - \sum_{i=0}^{\log_2(\epsilon \cdot \text{rank}_\pi(x))} \textsc{Rank}_{H_i}(x)\right|^2\right]^{1/2} \le O(\epsilon) \cdot \text{rank}_\pi(x).$$

where the expectation is taken over the randomness of the H_i's.

REMARK 4.12. *Before we prove this lemma, we remark that it suffices to prove that our algorithm succeeds with constant probability. This is because by Chebyshev's inequality, there is an absolute constant $c > 0$ such that*

$$\Pr\left[\left|\text{rank}_\pi(x) - \sum_{i=0}^{\log_2(\epsilon \cdot \text{rank}_\pi(x))} \textsc{Rank}_{H_i}(x)\right| \ge c \cdot \epsilon \cdot \text{rank}_\pi(x)\right] \le \frac{1}{3}.$$

But we know the total weight of $H_1, \ldots, H_{\log_2(\epsilon \cdot \text{rank}_\pi(x))}$ is at least $R_1 + R_2 + \cdots + R_{\epsilon \cdot \log_2(\epsilon \cdot \text{rank}_\pi(x))} > 1.5 \cdot \text{rank}_\pi(x)$. As long as, we pick $\epsilon < 1/(2c)$, there must be elements in these hierarchies that are larger than x. In this case, $\textsc{Rank}_{H_i}(x) = 0$ for all $i > \log_2(\epsilon \cdot \text{rank}_\pi(x))$.

Proof. By Claim 4.10, for any sub-sketch H_i, the number of important resets $t_i(x) = O\left(\frac{\text{rank}_{\pi_{H_i}}(x)}{R_i}\right)$. Together with $\mathsf{E}[\text{rank}_{\pi_{H_i}}(x)] \le \text{rank}_\pi(x)$, plug in Lemma 4.8, we get

$$\mathsf{E}\left[\Delta_{H_i}(x)^2\right]^{1/2} \le O(\epsilon)\cdot\sqrt{R_i\cdot\text{rank}_\pi(x) + \mathsf{E}[t_i(x)]\cdot R_i^2} = O(\epsilon)\cdot\sqrt{R_i\cdot\text{rank}_\pi(x)}.$$

Finally we use Fact 3.1 and sum over all the sub-sketches $H_1, H_2, \ldots, H_{\log_2(\epsilon\cdot\text{rank}_\pi(x))}$.

$$\begin{aligned}
&\mathsf{E}\left[\left|\text{rank}_\pi(x) - \sum_{i=0}^{\log_2(\epsilon\cdot\text{rank}_\pi(x))} \textsc{Rank}_{H_i}(x)\right|^2\right]^{1/2}\\
=&\mathsf{E}\left[\left|\sum_{i=0}^{\log_2(\epsilon\cdot\text{rank}_\pi(x))} \text{rank}_{\pi_{H_i}}(x) - \text{rank}_{\pi_i'}(x) - \textsc{Rank}_{H_i}(x)\right|^2\right]^{1/2}\\
\le&\sum_{i=0}^{\log_2(\epsilon\cdot\text{rank}_\pi(x))} \mathsf{E}\left[\Delta_{H_i}(x)^2\right]^{1/2}\\
\le&\sum_{i=0}^{\log_2(\epsilon\cdot\text{rank}_\pi(x))} O(\epsilon)\cdot\sqrt{R_i\cdot\text{rank}_\pi(x)}\\
=&O(\epsilon)\cdot\text{rank}_\pi(x).
\end{aligned}$$

Here π'_{H_i} is the stream of elements we move from H_i to H_{i+1}, and the last step follows since the values R_i's are exponentially increasing. ∎

4.4 Dynamic Space Allocation The only piece left from Lemma 4.11 is to specify the allocation of space for each H_i. Naively, the most straightforward approach is to give each H_i the same amount of space. However, since each H_i solves a top quantiles problem, we need at least $O\left(\frac{\sqrt{\log(\epsilon n)}}{\epsilon}\right)$ space for each H_i and can at best get an algorithm with $O\left(\frac{\log^{1.5}(\epsilon n)}{\epsilon}\right)$ space. To go beyond this lower bound, we will need to *dynamically* adjust the space used by each subsketch H_i *on-the-fly*. Before we give a complete description of the online space allocation procedure, we describe a way to allocate space in the restricted (offline) case, when all reset times are known in advance to the algorithm.

The offline space allocation problem. Our strategy for allocating space affects the behavior of each H_i and also changes the time when different levels reset. To best illustrate our idea, let us first consider the *offline* version, where the compactor reset times are fixed and known in advance.

First, we will view each operation call that changes the memory of the H_i's (i.e. for instance, an insertion of a sampled element into the zero-th level compactor $C_{i,0}$ or a resize call to H_i) as one time step[18]. Notably, it is clear from the algorithm description that there are $O(\mathsf{poly}(n))$ time steps in total, where n is the stream length. Also, let $t_{i,j}$ denote the time step at which H_i resets for the j-th time. We use $W_{i,j}$ to denote the set of time points within the interval $[t_{i,j}, t_{i,j+1})$ at which we resize H_i/insert from the sampler into $C_{i,0}$, and $s_{i,t}$ is the space we allocate to H_i at time t (which must be positive).

Our strategy satisfies the following two constraints:

- (The total space is bounded.) At any time t,

$$\sum_{i=0}^{\log(\epsilon n)} s_{i,t} = O(\epsilon^{-1}\log n\log(1/\epsilon)).$$

[18]Importantly, observe that this definition does not count those insertions that are not sampled in the "sampler" step, as these do not affect the state of the H_i's.

- (The space sequence is feasible.) For any H_i and time interval $[t_{i,j}, t_{i,j+1})$, we have

$$\sum_{t \in W_{i,j}} 2^{-\epsilon \cdot s_{i,t}} \leq 0.5.$$

Concretely, the offline space allocation question is defined as follows: given fixed time intervals $[t_{i,j}, t_{i,j+1})$ and the time of operations $W_{i,j}$'s, we need to find the proper sequence of space allocations $s_{i,t}$'s satisfying both of the constraints above.

REMARK 4.13. *To handle the edge case where some of the H_i's are empty, we reset those sub-sketches H_i at every time-step until they become non-empty. Note that this does not change the state of those empty H_i's at all, but ensures that when H_i is empty, the corresponding time intervals between consecutive resets are all of length 1. We will need this fact later when handling unknown stream length n.*

Space allocation for tree-like intervals. As a warm up, we will first describe the space allocation strategy for the following (simpler) special case: suppose that for all the intervals $[t_{i+1,j}, t_{i+1,j+1})$, there always exists an interval $\left[t_{i,\text{parent}(j)}, t_{i,\text{parent}(j)+1}\right)$ that contains it. In this case, these intervals form a tree-like structure, which is shown in Figure 4.

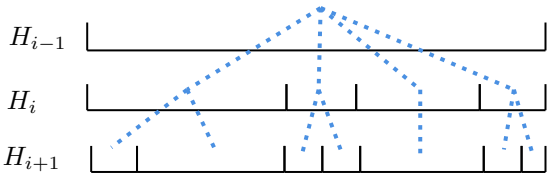

Figure 4: Tree-like intervals.

Our strategy for this case is easy to state: For all $t \in [t_{i+1,j}, t_{i+1,j+1})$, we set the space of H_i (the parent) as

$$s_{i,t} = \epsilon^{-1} \cdot \left(\log \frac{t_{i,\text{parent}(j)+1} - t_{i,\text{parent}(j)}}{t_{i+1,j+1} - t_{i+1,j}} + 5 \log(1/\epsilon) \right). \tag{4.10}$$

For the corner case when $i = \log(\epsilon n)$, we pretend that there is $H_{\log(\epsilon n)+1}$ which resets every step. That is, there is one addition level at the bottom with all length-1 intervals. We know that $s_{i,t}$ is always positive because $[t_{i+1,j}, t_{i+1,j+1})$ is always shorter or equal to its parent.

We need to verify that it satisfies the two constraints:

- (The total space is bounded.) This part is straightforward to verify. We first fix a time t. Let j_i be the index of the interval $[t_{i,j_i}, t_{i,j_i+1})$ that contains time t. Then, we obtain the following telescoping sum.

$$\begin{aligned}
\sum_{i=0}^{\log(\epsilon n)} s_{i,t} &= \sum_{i=0}^{\log(\epsilon n)} \epsilon^{-1} \cdot \left(\log \frac{t_{i,j_i+1} - t_{i,j_i}}{t_{i+1,j_{i+1}+1} - t_{i+1,j_{i+1}}} + 5 \log(1/\epsilon) \right) \\
&= \epsilon^{-1} \cdot \log \frac{\mathsf{poly}(n)}{1} + \epsilon^{-1} \cdot \log(\epsilon n) \cdot 5 \log(1/\epsilon) \\
&= O(\epsilon^{-1} \log n \log(1/\epsilon)).
\end{aligned}$$

  Thus the total space is at most $O(\epsilon^{-1} \log n \log^2(1/\epsilon))$ because the space used by each hierarchy H_i is $s_{i,t} \cdot \lceil \log_2(1/\epsilon) \rceil$.

- (The space sequence is feasible.) To prove this, we need the following claim.

  CLAIM 4.14. *Within any interval $[t_{i+1,j}, t_{i+1,j+1})$, H_i has at most $3/\epsilon^2$ many insertions into $C_{i,0}$. We observe that this is true in general, i.e. not only for the case that all resets occur in tree-like intervals.*

Proof. This is because each element in $C_{i,0}$ has weight $\epsilon^2 R_i$. If there are more than $3/\epsilon^2$ such insertions, it will triger a reset of H_{i+1}. As $t_{i+1,j+1}$ is the next reset after $t_{i+1,j}$, we know there can be at most $3/\epsilon^2$ insertions between them. ∎

Consider any interval $[t_{i,j}, t_{i,j+1})$. Suppose it is the union of its children $[t_{i+1,\ell}, t_{i+1,\ell+1})$, $[t_{i+1,\ell+1}, t_{i+1,\ell+2})$, $\ldots$, $[t_{i+1,r-1}, t_{i+1,r})$. We know that H_i only resizes at time step $t_{i+1,\ell}$, $t_{i+1,\ell+1}$, $\ldots$, $t_{i+1,r}$. Thus, together with Claim 4.14, we know that for each $[t_{i+1,m}, t_{i+1,m+1})$ $(l \le m \le r-1)$, we have $|W_{i,j} \cap [t_{i+1,m}, t_{i+1,m+1})| \le 3/\epsilon^2 + 2$. Thus, we know

$$\begin{aligned}\sum_{t\in W_{i,j}} 2^{-\epsilon\cdot s_{i,t}} &\le \sum_{m=\ell}^{r-1} \sum_{t\in W_{i,j}\cap[t_{i+1,m},t_{i+1,m+1})} 2^{-\epsilon\cdot s_{i,t}} \\ &\le \sum_{m=\ell}^{r-1} (3/\epsilon^2+2)\cdot \frac{t_{i+1,m+1}-t_{i+1,m}}{t_{i,j+1}-t_{i,j}}\cdot\epsilon^5 \le 0.25\cdot \frac{\sum_{m=\ell}^{r-1} t_{i+1,m+1}-t_{i+1,m}}{t_{i,j+1}-t_{i,j}} \le 0.25\end{aligned}$$

Space allocation for general intervals. In the general case, a single interval for H_{i+1} may intersect with multiple "parent" intervals for H_i above it. It might also be longer than any such "parent" interval, which could cause Equation (4.10) to allocate negative space. Thus, we need to carefully generalize our strategy.

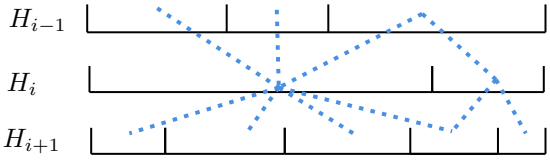

Figure 5: In general, the incidence relation between intervals can be far more complex than a tree.

First, we need the following claim, which upper bounds the number of "parent intervals" for a single interval by a constant.

CLAIM 4.15. *Within the time interval* $\left[t_j^{(i)}, t_{j+4}^{(i)}\right)$, H_{i+1} *resets at least once.*

Proof. Each time H_i resets, the total weight of H_i increases by at least $R_{i-1} = \epsilon^{-1}\cdot 2^{i-1}$ due to the corresponding batched insertion. Initially, the total weight of H_i is at least $2^i/\epsilon$. Within at most four such resets, its weight increases to at least $3\cdot 2^i/\epsilon$, which triggers a reset of H_{i+1}. ∎

To handle the corner case, again we imagine there is a $H_{\log(\epsilon n)+1}$ which resets every step. So there is a imaginary level at the bottom full of length-1 intervals. We define the potential $\phi_{i,j}$ for each interval $[t_{i,j}, t_{i,j+1})$ recursively as follows:

- For $i = \log(\epsilon n)+1$ (the imaginary level at the bottom), we simply let $\phi_{i,j} = 1$ for all j.
- For $0 \le i \le \log(\epsilon n)$, suppose $[t_{i,j}, t_{i,j+1})$ intersects with "children" intervals $[t_{i+1,\ell}, t_{i+1,\ell+1}]$, $[t_{i+1,\ell+1}, t_{i+1,\ell+2}]$, $\ldots$, $[t_{i+1,r-1}, t_{i+1,r}]$. We define

$$\phi_{i,j} = \sum_{m=\ell}^{r-1} \phi_{i+1,m}.$$

Intuitively speaking, the potential $\phi_{i,j}$ is a the "generalized length" of each interval. We then assign the space similar to the tree-like case. For all $t \in [t_{i,j}, t_{i,j+1}) \cap [t_{i+1,m}, t_{i+1,m+1})$, set

$$s_{i,t} = \epsilon^{-1}\cdot\left(\log\frac{\phi_{i,j}}{\phi_{i+1,m}} + 5\log(1/\epsilon)\right). \tag{4.11}$$

In this case, we know that $s_{i,t}$ is always positive because $\phi_{i,j}$ is always larger than $\phi_{i+1,m}$ by definition. We defer the analysis to Section 6. In the analysis, we crucially use the fact that all $\phi_{i,j}$'s are at most $\mathsf{poly}(n)$ so that the telescoping sum for calculating total space will sum up to $O(\log n)$. The rest of the analysis are almost identical to the tree-like case.

CLAIM 4.16. *For any $0 \le i \le \log(\epsilon n) + 1$, suppose there are ℓ_i intervals in total for H_i. Suppose that there are T time steps (counting insertion into $C_{i,0}$'s and resizes of H_i's) in total.*

Then,

$$\sum_{j=1}^{\ell_i} \phi_{i,j} \le 4^{\log(\epsilon n)+1-i} \cdot \ell_{\log(\epsilon n)+1}.$$

Specifically, the potential $\phi_{i,j}$ of any interval is bounded by $4^{\log(\epsilon n)+1} \cdot T = \mathsf{poly}(\epsilon n) \cdot T$.

Proof. We prove this by induction. When $i = \log(\epsilon n) + 1$, we know that every $\phi_{i,j} = 1$. Thus $\sum_{j=1}^{\ell_i} \phi_{i,j} = \ell_{\log(\epsilon n)+1}$, which equals T.

Suppose this is true for $i+1$. From Claim 4.15, we know that each interval $[t_{i+1,j'}, t_{i+1,j'+1})$ intersects with at most 4 "parent" intervals $[t_{i,j}, t_{i,j+1}]$'s.

Thus

$$\sum_{j=1}^{\ell_i} \phi_{i,j} \le 4 \cdot \sum_{j=1}^{\ell_{i+1}} \phi_{i,j+1} \le 4 \cdot 4^{\log(\epsilon n)-i} \cdot \ell_{\log(\epsilon n)+1},$$

where in the last step we use the induction hypothesis. ∎

Online space allocation. In the actual streaming model, we do not know the intervals in advance, so we need to adjust our strategy in order to allocate the space of each H_i on-the-fly.

Let t be the current time. For all unfinished intervals, we pretend that the interval ends (i.e. the corresponding sub-sketch H_i resets) at the *current* time t; then, we calculate all the potentials which we denote by $\phi_{i,j}^{(t)}$. We then round them up to the closest power of 2, which we denote by $\lceil\!\lceil \phi_{i,j}^{(t)} \rceil\!\rceil$. We also calculate $\widehat{s}_{i,t} = \epsilon^{-1} \cdot \left(\log \frac{\lceil\!\lceil \phi_{i,j}^{(t)} \rceil\!\rceil}{\lceil\!\lceil \phi_{i+1,m}^{(t)} \rceil\!\rceil} + 5\log(1/\epsilon) + 5\log\log n \right)$. We then allocate to H_i space $\widehat{s}_{i,t}$.

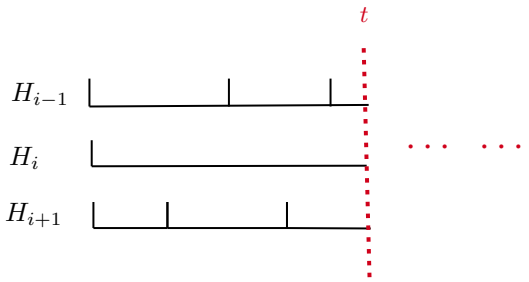

Figure 6: We calculate $\phi_{i,j}^{(t)}$'s using partial information.

Intuitively, this works because the rounding guaranteed that the space $\widehat{s}_{i,t}$ does not change too often. Roughly speaking the number of resize operations multiplies by $O(\log^2 n)$, which offsets by the extra $\log\log n$ term in $\widehat{s}_{i,t}$. We again leave the detailed analysis to Section 6.

Handling unknown stream length. One side benefit of our algorithm is that it handle unknown n naturally. In our algorithm, none of the parameters (k, R_i and buffer size) for each H_i depends on n. Thus, we can imagine running our algorithm with infinite many subsketches, $H_0, H_1, H_2, \ldots$.

The only potential issue is that, our recursive definition of the potential $\phi_{i,j}^{(t)}$'s needs a base case. Recall that by Remark 4.13, all empty H_i's have intervals of length 1. We can the potential of those intervals to 1. Because whenever H_i is empty, H_{i+1} must also be empty, any such length-1 interval of H_i can only intersect with a single length-1 interval of H_{i+1}. So the recurrence relation still holds for all intervals.

5 Implementation of the Elastic Compactor

In this section, we fill in the remaining implementation details for the elastic compactors $C_{i,j}$'s used in our subsketches. Before we proceed with this, we first provide a necessary overview of *relative compactor*, which was introduced in [3]; importantly, some basic operations of our elastic compactor data structure will be partially adapted from the relative compactor.[19]

5.1 Overview of the Relative Compactor

Blocks and Compaction. A relative compactor [3] stores s elements (indexed 1 through s), which are subdivided into $b = \lceil \frac{s}{k} \rceil$ blocks of size k. [20] Let $\{B_i\}_{i\in[b]}$ denote the blocks in C, and note that each block B_i contains elements indexed from $(i-1)\cdot k+1$ to $i \cdot k$. The elements stored in C are always maintained in sorted increasing order.

As new elements arrive in the stream, they are inserted directly into the relative compactor C while maintaining the increasing sorted order in C. When C is full, the suffix of ℓ blocks is fed as input to the *compaction* operation, which is defined below. This operation compacts the $(b-\ell+1)\cdot k$ elements in the suffix of blocks into $(b-\ell+1)\cdot k/2$ outputted elements. In particular, the compaction always frees-up space for future insertions.

Algorithm 5 Compacting suffix $[B_\ell, B_{\ell+1}, \ldots, B_b]$

```
1: procedure COMPACT(C, ℓ)
2:     Let y ∈_R {0,1} be a fair coin flip.
3:     if y = 0 then
4:         Output all odd indexed elements in blocks B_ℓ, B_{ℓ+1}, ..., B_b.
5:     else
6:         Output all even indexed elements in blocks B_ℓ, B_{ℓ+1}, ..., B_b.
7:     Remove all elements in suffix [B_ℓ, B_{ℓ+1}, ..., B_b] from C.
```

Compaction Schedule. However, we have not specified how to pick ℓ when calling $\textsc{Compact}(C,\ell)$. Indeed, the work of [3] presents an intricate method to select the number of blocks ℓ which will be compacted in a particular iteration; the sequence of choices of ℓ is called the *compaction schedule*. This schedule is essential for the error analysis in [3]. In the context of our work, we rephrase their strategy as follows.

Initially, each block B_i is associated with a bit z_i that is initialized to 0. The compactor C has a "progress measure", $z := (\overline{.z_1 z_2 \ldots z_b})_2$, where $0 \le z < 1$ is a binary fraction. To ensure that the total accumulated error stays small, the compactor C only has the capacity to handle a limited number of compactions. Intuitively, the bit-string z indicates how much of the "capacity" of C we have used already, and represents the "progress" of the compaction schedule. Eventually, if $z = 1 - 2^{-b}$, C cannot perform any more compactions. The scheduling strategy of [3] is to do the following for each compaction[21]:

1. Update $z \leftarrow z + 2^{-(b-1)}$.
2. Find the least significant bit z_r such that $z_r = 1$.
3. Call $\textsc{Compact}(C, r+1)$.

Note that in the furst line, we always add $2^{-(b-1)}$ instead of 2^{-b} to z, as a result, the last bit z_b is always zero. So $r \le b-1$, and in the last line we always compact at least one block.

[19] Although we presented a sketch of relative compactor in Section 2, the presentation here will be based on the deterministic compaction schedule of [3] (as we mentioned in Footnote 8), which is easier to formally analyze and helps up the basic notations for our elastic compactor.

[20] k is a even integer parameter which relates to the error incurred by compactor C.

[21] In [3], the authors maintain a counter Z for the number of compactions that have already been performed. Then, the number of blocks to compact is determined by the *number of trailing ones in the binary representation of Z + 1*. We note that these two definitions are equivalent. Numerically, the sequence of these numbers are $\{1,2,1,3,1,2,1,4,...\}$.

For any query $x \in \mathcal{U}$, the compactor should provide an estimate of the rank of x *with respect to the current contents of* C (each element has weight 1) *and with respect to the output stream* π' *of* C (each element in π' has weight 2). Specifically, this estimator is defined as

$$\widehat{\text{rank}}_\pi(x) = \text{rank}_C(x) + 2 \cdot \text{rank}_{\pi'}(x).$$

Recall that to achieve the relative error guarantee, we need to ensure that our estimate $\widehat{\text{rank}}_\pi(x)$ is very accurate when $\text{rank}_\pi(x)$ is small (in fact, the first $1/\epsilon$ ranks should be known exactly). The compaction schedule chosen in [3] facilitates this guarantee by ensuring that the "smaller" elements of the stream will get compacted much less frequently than the "larger" elements, so they will naturally incur a smaller amount of error over all time-steps of the algorithm.

Error Analysis Now, we examine the source of error for rank queries in the compaction procedure. Note that $\textsc{Compact}(C, \ell)$ introduces no error for x if all elements in block $B_\ell, B_{\ell+1}, \ldots, B_b$ are larger than x. So a compaction may introduce error, only when the involved blocks contains *at least one "important" element* (i.e. an element that is smaller than the query x). This notion is defined more formally in the next definition.

Definition 5.1. (Important element for query x) *For a fixed query $x \in \mathcal{U}$, we say a stream element y is* important *if and only if $y \le x$. We also say a compaction $\textsc{Compact}(C, \ell)$ is* important *if the $B_\ell, B_{\ell+1}, \ldots, B_b$ contains at least one important element.*

Since each important compaction induces an independent ± 1 error to the rank estimate, if there are N important compactions in total, the standard deviation of our estimation will be bounded by $\pm\sqrt{N}$. Thus, it suffices for us to upper bound the total number of important compactions.

In the previous work [3], the authors showed that the compaction schedule described above, the number of important compactions is upper-bounded by $\frac{\text{rank}_\pi(x)}{k}$, and C is able to handle roughly $k2^b$ insertions before the "progress measure" z overflows. By chaining $O(\log(\epsilon n))$ such compactors together, they construct a relative-error quantile sketch using space $\tilde{O}(\epsilon^{-1}\log^{1.5}(\epsilon n))$.

5.2 Implementation of Elastic Compactors Now, we are ready to describe the implementation of our *elastic compactor*. Notably, the basic compaction operation and the compaction schedule will be generalized from those of relative compactors. Additionally, our elastic compactor is augmented with a *resize* operation, which can be used to adjust the memory allocated to C, as well as a *reset* operation which simply zero-outs the "progress measure" z at the cost of introducing more error.

As before, we consider C_{elastic} to have blocks of size k, but unlike the original relative compactor, we will allocate the *number of blocks* in C_{elastic} dynamically; as a result, we can think of C_{elastic} as having possibly infinitely-many blocks $\{B_i\}_{i\ge 1}$ available, but only a specific number of blocks will actually be in-use at each time-step. As for the relative compactor, each block B_i will be equipped with a bit z_i, which is initialized to 0. Analogously to the progress measure for the relative compactor, the compaction procedure keeps track of its progress measure $z := (\overline{.z_1 z_2 \ldots})_2$ (observe that the number of bits is no longer fixed in advance).

Basic Operations. Recall that we have the following basic operations of the elastic compactor $\mathcal{C}_{\text{elastic}}$. (See Table 3 on the next page.) In the description below, we let s_t denote the total amount of space allocated to C_{elastic} at time t (i.e. if the space allocated is s_t, then there are $\lceil \frac{s_t}{k} \rceil$ blocks allocated to C_{elastic} at time t).

Recall that in Lemma 4.1, we require that C_{elastic} can handle any sequence of insert and resize calls with a sequence of space constraints $s_1, s_2 \ldots, s_m$ (after a reset) which satisfy $\sum_{t=1}^{m} 2^{-s_t/k} \le 0.5$. Since we implement insertions using resizes, each "external" insert call with parameter s generates two "internal" resize calls with parameters s and $2s$ (See Table 3). Let sequence $s'_1, s'_2, \ldots, s'_{m'}$ be the space parameters for all (internal and external) resize calls after a reset. As $2^s + 2^{-2s} \le 2 \cdot 2^{-s}$, we know that $\sum_{t=1}^{m'} 2^{-s'_t/k} \le 1$.

Now, let us relate the running sum $\sum_{t=1}^{m'} 2^{-s'_t/k}$ to our "progress measure" z.

Lemma 5.2. *Let sequence $s'_1, s'_2, \ldots, s'_{m'}$ be the space parameters for all (internal and external) resize calls after a reset. At any time in the operation sequence, we always have $z \le \sum_{t=1}^{m'} 2^{-s'_t/k}$. That is, the "progress measure" z is always upper-bounded by the running sum.*

Proof. We prove this by induction. Initially, after a reset, both z and the running sum are zero.

Basic operations for Elastic Compactor C

- COMPACT(C, ℓ): This operation is the same as Algorithm 5, while the number of blocks is $b = \lceil \frac{s_t}{k} \rceil$ for current time t.
- RESIZE(C, s'): Expand or compress the space allocated to C. Suppose that C had space s previously. Then:
  1. If $s' > s$, this corresponds to *expanding* the space for C. Allocate $\lceil \frac{s'}{k} \rceil - \lceil \frac{s}{k} \rceil$ additional blocks at the end of C. For each of these block B_i, we initialize z_i to 0.
  2. If $s' < s$, this corresponds to *compressing* the space for C. Let $\ell = \lceil \frac{s'}{k} \rceil$ be the target number of blocks after the resize. We do the following:
     - Increase z to the next multiple of $2^{-\ell}$.
     - Find the least significant bit z_r such that $z_r = 1$.
     - Call COMPACT$(C, r+1)$.
     - Release all blocks $B_{>\ell}$.

     See Figure 7 for an example.
- INSERT$(C, x_1, ..., x_m)$: Suppose C has space s. As described in Section 4.1, we implement insertion using RESIZE$(C, \cdot)$ (implemented below).
  - First, we call RESIZE$(C, 2s)$, and insert $x_1, x_2, \ldots, x_m$ into the new open spaces in memory (note that $m \leq s$ necessarily).
  - Then, we call RESIZE(C, s) to compress the space back to the original size s.

  Note that some elements may be outputted to the output stream π' during this last compression.
- RESET(C): Set $z \leftarrow 0$.

Table 3: Implementations of basic operations to Elastic Compactors.

Suppose that this is true for $t-1$. When there is a resize call RESIZE(C, s'_t), the running sum is increased by $2^{-s'_t/k}$. We will have $\ell = \lceil \frac{s'_t}{k} \rceil$ and z is increase to the next multiple of $2^{-\ell}$. Because that multiple is at most $2^{-\ell}$ away and $2^{-\ell} \leq 2^{-s'_t/k}$, we know that this lemma is true for time t. □

Having this lemma, we know that as long as $\sum_{t=1}^{m} 2^{-s_t} \leq 0.5$ (Equation (4.8)) holds, we always have $0 \leq z < 1$. Then resizes can always be performed because the least significant bit z_r such that $z_r = 1$ always exists. Moreover, we always have $r \leq \ell$ because z is a multiple of $2^{-\ell}$. Then after COMPACT$(C, r+1)$, when we release the blocks $B_{>\ell}$, these blocks are guaranteed to be empty. Finally, INSERT can also always be performed (again conditioning on Equation (4.8) holds) because it is implemented by calls to RESIZE.

5.3 Proof of Lemma 4.1 and Lemma 4.2 In this subsection, we will prove Lemma 4.2. If we set $t = 0$ (having no reset at all) in Lemma 4.2, it directly implies Lemma 4.1.

Let $x \in \mathcal{U}$ be a fixed query. We first recall the following definition of important elements, and provide a definition for an "important reset":

DEFINITION 5.3. *Recall the following definition from Definition 5.1: for a fixed query $x \in \mathcal{U}$, we say a stream element y is important if $y \leq x$. Also, we say a reset operation is important if, at the time of reset, at least one of the elements in the compactor is important.*

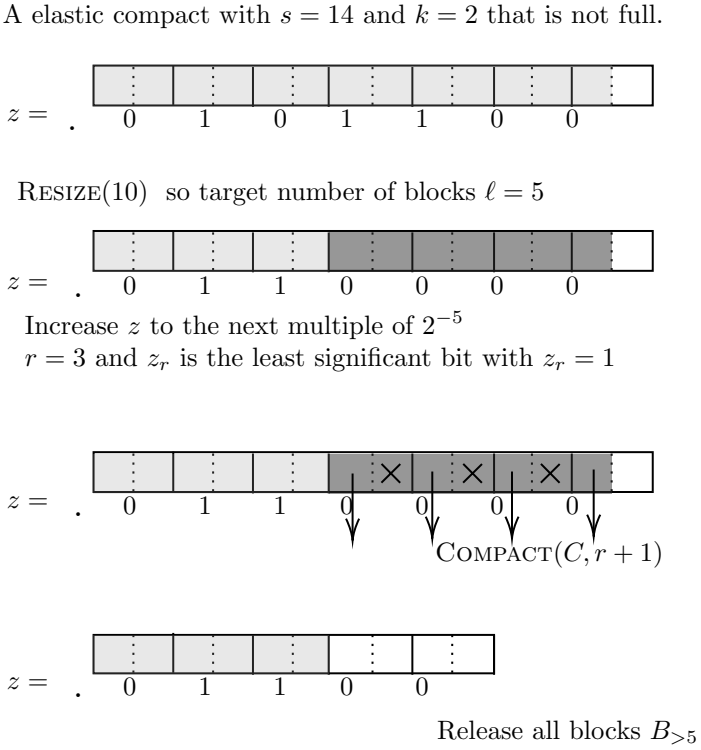

Figure 7: An example of the resize opeation.

Lemma 4.2 states the following:

LEMMA 4.2. *Suppose between any two adjacent resets, the sequence of space constraints in the resize operations satisfies Equation* (4.5)*. Let S be an upper bound on the number of elements in the compactor at any time. Then for any query $x \in \mathcal{U}$, if there are t important resets, the randomized compactor achieves expected squared error*

$$\mathsf{E}\left[\Delta(x)^2\right] \leq \frac{\mathrm{rank}_\pi(x) + t \cdot S}{k}.$$

Here $\Delta(x)$ is defined as in Equation (4.6)*.*

Recall that the estimation error solely stems from *important compactions* for query x (See Definition 5.1). Let N be the number of important compactions. Then, the error distribution $\Delta(x)$ is the sum of (at most) N independent ± 1 random variables. As a result, we have $\mathsf{E}\left[\Delta(x)^2\right] \leq N$. Therefore, it suffices show that $N \leq \frac{\mathrm{rank}_\pi(x)+t\cdot S}{k}$. Going forward, we will have a charging argument similar to that of [3]:

Initially all the blocks of the compactor are unmarked.

- For each important compaction, we will "mark" an unmarked block.
- A marked block will only be unmarked at a reset, or when k important elements are compacted from it. This allows us to upper bound the total number of marks by $\frac{\mathrm{rank}_\pi(x)+t\cdot S}{k}$.

Combining the two bullets above, we obtain the desired upper bound for N and conclude the proof of Lemma 4.2.

Mark the elements. Consider a call COMPACT$(C, r+1)$, for it to be important, there must be at least one important element (an element y such that $y \leq x$) in blocks $B_{\geq r+1}$. We mark the block B_r if COMPACT$(C, r+1)$ is important. We unmark block B_i when one of the following happens:

1. There is a compaction call COMPACT(C, ℓ) involving B_i (i.e., $\ell \leq i$).

2. There is a reset.

Let us prove that in our resizes, whenever we call $\textsc{Compact}(C, r+1)$, the block B_r must be unmarked before.

Claim 5.4. *The block B_i is marked only when $z_i = 1$.*

Proof. Fisrt, whenver we call $\textsc{Compact}(C, r+1)$ and mark a block B_r, z_r is always the least significant bit with $z_r = 1$. So when B_i is marked, we must have $z_i = 1$.

Second, when we change z_i to 0 in a resize call, suppose that the target number of blocks of the reset is ℓ and we call $\textsc{Compact}(C, r+1)$. If B_i is not unmarked, we must have $i \le r$. But all the bits $z_1, z_2, \ldots, z_{r-1}$ are not changed during this resize, and z_r is changed to 1. Therefore, z_i must remains 1 if B_i is still marked. ∎

Corollary 5.5. *Whenever we call $\textsc{Compact}(C, r+1)$ (in the resize operation), the block B_r must be unmarked before.*

Proof. In a resize, we first increase z to a multiple of $2^{-\ell}$. Because z_r is the least significant bit equal to 1 after the increment, we know that before the increment, it must be the case that $z_r = 0$. Then the block B_r must be unmarked by Claim 5.4. ∎

Bound the number of marks To bound the number of marks, we first formalize the following claim, which suggests that we can bound the numebr of marks by the number of important elements.

Claim 5.6. *A marked block contains only important elements.*

Proof. Initially, suppose that a block B_r is marked because the compaction $\textsc{Compact}(C, r+1)$ is important. Because elements in C are in sorted increasing order, all blocks $B_{\le r}$ must also be smaller than or equal to x. Thus, B_r contains only important elements.

Afterward, if B_r is not unmarked, it cannot be involved in any compaction. The only way the elements in B_r can change is due to insertions. Because we always sort the elements in C after insertions, the elements in B_r can only monotonely become smaller. Thus B_r still contains only important elements after the insertions. ∎

Lemma 5.7. *Suppose there are t important resets, the total number of blocks we mark is bounded by $\frac{\mathrm{rank}_\pi(x)+t\cdot S}{k}$ where S is an upper bound on the number of elements in C.*

Proof. For every block that is marked, it is either unmarked due to (1) a compaction or (2) a reset; or it stays marked at the end of the algorithm.

If it is unmarked due to compaction (or stays marked at the end of the algorithm). By Claim 5.6, we know at the time of that compaction (or at the end of the algorithm) it must contain only important elements. These k important elements are removed from C during the compaction (or stays in C till the end). This bounds the number of such marked blocks by $\frac{\mathrm{rank}_\pi(x)}{k}$.

If it is unmarked due to a reset, we note that during a reset, there are at most $\frac{S}{k}$ many blocks in C. So it unmarks at most $\frac{S}{k}$ blocks. There are t important resets in total that umarks at most $\frac{t\cdot S}{k}$ blocks. For the unimportant resets, by Claim 5.6, they never unmark any block.

Adding these two together finishes the proof. ∎

Putting together Lemma 5.7 and Corollary 5.5, we conclude that the number of important compactions can be at most $\frac{\mathrm{rank}_\pi(x)+t\cdot S}{k}$. So the final error is a sum of this many ± 1 Rademacher random variables. We conclude that the variance $\mathsf{E}[\Delta(x)^2] \le \frac{\mathrm{rank}_\pi(x)+t\cdot S}{k}$.

6 Analysis of the Full Sketch

In this section, we fill in the gaps in our algorithm analysis: we first finish the upper bound on the expected square error in Section 6.1. Then, we show that our algorithm succeeds with high probability in Section 6.2. Furthermore, to improve the space complexity from $\tilde{O}(\epsilon^{-1}\log n)$ to $\tilde{O}(\epsilon^{-1}\log(\epsilon n))$, we make a small modification to our algorithm, which is presented in Section 6.3. Finally, we finish the analysis of our dynamic space allocation rules and prove the final space complexity bound in Section 6.4.

6.1 Expectation Bound In this subsection, we first provide the error analysis for a single compactor hierarchy H which solves the top quantiles problem (i.e. supporting queries of rank at most R). Later, we generalize our analysis to our overall algorithm for the relative-error quantile estimation problem, in which sub-sketches H_i also have "reset" operations and support arbitrary queries.

To address the first objective above, we recall Lemma 4.5, which gives an upper-bound on the expected squared error of the top-R quantiles sketch H defined in section Section 4.2.

LEMMA 4.5. *Let π be the input stream and $s_1, s_2, \ldots, s_\ell$ be the sequence of space parameters after each resize / insertion from the sampler into C_0. Our randomized top-R quantiles sketch H, conditioning on*

$$\sum_{t=1}^{\ell} 2^{-\epsilon \cdot s_t} \leq 0.25, \tag{4.8}$$

for any query x with $\mathrm{rank}_\pi(x) \leq R$, achieves standard deviation

$$\mathsf{E}\left[|\textsc{Rank}_H(x) - \mathrm{rank}_\pi(x)|^2\right]^{1/2} \leq O(\epsilon) \cdot R.$$

Proof. We first decompose the error. Again let $C_0, \ldots, C_{\log(1/\epsilon)-1}$ be the compactors in H, and $C_{\log(1/\epsilon)}$ denotes the buffer. We use π_j to denote the input stream of C_j (which is the output stream of C_{j-1}).

$$\begin{aligned}
\textsc{Rank}_H(x) - \mathrm{rank}_\pi(x) &= \sum_{j=0}^{\log(1/\epsilon)} \textsc{Rank}_{C_j}(x) \cdot 2^j \cdot \epsilon^2 \cdot R - \mathrm{rank}_\pi(x) \\
&= \sum_{j=0}^{\log(1/\epsilon)-1} \left((\textsc{Rank}_{C_j}(x) + 2\mathrm{rank}_{\pi_{j+1}}(x)) - \mathrm{rank}_{\pi_j}(x)\right) \cdot 2^j \cdot \epsilon^2 R \\
&\quad + \left(\mathrm{rank}_{\pi_0}(x) \cdot \epsilon^2 R - \mathrm{rank}_\pi(x)\right) \\
&\quad + \left(\textsc{Rank}_{C_{\log(1/\epsilon)}}(x) - \mathrm{rank}_{\pi_{\log(1/\epsilon)}}(x)\right) \cdot \epsilon R. \qquad (6.12)
\end{aligned}$$

The first term is the error of each compactor C_j. The second one is the error of the sampler. Third one is the buffer.

We first argue that the buffer (the third term) never contributes to the total error:

- If $\mathrm{rank}_{\pi_{\log(1/\epsilon)}}(x) \leq 1/\epsilon$, because the buffer always keep the smallest $1/\epsilon$ elements, we always have $\mathrm{rank}_{C_{\log(1/\epsilon)}}(x) = \mathrm{rank}_{\pi_{\log(1/\epsilon)}}(x)$. Thus the third term is zero.
- If $\mathrm{rank}_{\pi_{\log(1/\epsilon)}}(x) > 1/\epsilon$, intuitively, because $\mathrm{rank}_\pi(x) \leq R$, the compactors already over estimate the rank of x. By only keep the first $1/\epsilon$ elements, the buffer only reduces the error.

  Formally, the sum of the first two terms equals

$$\left(\sum_{j=0}^{\log(1/\epsilon)-1} \textsc{Rank}_{C_j}(x) \cdot 2^j \cdot \epsilon^2 R + \mathrm{rank}_{\pi_{\log(1/\epsilon)}}(x) \cdot \epsilon R\right) - \mathrm{rank}_\pi(x).$$

  This sum, because $\mathrm{rank}_\pi(x) \leq R$, is at least $\left(\mathrm{rank}_{\pi_{\log(1/\epsilon)}}(x) - 1/\epsilon\right) \cdot \epsilon R$. Thus as the buffer $C_{\log(1/\epsilon)}$ keeps the first $1/\epsilon$ elements and throw away the rest. It is negative and of a smaller magnitude than the first two terms. So it can only reduce the error.

Second, let us analyze the first term. We apply Lemma 4.1 to each compactor C_j. We need to verify that the condition (Equation (4.5)) holds. For $\textsc{Resize}(C_j, \cdot)$, we only call it when we resize H. For $\textsc{Insert}(C_j, \cdot)$, we call it once only when either there is a resize of H or an insertion into C_0. Thus we have

$$(4.5) \leq 2 \cdot \sum_{t=1}^{\ell} 2^{-s_t/k} = 2 \cdot \sum_{t=1}^{\ell} 2^{-\epsilon \cdot s_t} \leq 0.5.$$

Let π_j be the input stream of C_j and

$$\Delta_j(x) = (\textsc{Rank}_{C_j}(x) + 2\text{rank}_{\pi_{j+1}}(x)) - \text{rank}_{\pi_j}(x)$$

be the error of C_j as defined in Equation (4.4). As $\mathsf{E}[\text{rank}_{\pi_j}(x)] = \frac{\text{rank}_\pi(x)}{\epsilon^2 R \cdot 2^j} \le \frac{1}{\epsilon^2 \cdot 2^j}$ (because $\text{rank}_\pi(x) \le R$), we know (from Lemma 4.1) that,

$$\mathsf{E}\left[\Delta_j(x)^2\right]^{1/2} \le \mathsf{E}\left[\frac{\text{rank}_{\pi_j}(x)}{k}\right]^{1/2} \le \sqrt{\frac{1}{k\epsilon^2 \cdot 2^j}} = \sqrt{\frac{1}{\epsilon \cdot 2^j}}.$$

The contribution from the first term is therefore (by Fact 3.1),

$$\sum_{j=0}^{\log_2(1/\epsilon)-1} \epsilon^2 R \cdot 2^j \cdot \mathsf{E}\left[\Delta_j(x)^2\right]^{1/2} \le \sum_{j=0}^{\log_2(1/\epsilon)-1} \epsilon^2 R \cdot 2^j \cdot \sqrt{\frac{1}{\epsilon \cdot 2^j}} = O(\epsilon) \cdot R.$$

Thirdly, we will also verify the error of the sampler (the second term) is $O(\epsilon) \cdot R$. This is because we sample with probability $p = \frac{1}{\epsilon^2 R}$. We have that the rank in the sampled stream $\text{rank}_{\pi_0}(x) = \sum_{t=1}^{\text{rank}_\pi(x)} \text{Bernoulli}(p)$. Thus

$$\mathsf{E}\left[\left(\text{rank}_{\pi_0}(x) \cdot \epsilon^2 R - \text{rank}_\pi(x)\right)^2\right]^{1/2} \le \epsilon^2 R \cdot \sqrt{R \cdot p(1-p)} \le \epsilon \cdot R.$$

Finally, we use Fact 3.1 again to add these parts together. This concludes our proof. ∎

We can extend this proof to our subsketch H_i's which have error guarantee for arbitrary queries and can handle resets. Recall Lemma 4.8 which is for our subsketch H_i's.

LEMMA 4.8. *Consider an arbitrary query x. Suppose there are $t_i(x)$ important resets. Given the same condition as Lemma 4.5, we have*

$$\mathsf{E}\left[\Delta_{H_i}(x)^2\right]^{1/2} = O(\epsilon) \cdot \sqrt{R_i \cdot \text{rank}_{\pi_{H_i}}(x) + t_i(x) \cdot R_i^2}.$$

Proof. Recall that we use π_{H_i} to denote the input stream to H_i and π'_{H_i} to denote the elements that we move from H_i to H_{i+1}. Further more, we use $\pi_{i,j}$ to denote the input stream of compactor $C_{i,j}$ and $\pi'_{i,j}$ to denote the output stream. Specially, $\pi_{i,j}^{\text{remove}}$ denotes the stream of maximum elements that we removed from $C_{i,j}$. Note that each element in $\pi_{i,j}^{\text{remove}}$ is added $2^j \cdot \epsilon^2 R$ times to π'_{H_i} (Line 7, Algorithm 4).

We perform a similar decomposition of error that is slightly more complicated than (6.12) and get

$$\begin{aligned}
\Delta_{H_i}(x) =& \textsc{Rank}_{H_i}(x) + \text{rank}_{\pi'_{H_i}}(x) - \text{rank}_{\pi_{H_i}}(x) \\
=& \sum_{j=0}^{\log(1/\epsilon)} \left(\textsc{Rank}_{C_{i,j}}(x) + \text{rank}_{\pi_{i,j}^{\text{remove}}}(x)\right) \cdot 2^j \cdot \epsilon^2 R_i - \text{rank}_{\pi_{H_i}}(x) \\
=& \sum_{j=0}^{\log(1/\epsilon)-1} \left((\textsc{Rank}_{C_{i,j}}(x) + 2\text{rank}_{\pi_{i,j+1}}(x) + \text{rank}_{\pi_{i,j}^{\text{remove}}}(x)) - \text{rank}_{\pi_{i,j}}(x)\right) \cdot 2^j \cdot \epsilon^2 R_i \\
&+ \left(\text{rank}_{\pi_{i,0}}(x) \cdot \epsilon^2 R - \text{rank}_{\pi_{H_i}}(x)\right) \\
&+ \left((\textsc{Rank}_{C_{i,\log(1/\epsilon)}}(x) + \text{rank}_{\pi_{i,\log(1/\epsilon)}^{\text{remove}}}(x)) - \text{rank}_{\pi_{i,\log(1/\epsilon)}}(x)\right) \cdot \epsilon R
\end{aligned}$$

In this case, the error from the buffer (the third term) is identically zero. This is because the buffer can store $3/\epsilon$ many elements each of weight ϵR_i, and we keep the total weight in H_i below $3R_i$. As a result, the buffer is never full. The elements in the buffer ($C_{i,\log(1/\epsilon)}$) plus the elements remove from the buffer ($\pi_{i,\log(1/\epsilon)}^{\text{remove}}$) equal exactly the elements that are ever inserted into the buffer ($\pi_{i,\log(1/\epsilon)}$).

Then for the first term, we use Lemma 4.2 instead of Lemma 4.1. We plug the following definition from Equation (4.6) into Lemma 4.2.

$$\Delta_{i,j}(x) \coloneqq (\textsc{Rank}_{C_{i,j}}(x) + 2\text{rank}_{\pi_{i,j+1}}(x) + \text{rank}_{\pi_{i,j}^{\text{remove}}}(x)) - \text{rank}_{\pi_{i,j}}(x).$$

Also we note because the total weight is at most $3R_i$, each compactor $C_{i,j}$ has at most $\frac{3R_i}{2^j \cdot \epsilon^2 \cdot R_i} = \frac{3}{2^j \cdot \epsilon^2}$ elements in it. Same as the previous proof, we have $\mathsf{E}[\text{rank}_{\pi_{i,j}}(x)] \leq \frac{\text{rank}_{\pi_{H_i}}(x)}{2^j \cdot \epsilon^2 R_i}$.

We get the contribution from the first term satisfies (using Fact 3.1),

$$\begin{aligned}\sum_{j=0}^{\log(1/\epsilon)-1} \epsilon^2 R_i \cdot 2^j \cdot \mathsf{E}\left[\Delta_{i,j}(x)^2\right]^{1/2} &\leq \sum_{j=0}^{\log(1/\epsilon)-1} \epsilon^2 R_i \cdot 2^j \cdot \mathsf{E}\left[\frac{\text{rank}_{\pi_{i,j}}(x) + t_i(x) \cdot \frac{3}{2^j \cdot \epsilon^2}}{k}\right]^{1/2} \\ &\leq \sum_{j=0}^{\log(1/\epsilon)-1} \epsilon^2 R_i \cdot 2^j \cdot \left(\frac{\text{rank}_{\pi_{H_i}}(x)}{2^j \cdot \epsilon^2 R_i \cdot k} + \frac{3t_i(x)}{2^j \cdot \epsilon^2 \cdot k}\right)^{1/2} \\ &= O(\epsilon) \cdot \sqrt{R_i \cdot \text{rank}_{\pi_{H_i}}(x) + t_i(x) \cdot R_i^2}.\end{aligned}$$

Here in the last step, we plug in $k = 1/\epsilon$ and sum over all j.

Finally, for the second term, we have that the rank in the sampled stream $\text{rank}_{\pi_{i,0}}(x) = \sum_{t=1}^{\text{rank}_{\pi_{H_i}}(x)} \text{Bernoulli}(p_i)$ where $p_i = \frac{1}{\epsilon^2 R_i}$. Thus,

$$\mathsf{E}\left[\left(\text{rank}_{\pi_{i,0}}(x) \cdot \epsilon^2 R_i - \text{rank}_{\pi_{H_i}}(x)\right)^2\right]^{1/2} \leq \epsilon^2 R_i \cdot \sqrt{\text{rank}_{\pi_{H_i}}(x) \cdot p(1-p)} \leq \epsilon \cdot \sqrt{R_i \cdot \text{rank}_{\pi_{H_i}}(x)}.$$

We conclude the proof by applying Fact 3.1 to sum up the contribution from different parts. □

Now that we have error bounds for the H_i's, Lemma 4.11 tells us, for any query x, the total error of the algorithm is at most

$$\mathsf{E}\left[\left|\text{rank}_\pi(x) - \sum_{i=0}^{\log_2(\epsilon n)} \textsc{Rank}_{H_i}(x)\right|^2\right]^{1/2} \leq O(\epsilon) \cdot \text{rank}_\pi(x).$$

This finishes the error analysis.

6.2 Extremely High Probability Bound Now let us prove that our algorithm answers a single query x approximately correct with high probability. Note from the expectation bound and Chebyshev's inequality, we know that our algorithm outputs an estimate that is within $(1 \pm \epsilon)\text{rank}_\pi(x)$ with constant probability. The naïve amplification of maintaining $\log(1/\delta)$ copies of our algorithm in parallel and outputting their median already gives an algorithm that succeeds with $1 - \delta$ probability.

In the following, we will change a few parameters of our algorithm and focus on getting an extremely high probability bound (double logarithmic dependency on δ). If combined with a small optimization which we will explain in Section 6.3, our final space complexity will be

$$\tilde{O}(\epsilon^{-1} \cdot \log(\epsilon n) \cdot (\log\log 1/\delta)^3).$$

The analysis uses the idea of analyzing top compactors deterministically, which first appears in [13].

Remark 6.1. (Connection to deterministic algorithms) *We note that the previous work [3], also gives an extremely high probability bound with space*

$$O(\epsilon^{-1} \cdot \log^2(\epsilon n) \cdot \log\log(1/\delta)).$$

Our algorithm improves the previous bound for not-too-small δ.

In comparison model, there are at most $n!$ many possible input streams π. So, for a single query, if the success probability $\delta < 1/n!$ (that is, $\log\log(1/\delta) = O(\log n)$), there must exist a fixing of the randomness that works for all possible input streams π. Then, by selecting $\delta < 1/n!$, we can obtain a deterministic algorithm. Using this connection, the previous algorithm implies the same $\tilde{O}(\epsilon^{-1} \cdot \log^3(\epsilon n))$ space deterministic upper bound, matching the state-of-the-art result of Zhang and Wang [19].

However, since our improvement is conditioned on δ being not-too-small, we are not able to directly improve upon the known upper bound for deterministic algorithms; in fact, our bound is actually worse in this extreme setting. Thus, improving the upper bound for deterministic quantile estimation in the relative-error regime remains an interesting open problem. We will discuss this more in the Open Problem section (Section 8).

Now let us specify the parameter changes for each H_i.

- We will set the block length of compactors $k = c \cdot \frac{(\log\log(1/\delta))^2}{\epsilon}$ with a large enough constant c (say $c = 128$) throughout the algorithm.
- For the sampler, we adjust the sampling probability of each sampler to $\frac{c\cdot\log(1/\delta)}{\epsilon^2\cdot R_i}$, as a result, now each element outputted by the sampler has weight $\frac{\epsilon^2\cdot R_i}{c\cdot\log(1/\delta)}$.
- We set $h_i \coloneqq \log\log(1/\delta) + \log(c/(k\epsilon^2))$. For each $0 \le j < h_i$, the compactor $C_{i,j}$ has elements of weight $w_{i,j} \coloneqq \frac{\epsilon^2\cdot R_i\cdot 2^j}{c\cdot\log(1/\delta)}$.
- The buffer now has size $3k$, each element in the buffer (which we denote by C_{i,h_i}) has weight $w_{i,h_i} = \frac{\epsilon^2\cdot R_i\cdot 2^{h_i}}{c\cdot\log(1/\delta)} = \frac{R_i}{k}$.

The rest of the algorithm stays the same.

In our analysis we define $h_i' = h_i - \log\log(1/\delta) = \log(1/(k\epsilon^2))$. For levels $0 \le j \le h_i'$, we are going to analyze compactor $C_{i,j}$'s using Chernoff bounds. But, for the top level compactors with $h_i' < j \le h_i$, we analyze the error deterministically as done in [13] and [3]. We defer the full analysis of the high probability bound to Appendix A.

6.3 Optimizing Space with Batch Insertions The algorithm we described in Section 4 gives a relative-error solution to the quantile estimation problem using space $\tilde{O}(\epsilon^{-1}\log n)$ with constant success probability. In this subsection, we add a simple optimization to our algorithm which reduces the space to $\tilde{O}(\epsilon^{-1}\log(\epsilon n))$. In particular, consider the following adjustment to our original sub-sketches H_i, which were described in Section 4:

- For each H_i, we create an additional "temporary" buffer B_i' which will store incoming stream elements *before* they are passed as a "batch" to the sampler. We set the batch-size $|B_i'| = 1/\epsilon$.
- To insert an element x into H_i, we first place x into the temporary buffer B_i', and once B_i' becomes full, we pass all of the elements stored in B_i' to the sampler (as a batch). The sampler operates in the same way as defined previously, i.e. it samples the elements in B_i' with probability $\frac{1}{\epsilon^2 R_i}$ (or alternatively, with probability $\frac{c\cdot\log(1/\delta)}{\epsilon^2 R_i}$ to obtain an extremely high-probability bound), and then inserts the sampled elements into the first level compactor $C_{i,0}$ together.

Now, recall that in Section 4.4, we defined a single time step to correspond to an insertion operation to $C_{i,0}$ or a resize of H_i. Also, recall that operation $\textsc{Insert}(C_{i,0}, x_1, x_2, \ldots, x_m)$ can handle $m \le s$ elements in one step where s is the current space for $C_{i,0}$. Since we never resize $C_{i,0}$ to a have space smaller than $1/\epsilon$ (see e.g. Equation (4.11)), the elements from B_i' can be inserted together in one time step even when they are all sampled.

As a result of the observation above, the total number of time steps goes from $\mathsf{poly}(n)$ to $\mathsf{poly}(\epsilon n)$. We will see in Section 6.4 that this reduces the space to $\tilde{O}(\epsilon^{-1}\log(\epsilon n))$ as claimed.

6.4 Space Allocation Analysis In this section, we will assume the optimization in Section 6.3 is applied so that the total number of time steps is $T = \mathsf{poly}(\epsilon n)$. When proving the total space is bounded, we now aim for a space bound of $\tilde{O}(\epsilon^{-1}\log(\epsilon n))$.

Analysis for offline general intervals. Recall that $t_{i,j}$ denotes the time step at which H_i resets for the j-th time. We defined the potential function $\phi_{i,j} = 1$ for $i = \log(\epsilon n)+1$ and $\phi_{i,j} = \sum_{m=\ell}^{r-1}\phi_{i+1,m}$ when $[t_{i,j}, t_{i,j+1})$ intersects with "children" intervals $[t_{i+1,\ell}, t_{i+1,\ell+1}]$, $[t_{i+1,\ell+1},\ t_{i+1,\ell+2}]$. Also the choice of space parameters $s_{i,t} = \epsilon^{-1}\cdot\left(\log\frac{\phi_{i,j}}{\phi_{i+1,m}} + 5\log(1/\epsilon)\right)$.

We will need Claim 4.14 and Claim 4.16, which we recall below:

Claim 4.14. *Within any interval $[t_{i+1,j}, t_{i+1,j+1})$, H_i has at most $3/\epsilon^2$ many insertions into $C_{i,0}$. We observe that this is true in general, i.e. not only for the case that all resets occur in tree-like intervals.*

Claim 4.16. *For any $0 \le i \le \log(\epsilon n)+1$, suppose there are ℓ_i intervals in total for H_i. Suppose that there are T time steps (counting insertion into $C_{i,0}$'s and resizes of H_i's) in total.*

Then,

$$\sum_{j=1}^{\ell_i} \phi_{i,j} \le 4^{\log(\epsilon n)+1-i} \cdot \ell_{\log(\epsilon n)+1}.$$

Specifically, the potential $\phi_{i,j}$ of any interval is bounded by $4^{\log(\epsilon n)+1} \cdot T = \mathsf{poly}(\epsilon n) \cdot T$.

We need to verify the two constraints are satisfied:

- (The total space is bounded.) Fix a time t. We also let j_i be the index of the interval $[t_{i,j_i}, t_{i,j_i+1})$ that contains t.

  From a similar telescoping sum as the tree-like case, we have

$$\begin{aligned}\sum_{i=0}^{\log(\epsilon n)} s_{i,t} &= \sum_{i=0}^{\log(\epsilon n)} \epsilon^{-1} \cdot \left(\log \frac{\phi_{i,j_i}}{\phi_{i+1,j_{i+1}}} + 5\log(1/\epsilon)\right) \\ &= \epsilon^{-1} \cdot \log_2 \frac{\phi_{0,j_0}}{1} + \epsilon^{-1} \cdot \log(\epsilon n) \cdot 5\log(1/\epsilon)\end{aligned}$$

  To bound the total space,$\lceil \log_2(1/\epsilon) \rceil \cdot \sum_{i=0}^{\log(\epsilon n)} s_{i,t}$[22], by $O(\epsilon^{-1} \log(\epsilon n) \log^2(1/\epsilon))$, we only need that $\phi_{0,j_0} = \mathsf{poly}(\epsilon n)$, which is exactly Claim 4.16 with $T = \mathsf{poly}(\epsilon n)$.

- (The space sequence is feasible.) Similar as the tree-like case, we consider any interval $[t_{i,j}, t_{i,j+1})$. Suppose it now intersects with children $[t_{i+1,\ell}, t_{i+1,\ell+1})$, $[t_{i+1,\ell+1}, t_{i+1,\ell+2})$, $\ldots$, $[t_{i+1,r-1}, t_{i+1,r})$. Again by Claim 4.14, we know that $|W_{i,j} \cap [t_{i+1,m}, t_{i+1,m+1})| \le 3/\epsilon^2 + 2$. Then,

$$\begin{aligned}\sum_{t \in W_{i,j}} 2^{-\epsilon \cdot s_{i,t}} &\le \sum_{m=\ell}^{r-1} \sum_{t \in W_{i,j} \cap [t_{i+1,m}, t_{i+1,m+1})} 2^{-\epsilon \cdot s_{i,t}} \\ &\le \sum_{m=\ell}^{r-1} (3/\epsilon^2 + 2) \cdot \frac{\phi_{i+1,m}}{\phi_{i,j}} \cdot \epsilon^5 \le 0.25 \cdot \frac{\sum_{m=\ell}^{r-1} \phi_{i+1,m}}{\phi_{i,j}} \le 0.25.\end{aligned}$$

  Thus, the space we allocate satisfies the premise of Lemma 4.5 (Equation (4.8)).

Analysis for online space allocation. Recall that for all unfinished intervals, we pretend that the interval ends (i.e. the corresponding sub-sketch H_i resets) at the *current* time t; then, we calculate all the potentials which we denote by $\phi_{i,j}^{(t)}$. We then round them up to the closest power of 2, which we denote by $\lceil\!\lceil \phi_{i,j}^{(t)} \rceil\!\rceil$, and let $\widehat{s}_{i,t} = \epsilon^{-1} \cdot \left(\log \frac{\lceil\!\lceil \phi_{i,j}^{(t)} \rceil\!\rceil}{\lceil\!\lceil \phi_{i+1,m}^{(t)} \rceil\!\rceil} + 5\log(1/\epsilon) + 5\log\log n\right)$.

To see why it works, we again need to verify the two constraints:

- (The total space is bounded.) This follows from exactly the same telescoping sum as before with $\phi_{i,j}$'s replaced by $\phi_{i,j}^{(t)}$'s, and the total space is now $O(\epsilon^{-1} \cdot \log(1/\epsilon) \cdot \log(\epsilon n)(\log(1/\epsilon) + \log\log n))$.
- (The space sequence is feasible.) This is the more subtle part. Again suppose interval $[t_{i,j}, t_{i,j+1})$ intersects with children $[t_{i+1,\ell}, t_{i+1,\ell+1})$, $[t_{i+1,\ell+1}, t_{i+1,\ell+2})$, $\ldots$, $[t_{i+1,r-1}, t_{i+1,r})$. First, we need to prove that we do not have too many resizes within each children interval. That is,

$$|W_{i,j} \cap [t_{i+1,m}, t_{i+1,m+1})| \le 3/\epsilon^2 + 2 + O(\log^2(\epsilon n)) \tag{6.13}$$

  This is because the only changing term in $\widehat{s}_{i,t}$ is $\log \frac{\lceil\!\lceil \phi_{i,j}^{(t)} \rceil\!\rceil}{\lceil\!\lceil \phi_{i+1,m}^{(t)} \rceil\!\rceil}$. Within each intersection $[t_{i,j}, t_{i,j+1}) \cap [t_{i+1,m}, t_{i+1,m+1})$, both numerator and denominator are increasing, and by Claim 4.16 have at most

[22]Note each hierarchy H_i has $\lceil \log_2(1/\epsilon) \rceil$ compactors in it and uses space $\lceil \log_2(1/\epsilon) \rceil \cdot s_{i,t}$

$O(\log(\epsilon n))$ different values. This means we have at most $O(\log^2(\epsilon n))$ extra resizes due to online allocation. This will be offset by the extra $5\log\log n$ term in $\widehat{s}_{i,t}$.

Second, we need to upper bound

$$\sum_{t\in W_{i,j}} 2^{-\epsilon\cdot s_{i,t}} \le \sum_{m=\ell}^{r-1} \sum_{\substack{t\\ t\in W_{i,j}\cap[t_{i+1,m},t_{i+1,m+1})}} 2^{-\epsilon\cdot s_{i,t}}$$
$$= \sum_{m=\ell}^{r-1} \sum_{\substack{t\\ t\in W_{i,j}\cap[t_{i+1,m},t_{i+1,m+1})}} \frac{[\![\phi_{i+1,m}^{(t)}]\!]}{[\![\phi_{i,j}^{(t)}]\!]}\cdot\epsilon^{-5}\cdot(\log n)^{-5}$$

Note $[\![\phi_{i,j}^{(t)}]\!]$ is monotone in t and has $O(\log(\epsilon n))$ many different values. For each possible value 2^x, we let $[a(x),b(x)]\subseteq[t_i,t_{j+1})$ be the time interval such that $[\![\phi_{i,j}^{(t)}]\!]=2^x$, and suppose it intersects with children $[t_{i+1,\ell(x)},t_{i+1,\ell(x)+1})$, $[t_{i+1,\ell(x)+1},t_{i+1,\ell(x)+2})$, $\ldots$, $[t_{i+1,r(x)-1},t_{i+1,r(x)})$.

Then for every $0\le x\le O(\log(\epsilon n))$, we have

$$\sum_{m=\ell(x)}^{r(x)-1} \sum_{\substack{t\\ t\in W_{i,j}\cap[t_{i+1,m},t_{i+1,m+1})\\ \cap[a(x),b(x)]}} \frac{[\![\phi_{i+1,m}^{(t)}]\!]}{[\![\phi_{i,j}^{(t)}]\!]}\cdot\epsilon^{-5}\cdot(\log n)^{-5}$$

(Here we use $|W_{i,j}\cap[t_{i+1,m},t_{i+1,m+1})|\le 3/\epsilon^2+2+O(\log^2(\epsilon n))$.)

$$\le \sum_{m=\ell(x)}^{r(x)-1} \left(3/\epsilon^2+2+O(\log^2(\epsilon n))\right)\cdot\frac{2\cdot\phi_{i+1,m}^{(b(x))}}{2^x}\cdot\epsilon^{-5}\cdot(\log n)^{-5}$$
$$\le 0.25\cdot\frac{\sum_{m=\ell(x)}^{r(x)-1}\phi_{i+1,m}^{b(x)}}{2^x}\le 0.25$$

Here the last step is because, by definition of $[\![\cdot]\!]$, we know that $\phi_{i,j}^{b(x)}\le 2^x$. On the other hand, we know $\phi_{i,j}^{b(x)}\ge\sum_{m=\ell(x)}^{r(x)-1}\phi_{i+1,m}^{b(x)}$. Thus, the space sequence is always feasible.

This finishes our proof. For the space allocation of the algorithm with extremely-high success probability, see Appendix A.

7 Lower and Upper bounds for the Top-R Quantiles Problem

7.1 Lower bound This subsection is devoted to the proof of Lemma 4.3. We first recall its statement:

LEMMA 4.3. *Let $n=|\pi|$ be the stream length. When $\epsilon=1/\sqrt{\log n}$, any comparison-based algorithm that solves the top quantiles problem for $R=\log n$ with $\delta<0.1$ failure probability requires at least $\Omega(\log n)$ space.*

The idea is to construct the hard input stream $\pi=x_1x_2\cdots x_n$ recursively, randomly, and adaptively. For any (possibly randomized) algorithm $\mathcal{A}$, we use $\mathcal{M}_t$ to denote the set of elements in its memory after reading x_t. We say that $\mathcal{A}$ remembers an element e at time t if and only if $\Pr[e\in\mathcal{M}_t]\ge 1/2$ where the probability is taken both over the randomized algorithm and the potentially randomized input stream.

Construction of the hard input stream. For any randomized algorithm $\mathcal{A}$, we can construct a length 2^k-1 (partial) input stream $\pi_k^{\mathcal{A}}$ as follows. Suppose here A takes n elements as input and we allow n to be larger or equal to 2^k-1.

- We first insert one element e as the first element in $\pi_k^{\mathcal{A}}$.
- Let $\mathcal{A}'$ be the algorithm $\mathcal{A}$ with the first input hard-coded to e. It takes $n-1$ inputs. We insert the stream $\pi_{k-1}^{\mathcal{A}'}$ and let all these elements be strictly larger than e.

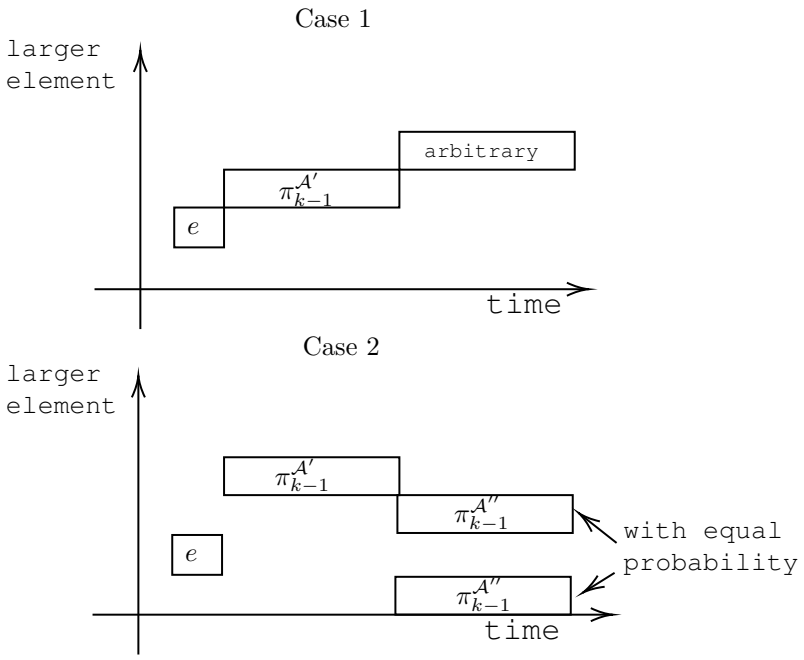

Figure 8: Construction of the hard stream $\pi_k^{\mathcal{A}}$.

- Now we have two cases:
  - Case 1: The algorithm $\mathcal{A}$ *remembers* e (by the definition above) at this time ($1 + |\pi_{k-1}^{\mathcal{A}'}| = 2^{k-1}$). In this case, we simply insert arbitrary $2^{k-1} - 1$ elements that are larger than element e and those elements in $\pi_{k-1}^{\mathcal{A}'}$.
  - Case 2: Otherwise, we let $\mathcal{A}''$ be the algorithm with input prefix hard-coded to the length-2^{k-1} partial stream we have constructed so far. We then insert $\pi_{k-1}^{\mathcal{A}''}$. With probability $1/2$, we let elements in it be greater than e but smaller than elements in $\pi_{k-1}^{\mathcal{A}'}$. With probability $1/2$, we let elements in $\pi_{k-1}^{\mathcal{A}'}$ be smaller than e.

In the base case, when $k = 0$, the stream is just empty. It is easy to see that $|\pi_k| = 1 + 2|\pi_{k-1}| = 1 + 2 \cdot (2^{k-1} - 1) = 2^k - 1$. See Figure 8 for an illustration.

The idea behind this construction is as follows. In Case 1, the algorithm $\mathcal{A}$ remembers e throughout the first half of the stream. Intuitively, this means that the algorithm $\mathcal{A}'$ has one less space in memory to store other elements in $\pi_{k-1}^{\mathcal{A}'}$. Alternatively, in Case 2, the algorithm A does not remember e through the first half. By letting e be either greater than $\pi_{k-1}^{\mathcal{A}''}$ or less than $\pi_{k-1}^{\mathcal{A}''}$ with equal probability, we create more uncertainty for rank queries. Thus, the algorithm $\mathcal{A}''$ must estimate the ranks in $\pi_{k-1}^{\mathcal{A}''}$ with a smaller error. In conclusion, from level k to $k-1$, either the space or error constraint of the algorithm are more strained. If we further expand out the recursion like this, finally in the base case, neither the space nor the error can be negative. Thus, the original algorithm $\mathcal{A}$ must either have a large enough space or large error in the beginning.

Space & Error Analysis. Now, we carry out the idea described above and prove the lower bound via induction.

LEMMA 7.1. *For any algorithm $\mathcal{A}$, let S_t be the space it use at time t. After reading the (partial) input stream $\pi_k^{\mathcal{A}} = x_1 x_2 \dots x_{2^k-1}$, we will choose an index i ($1 \le i \le |\pi_k^{\mathcal{A}}|$) depending on $\mathcal{A}$ but independent of the randomness of $\pi_k^{\mathcal{A}}$ and $\mathcal{A}$.*

Let $\widehat{\text{rank}}_{\pi_k^{\mathcal{A}}}(x_i)$ be any estimator for the rank of x_i based on the memory content of $\mathcal{A}$ (after reading this

partial stream). We can choose i so that we always have $\operatorname{rank}_{\pi_k^{\mathcal{A}}}(x_i) \le k$ *and*

$$\max_{1\le t\le|\pi_k^{\mathcal{A}}|} \mathsf{E}[S_t] + \mathsf{E}\left[\left(\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}}}(x_i) - \operatorname{rank}_{\pi_k^{\mathcal{A}}}(x_i)\right)^2\right] \ge 0.1k.$$

Proof. We prove this lemma by induction. In the base case where $k = 0$, this is true because none of the terms can be negative.

Now we verify it is true for both cases:

- Case 1: In this case, the first element e satisfies $\Pr[e \in \mathcal{M}_{2^{k-1}}] \ge 1/2$. Let $\mathcal{A}'$ be the algorithm $\mathcal{A}$ with the first element fixed to e as before, and $S'_1, S'_2, \dots, S'_{2^{k-1}-1}$ be its space after reading each element in $\pi_{k-1}^{\mathcal{A}'}$ (not counting the element e).

  Let $\pi_{k-1}^{\mathcal{A}'} = x'_1x'_2\dots x'_{2^{k-1}-1}$. By our induction hypothesis, we know that we can choose an i with $\operatorname{rank}_{\pi_k^{\mathcal{A}'}}(x'_i) \le k-1$ and

  $$\max_{1\le t\le 2^{k-1}-1} \mathsf{E}[S'_t] + \mathsf{E}\left[\left(\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}'}}(x'_i) - \operatorname{rank}_{\pi_k^{\mathcal{A}'}}(x'_i)\right)^2\right] \ge 0.1(k-1) \tag{7.14}$$

  for any estimator $\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}}}(x'_i)$ that only depends on the memory of $\mathcal{A}'$ after reading $\pi_{k-1}^{\mathcal{A}'}$.

  For all $1 \le t \le 2^{k-1}-1$, we have $\mathsf{E}[S_{t+1}] = \mathsf{E}[S'_t] + \Pr[e \in \mathcal{M}_t]$, and $\Pr[e \in \mathcal{M}_t] \ge \Pr[e \in \mathcal{M}_{2^{k-1}}] \ge 1/2$ (because once $\mathcal{A}$ removes one element from the memory it cannot get it back). Also, the second half of the stream consists of only elements with rank larger than e and $\pi_{k-1}^{\mathcal{A}'}$, so they are irrelavant for the rank of x'_i and the second term of (7.14) does not change. Thus, we conclude that, for $\pi_k^{\mathcal{A}} = x_1x_2\dots x_k$ (with $x_1 = e$), we have

  $$\max_{1\le t\le|\pi_k^{\mathcal{A}}|} \mathsf{E}[S_t] + \mathsf{E}\left[\left(\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}}}(x_{i+1}) - \operatorname{rank}_{\pi_k^{\mathcal{A}}}(x_{i+1})\right)^2\right] \ge 0.1(k-1) + 1/2 \ge 0.1k.$$

  Finally, we verify that $\operatorname{rank}_{\pi_k^{\mathcal{A}}}(x_{i+1}) = \operatorname{rank}_{\pi_k^{\mathcal{A}'}}(x'_i) + 1 \le k$.

- Case 2: In this case, the first element e satisfies $\Pr[e \in \mathcal{M}_{2^{k-1}}] \le 1/2$. Let $\mathcal{A}''$ be the algorihtm $\mathcal{A}''$ with prefix fixed to the concatenation of e and $\pi_{k-1}^{\mathcal{A}'}$, and $S''_1, S''_2, \dots, S''_{2^{k-1}-1}$ be the space after reading each element in $\pi_{k-1}^{\mathcal{A}''}$ (not counting the element e nor the elements in $\pi_{k-1}^{\mathcal{A}'}$).

  Let $\pi_{k-1}^{\mathcal{A}''} = x''_1x''_2\dots x''_{2^{k-1}-1}$. Again, by our induction hypothesis, we can select an index i (independent of all the randomness) such that

  $$\max_{1\le t\le 2^{k-1}-1} \mathsf{E}[S''_t] + \mathsf{E}\left[\left(\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}''}}(x''_i) - \operatorname{rank}_{\pi_k^{\mathcal{A}''}}(x''_i)\right)^2\right] \ge 0.1(k-1).$$

  For all $1 \le t \le 2^{k-1}-1$, we have $\mathsf{E}[S_{t+2^{k-1}}] \ge \mathsf{E}[S''_t]$. For the second term, let E_k be the event that $e \in \mathcal{M}_{2^k}$. We know that

  $$\begin{aligned}&\mathsf{E}\left[\left(\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}}}(x_{i+2^{k-1}}) - \operatorname{rank}_{\pi_k^{\mathcal{A}}}(x_{i+2^{k-1}})\right)^2 \;\middle|\; E_k\right]\\ \ge&\mathsf{E}\left[\left(\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}''}}(x''_i) - \operatorname{rank}_{\pi_k^{\mathcal{A}''}}(x''_i)\right)^2 \;\middle|\; E_k\right]\end{aligned}$$

  because $x_{i+2^{k-1}}$ and x''_i are the same element and e is hard-coded in A''. For any estimator $\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}}}(x_{i+2^{k-1}})$, we can always compare e with $x_{i+2^{k-1}}$ for free and get the estimator $\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}''}}(x''_i)$.

  On the other hand, we have

  $$\begin{aligned}&\mathsf{E}\left[\left(\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}}}(x_{i+2^{k-1}}) - \operatorname{rank}_{\pi_k^{\mathcal{A}}}(x_{i+2^{k-1}})\right)^2 \;\middle|\; \neg E_k\right]\\ \ge&\mathsf{E}\left[\left(\widehat{\operatorname{rank}}_{\pi_k^{\mathcal{A}''}}(x''_i) - \operatorname{rank}_{\pi_k^{\mathcal{A}''}}(x''_i)\right)^2 \;\middle|\; \neg E_k\right] + 0.25\end{aligned}$$

The reasoning is as follows. First, the event E_k only depends on e and $\pi_{k-1}^{\mathcal{A}'}$ and is independent of $\pi_{k-1}^{\mathcal{A}''}$. Second, conditioning on $\neg E_k$, any estimator $\widehat{\text{rank}}_{\pi_k^{\mathcal{A}}}(x_{i+2^{k-1}})$ can never distinguish the two possibilities of the relative order between e and $\pi_{k-1}^{\mathcal{A}''}$. (Because the element e is never able to be compared elements in $\pi_{k-1}^{\mathcal{A}''}$, including $x_{i+2^{k-1}}$.).

So, the optimal strategy for the estimator is to simply estimate the rank of $x_{i+2^{k-1}}$ in $\pi_k^{\mathcal{A}''}$ and plus 1/2. The squared error of this is lower bounded by $\mathsf{E}\left[\left(\widehat{\text{rank}}_{\pi_k^{\mathcal{A}''}}(x_i'') - \text{rank}_{\pi_k^{\mathcal{A}''}}(x_i'')\right)^2 \;\middle|\; \neg E_k\right]$ plus 1/4 squared error because the ground truth $\text{rank}_{\pi_k^{\mathcal{A}}}(x_{i+2^{k-1}})$ differs by 1 in these two indistinguishable possibilities.

Combining these and $\Pr[E_k] \le 1/2$, we conclude that

$$\mathsf{E}\left[\left(\widehat{\text{rank}}_{\pi_k^{\mathcal{A}}}(x_{i+2^{k-1}}) - \text{rank}_{\pi_k^{\mathcal{A}}}(x_{i+2^{k-1}})\right)^2\right] \ge \mathsf{E}\left[\left(\widehat{\text{rank}}_{\pi_k^{\mathcal{A}''}}(x_i'') - \text{rank}_{\pi_k^{\mathcal{A}''}}(x_i'')\right)^2\right] + 0.125.$$

Thus we have,

$$\max_{1\le t\le|\pi_k^{\mathcal{A}}|} \mathsf{E}[S_t] + \mathsf{E}\left[\left(\widehat{\text{rank}}_{\pi_k^{\mathcal{A}}}(x_{i+1}) - \text{rank}_{\pi_k^{\mathcal{A}}}(x_{i+1})\right)^2\right] \ge 0.1(k-1) + 0.125 \ge 0.1k.$$

□

Having this lemma and letting $k = \Theta(\log n)$, we see that for any algorithm $\mathcal{A}$, it either uses $\Omega(\log n)$ space, or for an element of rank at most $\log n$, introduce an error of at least $\Omega(\sqrt{\log n})$ with 0.99 probability (by Chebyshev's inequality). This proves Lemma 4.3.

7.2 Upper bound In this subsection, we provide a matching upper bound for the top-R quantiles problem, as defined in Table 2. For this problem, recall that the algorithm should provide the following guarantee: given any query $x \in \mathcal{U}$ such that $\text{rank}(x) \le R$, the algorithm should return an estimate $\widehat{\text{rank}}(x)$ such that $|\widehat{\text{rank}}(x) - \text{rank}(x)| \le \epsilon \cdot R$. Our algorithm for this problem will closely mimic the construction given earlier in Section 4, but we will replace the resizable elastic compactors with ordinary relative compactors (as originally defined in [3], see the description in Section 6).

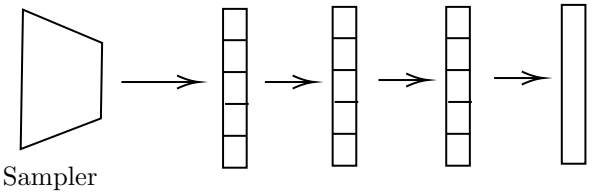

Figure 9: Structure of sketch $\mathcal{M}$ for Top Quantiles

As shown above, the sketch $\mathcal{M}$ will consist of the following components:

1. **Sampler:** For each new stream element x in π, the sampler inserts x into the first relative compactor C_0 with probability $\frac{1}{\epsilon^2 R}$, and discards x otherwise.

   As before, we attach a weight to each new element x: when x first appears in the stream π, x initially has weight 1. After passing through the sampler, each sampled element has weight $\epsilon^2 R$.

2. **Relative compactors:** for each $0 \le j < O(\log_2(1/\epsilon))$, we maintain a relative compactor C_j with block-size $k = O\left(\frac{1}{\epsilon\sqrt{\log(n)}}\right)$, such that each C_j can store at most $s = O\left(\frac{\sqrt{\log(n)}}{\epsilon}\right)$ elements.[23] After each compaction

[23]Throughout this subsection, we assume the range of parameter $\epsilon \le \frac{d}{\sqrt{\log n}}$ for some sufficiently small universal constant $d > 0$. This bound enables us to pick k to be an integer.

of C_j, C_j may output some elements $x_1, x_2, \ldots, x_m$, which we insert into the next compactor C_{j+1} in the hierarchy. Eventually, the output of the last compactor is inserted into the buffer.

For each $0 \le j < \log_2(1/\epsilon)$, elements in C_j all have weight $\epsilon^2 R \cdot 2^j$.

3. **Buffer:** At the end of the compactor hierarchy (i.e. after compactor $C_{O(\log(1/\epsilon))-1}$), we have a buffer B which stores $O(1/\epsilon)$ elements.

   Note that each element in the buffer has weight ϵR.

Additionally, observe that since the smallest $1/\epsilon$ elements in the buffer already have total weight R and we only need to estimate the first R ranks, we can afford to throw away other elements from the buffer (just as we did previously, in Section 4.2).

To estimate the rank for any query $x \in \mathcal{U}$, we use the same estimator as we previously in Section 4.2: let

$$\textsc{Rank}_{\mathcal{M}}(x) \coloneqq \sum_{j=0}^{O(\log(1/\epsilon))-1} \frac{2^j}{\epsilon^2 R} \cdot \textsc{Rank}_{C_j}(x) + \frac{1}{\epsilon R} \cdot \textsc{Rank}_B(x) \tag{7.15}$$

Now, we show that this construction will solve the Top-R Quantiles problem using space $\tilde{O}(\epsilon^{-1}\sqrt{\log n})$ with constant probability.

LEMMA 7.2. *Let π be the input stream and let $s = O(\epsilon^{-1}\sqrt{\log n})$ be the space allocated to each compactor C_j in the hierarchy, for $0 \le j \le O(\log(1/\epsilon)) - 1$. Then, we have that for any query $x \in \mathcal{U}$ such that $\operatorname{rank}_\pi(x) \le R$, our sketch $\mathcal{M}$ achieves standard deviation*

$$\mathbb{E}\left[|\textsc{Rank}_{\mathcal{M}}(x) - \operatorname{rank}_\pi(x)|^2\right] \le O(\epsilon) \cdot R$$

with constant probability.

Proof. Just as we did in the proof of Lemma 4.5, we begin by decomposing the error incurred by the rank estimate $\textsc{Rank}_{\mathcal{M}}(x)$. Let $C_0, \ldots, C_{O(\log(1/\epsilon))-1}$ be the compactors in $\mathcal{M}$, and $C_{O(\log(1/\epsilon))}$ denotes the buffer. As before, we use π_j to denote the input stream of C_j (which is the output stream of C_{j-1}).

$$\begin{aligned}
\textsc{Rank}_{\mathcal{M}}(x) - \operatorname{rank}_\pi(x) &= \sum_{j=0}^{O(\log(1/\epsilon))} \textsc{Rank}_{C_j}(x) \cdot 2^j \cdot \epsilon^2 \cdot R - \operatorname{rank}_\pi(x) \\
&= \sum_{j=0}^{O(\log(1/\epsilon))-1} \left((\textsc{Rank}_{C_j}(x) + 2\operatorname{rank}_{\pi_{j+1}}(x)) - \operatorname{rank}_{\pi_j}(x)\right) \cdot 2^j \cdot \epsilon^2 R \\
&\quad + \left(\operatorname{rank}_{\pi_0}(x) \cdot \epsilon^2 R - \operatorname{rank}_\pi(x)\right) \\
&\quad + \left(\textsc{Rank}_{C_{\log(1/\epsilon)}}(x) - \operatorname{rank}_{\pi_{\log(1/\epsilon)}}(x)\right) \cdot \epsilon R. \qquad (7.16)
\end{aligned}$$

We examine each term in Equation (7.16) above: the first term is the error contribution from the compactor hierarchy, and the second and third terms represent the error from the sampler and buffer, respectively. By the same argument as given in the proof of Lemma 4.5, we note that the third term (corresponding to the buffer) will never contribute to the error of the sketch. Next, we take a closer look at the first term. Let us denote

$$\Delta_j(x) = (\textsc{Rank}_{C_j}(x) + 2\operatorname{rank}_{\pi_{j+1}}(x)) - \operatorname{rank}_{\pi_j}(x)$$

Since each C_j is a relative compactor, we can apply Lemma 5 from [3] to bound the standard deviation. Moreover, recall that since we are guaranteed that $\operatorname{rank}_\pi(x) \le R$ and stream elements are sampled with probability $\frac{1}{\epsilon^2 R}$, we have $\mathbb{E}[\operatorname{rank}_{\pi_j}(x)] = \frac{\operatorname{rank}_{\pi_j}(x)}{\epsilon^2 R 2^j} \le \frac{1}{\epsilon^2 2^j}$. Thus, we get

$$\mathbb{E}\left[\Delta_j(x)^2\right]^{1/2} \le \mathbb{E}\left[\frac{\operatorname{rank}_{\pi_j}(x)}{k}\right]^{1/2} \le \sqrt{\frac{1}{\epsilon^2 \cdot 2^j \cdot k}}$$

Then, we see that the first term incurs standard deviation

$$\sum_{j=0}^{O(\log(1/\epsilon))-1} \mathbb{E}[\Delta_j(x)^2]^{1/2} \cdot 2^j \cdot \epsilon^2 R \leq \sum_{j=0}^{O(\log(1/\epsilon))-1} \sqrt{\frac{1}{\epsilon^2 \cdot 2^j \cdot k}} \cdot 2^j \cdot \epsilon^2 R \leq O(\epsilon) R \cdot \sqrt{\text{poly}(1/\epsilon) \cdot \sqrt{\log n}}$$

$$\leq O(\epsilon) \cdot R$$

Finally, we upper bound the error resulting from the sampler, i.e. the second term. Since we sample each stream element with probability $p = \frac{1}{\epsilon^2 R}$, we have that $\text{rank}_{\pi_0}(x) = \sum_{t=1}^{\text{rank}_\pi(x)} \text{Bernoulli}(p)$, i.e. we get

$$\mathbb{E}\left[(\text{rank}_{\pi_0}(x) \cdot \epsilon^2 R - \text{rank}_\pi(x))^2\right]^{1/2} \leq \epsilon^2 R \cdot \sqrt{R \cdot p(1-p)} \leq \epsilon R$$

Therefore, by applying Fact 3.1, we obtain the desired bound on the standard deviation. ∎

COROLLARY 7.3. *Suppose that $\epsilon \leq \frac{d}{\sqrt{\log n}}$ for some sufficiently small universal constant $d > 0$. Then the sketch $\mathcal{M}$ solves the Top-R Quantiles problem using space $O(\epsilon^{-1}\sqrt{\log(n)} \cdot \log(1/\epsilon))$ with constant probability.*

Proof. This follows directly from Lemma 7.2 together with Chebyshev's inequality. ∎

REMARK 7.4. *Using the analysis in Section 6.2, we can similarly obtain the same result with a high success probability by analyzing the top compactors deterministically.*

8 Open Problems

Several variants of the quantile estimation problem in the streaming model still remain open. In the next few paragraphs, we describe some open problems which relate to our work.

Obtaining optimal bounds for relative-error quantile estimation. Our algorithm achieves a $(1 \pm \epsilon)$ relative-error guarantee using total space $O(\epsilon^{-1} \log(\epsilon n) \cdot (\log \log n + \log(1/\epsilon)) \cdot (\log \log 1/\delta)^3)$. The main open question in the randomized, relative-error regime is to close the gap between our upper bound and the best known lower bound $\Omega(\epsilon^{-1} \log(\epsilon n))$.

Full-mergeability. Before we state the next open question, we recall the definition of a *fully-mergeable* sketch. Suppose we arbitrarily partition the input stream $\pi = \bigsqcup_{i=1}^{\ell} \pi_i$ and summarize each piece π_i of the stream using a separate sketch $\mathcal{M}_i$. Then, a sketch is fully-mergeable if any sequence of pairwise merging operations that merges these ℓ sketches $\mathcal{M}_1, ..., \mathcal{M}_\ell$ together results in a new sketch with essentially the same error and space guarantees as if the stream had been summarized using one sketch directly. Notably, this full-mergeability property is very important in practice: since massive datasets are often partitioned and stored on multiple servers, it is often useful to sketch the data stored locally on each device, and re-combine the sketches later to obtain a summary of the entire dataset. Since our sketch relies on an intricate dynamic space allocation scheme for the compactor hierarchy H_i stored for each scale $[\epsilon^{-1}2^i, \epsilon^{-1}2^{i+1}]$, it is not immediately clear how to "merge" the separate flushing patterns of the two sketches in order to obtain a valid resizable sketch H_i for each scale, while maintaining total space $\tilde{O}(\epsilon^{-1} \log(\epsilon n))$. With this in mind, we ask whether our sketch is fully mergeable, or if it is not, we ask if there is a way to adjust our construction to prove a mergeability guarantee.

Comparison-based, deterministic relative-error quantiles. Additionally, we ask whether it is possible to close the gap for comparison-based, deterministic relative-error quantile estimation. Currently, the only known lower bound is $\Omega\left(\frac{\log^2(\epsilon n)}{\epsilon}\right)$ [6], and the best known upper bound is $O\left(\frac{\log^3(\epsilon n)}{\epsilon}\right)$, via the merge-and-prune algorithm of [19]. To this end, we ask whether it is possible to apply our dynamic space allocation and sketch resizing techniques to improve the space used in the deterministic setting. Perhaps, as an intermediate step, it is an interesting open problem to resolve the deterministic space complexity for the Top Quantiles problem that we defined in Table 2. Again, the only known lower bound is $\Omega\left(\frac{\log(\epsilon n)}{\epsilon}\right)$ from the additive error setting [6], and the best upper bound is $O\left(\frac{\log^2(\epsilon n)}{\epsilon}\right)$ from the same merge-and-prune algorithm.

Non-comparison-based relative-error quantiles. Although comparison-based algorithms are more powerful and can handle quantile queries for an unknown (totally-ordered) universe, there are many applications where the universe in fixed and known in advance. Then, the goal shifts from minimizing the number of elements stored to a more fine-grained measure: the number of bits used in total.

Previously, [5] gave a non-comparison-based bq-summary algorithm using $O\left(\frac{\log(\epsilon n)\log U}{\epsilon}\right)$ words of memory (where each word is $\log U + \log(\epsilon n)$ bits), while the offline lower bound is $\Omega\left(\frac{\log n}{\epsilon}\right)$ words. Although our algorithm almost closes the gap for randomized relative-error quantile estimation, we ask whether there is a non-comparison-based approach that could result in simpler deterministic algorithm.

As a remark, the bq-summary algorithm is based on the q-digest, which is a deterministic non-comparison-based algorithm for the additive-error quantiles problem [17]. Recently, the upper bound for q-digest was improved from $O\left(\frac{\log U}{\epsilon}\right)$ to only $O\left(\frac{1}{\epsilon}\right)$ words [12], which is optimal. We ask whether it is possible to use similar ideas as in [12] to improve the upper bound in the deterministic relative-error setting as well.

Tight characterization for resizable sketches. Suppose a resizable sketch is given space s_t after reading the t-th element in the input. Our resizable sketch for the Top Quantiles problem (Lemma 4.5) works for space sequences $s_1, s_2, \ldots, s_n$ satisfying $\sum_{t=1}^{n} 2^{-s_i} \leq 0.5$. By following a more careful analysis and adjusting the block size k, one could actually improve this condition to $\sum_{t=1}^{n} 2^{-\epsilon^2 s_t^2} \leq 0.5$. On the other hand, it is conceivable that one might be able to prove a lower bound saying that under the opposite condition for the space sequence (i.e. $\sum_{t=1}^{n} 2^{-\epsilon^2 s_t^2} > 0.5$), there is no comparison-based algorithm for the Top Quantiles Problem.

Motivated by this, we ask whether one can give such a tight characterization for resizable sketches for the Top Quantile Problem, or analogously, if we can prove similar space-sequence bounds for other natural streaming problems.

References

[1] Rakesh Agarwal and Arun Swami. A one-pass space-efficient algorithm for finding quantiles. In *Proceedings of 7th International Conference on Management of Data (COMAD-95)*, 1995.

[2] Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedsing of the 23rd AC SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems (PODS 2004)*, pages 286–296, 2004.

[3] Graham Cormode, Zohar Karnin, Edo Liberty, Justin Thaler, and Pavel Veselỳ. Relative error streaming quantiles. *Journal of the ACM*, 70(5):1–48, 2023.

[4] Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Effective computation of biased quantiles over data streams. In *21st International Conference on Data Engineering (ICDE'05)*, pages 20–31. IEEE, 2005.

[5] Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Space-and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 263–272, 2006.

[6] Graham Cormode and Pavel Veselỳ. A tight lower bound for comparison-based quantile summaries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 81–93, 2020.

[7] Niv Dayan, Ioana Bercea, Pedro Reviriego, and Rasmus Pagh. Infinifilter: Expanding filters to infinity and beyond. In *Proc. ACM Manag. Data, Vol. 1, No. 2, Article 140*, 2023.

[8] David Felber and Rafail Ostrovsky. A randomized online quantile summary in o(1/epsilon * log(1/epsilon)) words. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015), volume 40 of Leibniz International Proceedings in Informatics*, pages 775–785, 2015.

[9] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2):58–66, 2001.

[10] Elena Gribelyuk, Pachara Sawettamalya, Hongxun Wu, and Huacheng Yu. Simple & optimal quantile sketch: Combining greenwald-khanna with khanna-greenwald. In *Proceedings of the ACM on Management of Data, Volume 2, Issue 2 (PODS 2024)*, 2024.

[11] Anupam Gupta and Francis Zane. Counting inversions in lists. In *SODA*, volume 3, pages 253–254, 2003.

[12] Meghal Gupta, Mihir Singhal, and Hongxun Wu. Optimal quantile estimation: beyond the comparison model. *arXiv preprint arXiv:2404.03847*, 2024.

[13] Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*, pages 71–78. IEEE, 2016.

[14] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass with limited memory. In *ACM SIGMOD record, volume 27*, pages 426–435, 1998.
[15] Rasmus Pagh, Gil Segev, and Udi Wieder. How to approximate a set without knowing its size in advance. In *IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, 2013.
[16] Viswanath Poosala, Venkatesh Ganti, and Yannis E. Ioannidis. Approximate query answering using histograms. In *IEEE Data Eng. Bull., 22(4):5-15*, 1999.
[17] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, 2004.
[18] Simons Institute Workshop: Data Structures and Optimization for Fast Algorithms. Sketching and algorithm design, October 2023. `https://simons.berkeley.edu/talks/rasmus-pagh-university-copenhagen-2023-10-13`.
[19] Qi Zhang and Wei Wang. An efficient algorithm for approximate biased quantile computation in data streams. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 1023–1026, 2007.
[20] Ying Zhang, Xuemin Lin, Jian Xu, Flip Korn, and Wei Wang. Space-efficient relative error order sketch over data streams. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 51–51. IEEE, 2006.

A Proof of the Extremely High Probability Bound

We first recall the parameter changes for extremely high probability bound:

- We will set the block length of compactors $k = c \cdot \frac{(\log\log(1/\delta))^2}{\epsilon}$ with a large enough constant c (say $c = 128$) throughout the algorithm.
- For the sampler, we adjust the sampling probability of each sampler to $\frac{c\cdot\log(1/\delta)}{\epsilon^2\cdot R_i}$, as a result, now each element outputted by the sampler has weight $\frac{\epsilon^2\cdot R_i}{c\cdot\log(1/\delta)}$.
- We set $h_i \coloneqq \log\log(1/\delta) + \log(c/(k\epsilon^2))$. For each $0 \le j < h_i$, the compactor $C_{i,j}$ has elements of weight $w_{i,j} \coloneqq \frac{\epsilon^2\cdot R_i\cdot 2^j}{c\cdot\log(1/\delta)}$.
- The buffer now has size $3k$, each element in the buffer (which we denote by C_{i,h_i}) has weight $w_{i,h_i} = \frac{\epsilon^2\cdot R_i\cdot 2^{h_i}}{c\cdot\log(1/\delta)} = \frac{R_i}{k}$.

Error Analysis Recall that $\pi_{i,j}$ is the input stream of compactor $C_{i,j}$ and π_{H_i} is the input stream of sub-sketch H_i while π'_{H_i} is the output of the sub-sketch H_i. $w_{i,j}$ is the weight of elements in $C_{i,j}$.

In addition, we let $\pi^I_{H_i}$ be the elements in the original input stream π that we directly inserts into H_i. The difference with π_{H_i} is that, $\pi^I_{H_i}$ do not contains those elements in $\pi'_{H_{i-1}}$.

We first show that with probability $1-\delta$, the rank of query x in every $\pi_{i,j}$ cannot be too large. This will later be used to upper bound the number of relevant compactions.

LEMMA A.1. *With probability $1 - O(\delta)$, for all $0 \le i \le \log(\epsilon\cdot\mathrm{rank}_\pi(x))$ and $0 \le j \le h_i$, we have*

$$\mathrm{rank}_{\pi_{i,j}}(x) \le 2\cdot\frac{\mathrm{rank}_\pi(x)}{w_{i,j}} \quad \textit{and} \quad \mathrm{rank}_{\pi_{H_i}}(x) \le 2\cdot\mathrm{rank}_\pi(x). \tag{A.1}$$

where $w_{i,j} \coloneqq \frac{\epsilon^2\cdot R_i\cdot 2^j}{\log(1/\delta)}$ is the weight of elements in $C_{i,j}$.

Proof. We prove by induction for $0 \le i < \log(\epsilon\cdot\mathrm{rank}_\pi(x))$ that, conditioning on Equation (A.1) holds for all i' with $0 \le i' < i$, it holds for i with probability $1 - \delta\cdot\frac{R_i}{\mathrm{rank}_\pi(x)}$ for some absolute constant c. Then a union bound over all i proves this lemma.

Now suppose this holds for all i' with $0 \le i' < i$. We first completely unroll the expression for $\mathrm{rank}_{\pi_{i,j}}(x)$ and $\mathrm{rank}_{\pi_{H_i}}(x)$:

First for π_{H_i}, we decompose the rank of x similar to Lemma 4.8:

$$\begin{aligned}
\mathrm{rank}_{\pi_{H_i}}(x) &= \mathrm{rank}_{\pi^I_{H_i}}(x) + \mathrm{rank}_{\pi'_{H_{i-1}}}(x) \\
&= \mathrm{rank}_{\pi^I_{H_i}}(x) + \sum_{j=0}^{h_{i-1}} w_{i-1,j} \cdot \mathrm{rank}^{\mathrm{remove}}_{\pi_{i,j}}(x) \\
&\le \mathrm{rank}_{\pi^I_{H_i}}(x) + w_{i-1,0} \cdot \mathrm{rank}_{\pi_{i-1,0}}(x) \\
&\text{(Note } w_{i-1,j} = 2w_{i-1,j-1} \text{ and } \mathrm{rank}^{\mathrm{remove}}_{\pi_{i,h_i}}(x) \le 2\mathrm{rank}_{\pi_{i,h_i-1}}(x).) \\
&\qquad + \sum_{j=0}^{h_{i-1}-1} w_{i-1,j+1} \cdot (2\mathrm{rank}_{\pi_{i-1,j+1}}(x) + \mathrm{rank}^{\mathrm{remove}}_{\pi_{i-1,j}}(x) - \mathrm{rank}_{\pi_{i-1,j}}(x)) \\
&\le \mathrm{rank}_{\pi^I_{H_i}}(x) + w_{i-1,0} \cdot \mathrm{rank}_{\pi_{i-1,0}}(x) + \sum_{j=0}^{h_{i-1}} w_{i-1,j} \cdot \Delta_{i-1,j}(x)
\end{aligned}$$

where $\pi^I_{H_i}$ and π'_{H_i} are defined at the beginning of this section. In the last line we use the fact that

$$\Delta_{i,j}(x) \coloneqq \left(\textsc{Rank}_{C_{i,j}}(x) + 2 \cdot \mathrm{rank}_{\pi_{i,j+1}}(x) + \mathrm{rank}_{\pi^{\mathrm{remove}}_{i,j}}(x)\right) - \mathrm{rank}_{\pi_{i,j}}(x).$$

For the third term $\sum_{j=0}^{h_{i-1}} w_{i-1,j} \cdot \Delta_{i-1,j}(x)$, note there are at most $\frac{\mathrm{rank}_{\pi_{i-1,j}}(x) + t_i(x) \cdot S}{k}$ important flushes for $C_{i,j}$. Here $S \le \frac{3R_i}{w_{i,j}}$ is a upper bound on the number of elements in $C_{i,j}$. By Claim 4.10, we know $t_i(x) \le 2 \cdot \frac{\mathrm{rank}_{H_i}(x)}{R_i}$. Together with our inductive hypothesis, we get

$$\frac{\mathrm{rank}_{\pi_{i-1,j}}(x) + t_i(x) \cdot S}{k} \le \frac{2 \cdot \mathrm{rank}_{\pi}(x) + 6 \cdot \mathrm{rank}_{H_i(x)}}{w_{i,j} \cdot k} \le \frac{14 \cdot \mathrm{rank}_{\pi}(x)}{w_{i,j} \cdot k}.$$

Thus conditioning on any fixed realization of $\pi_{i-1,j}$, $\Delta_{i-1,j}(x)$ is still statistically dominated by

$$\sum_{t=1}^{14 \cdot \mathrm{rank}_{\pi}(x)/(w_{i,j} \cdot k)} (\mathrm{Bernoulli}(1/2) - 1/2)). \tag{A.2}$$

On the other hand, $\Delta_{i-1,j}(x)$ is always deterministically at most

$$\frac{14 \cdot \mathrm{rank}_{\pi}(x)}{w_{i,j} \cdot k} \cdot w_{i,j} = \frac{14 \cdot \mathrm{rank}_{\pi}(x)}{k}. \tag{A.3}$$

We will choose when to use these two bounds. Then, we keep unrolling the second term.

$$w_{i-1,0} \cdot \mathrm{rank}_{\pi_{i-1,0}}(x) = w_{i-1,0} \cdot \sum_{t=1}^{\mathrm{rank}_{\pi_{H_{i-1}}}(x)} \mathrm{Bernoulli}(1/w_{i,0}),$$

which by our inductive hypothesis, is stochastic dominated by

$$\mathrm{rank}_{\pi_{H_{i-1}}}(x) + w_{i,0} \cdot \left(\sum_{t=1}^{2 \cdot \mathrm{rank}_{\pi}(x)} \mathrm{Bernoulli}(1/w_{i,0}) - 1/w_{i,0} \right)$$

even when conditioned on any realization on $\pi_{H_{i-1}}$.

We keep unrolling $\text{rank}_{\pi_{H_{i-1}}}(x)$ in the same way as we unroll $\text{rank}_{\pi_{H_i}}(x)$. Eventually, we get that $\text{rank}_{\pi_{H_i}}(x)$ is stochastically dominated by the following sum (which we will explain how we get it):

$$\begin{aligned} Z_i = &\sum_{i'=1}^{i} \text{rank}^I_{H_{i'}}(x) \\ &+ \sum_{i'=1}^{i-1} \left(\sum_{j=0}^{h_{i'}-1} w_{i',j} \cdot \left(\sum_{t=1}^{14\cdot\text{rank}_\pi(x)/(w_{i',j}\cdot k)} \text{Bernoulli}(1/2) - 1/2 \right) \cdot \mathbb{1}[w_{i',j} \le w_{i,h'_i}] \right. \\ &+ \sum_{j=0}^{h_{i'}-1} \frac{14\text{rank}_{\pi(x)}}{k} \cdot \mathbb{1}[w_{i',j} > w_{i,h'_i}] \\ &\left. + w_{i',0} \cdot \left(\sum_{t=1}^{2\cdot\text{rank}_\pi(x)} \text{Bernoulli}(1/w_{i',0}) - 1/w_{i',0} \right) \right) \end{aligned}$$

Here we are using bound (A.2) for those $\Delta_{i',j}(x)$ with $w_{i',j} \le w_{i,h'_i}$ and bound (A.3) for those with $w_{i',j} > w_{i,h'_i}$. This is the idea of analyzing top compactors deterministically in [13].

We know that $h'_i = h_i - \log\log 1/\delta$. Thus there are $\log\log 1/\delta - (i - i')$ many j's pairs with $w_{i',j} > w_{i,h'_i}$ for each i. In total, there are $(\log\log 1/\delta)^2$ many such (i', j) pairs. As a result,

$$\sum_{j=0}^{h_{i'}-1} \frac{14\text{rank}_\pi(x)}{k} \cdot \mathbb{1}[w_{i',j} > w_{i,h'_i}] \le \frac{14\text{rank}_\pi(x)}{k} \cdot (\log\log 1/\delta)^2 \le \epsilon \cdot \text{rank}_\pi(x). \tag{A.4}$$

We can without loss of generality assume that $\epsilon < 0.1$, so that this is less than $0.1 \cdot \text{rank}_\pi(x)$.

Note we also have $\sum_{i'=1}^{i} \text{rank}^I_{H_{i'}}(x) \le \text{rank}_\pi(x)$. The rest is a sum of the deviation of Bernoulli random variables. We upper bound it using the Chernoff bound for subguassian random variables. Because Bernoulli random variable X is $\text{Var}[X]$ subgaussian, by Fact 3.2, we know Z is also $\text{Var}[Z]$-subgaussian.

The following calculation bounds its variance:

$$\text{Var}[Z_i] = \sum_{i'=1}^{i-1} \left(\sum_{j=0}^{h_i-1} w^2_{i',j} \cdot \frac{14 \cdot \text{rank}_\pi(x)}{w_{i',j} \cdot k} \cdot \mathbb{1}[w_{i',j} \le w_{i,h'_i}] \right) + w^2_{i',0} \cdot 2\text{rank}_\pi(x) \cdot \frac{1}{w_{i',0}}.$$

(Because $w_{i',j}$'s are exponential in j.)

$$= O(1) \cdot \sum_{i'=1}^{i-\log\log(1/\delta)} w_{i',h_{i'}} \cdot \frac{\text{rank}_\pi(x)}{k} + \log\log(1/\delta) \cdot w_{i,h'_i} \cdot \frac{\text{rank}_\pi(x)}{k} + w_{i',0} \cdot \text{rank}_\pi(x)$$

(Because $w_{i',j}$'s are exponential in i' as well.)

$$= O(1) \cdot \left(\log\log(1/\delta) \cdot w_{i,h'_i} \cdot \frac{\text{rank}_\pi(x)}{k} + w_{i-1,0} \cdot \text{rank}_\pi(x) \right).$$

Since we know $\frac{w_{i,h'_i}}{k} \le \frac{R_i}{k^2 \log(1/\delta)} \le \frac{\epsilon^2}{c\cdot\log(1/\delta)\cdot\log\log(1/\delta)} \cdot R_i$ and $w_{i-1,0} \le \frac{\epsilon^2}{c\cdot\log(1/\delta)} \cdot R_i$, plug them in we get $\text{Var}[Z_i] \le O(1) \cdot \frac{\epsilon^2}{c\cdot\log(1/\delta)} \cdot R_i \cdot \text{rank}_\pi(x)$. (Here c is the constant factor we picked in our algorithm.) Then, we upper bound the error we have using the Chernoff bound for subgaussian random variables. We conclude that there exists an absolute constant $d > 0$ such that

$$\begin{aligned} \Pr[Z_i \ge 0.1 \cdot \text{rank}_{\pi_i}(x)] &\le \exp\left(-\frac{(0.1 \cdot \text{rank}_\pi(x))^2}{2\text{Var}[\text{Z}_\text{i}]} \right) \\ &\le \exp\left(-d \cdot \frac{c \cdot \log(1/\delta)}{\epsilon^2} \cdot \frac{\text{rank}_\pi(x)}{R_i} \right) \\ &\text{(As long as we pick } c > 2/d.\text{)} \quad \le \delta \cdot \frac{R_i}{\text{rank}_\pi(x)} \cdot \frac{1}{\log(1/\epsilon) + \log\log(1/\delta)} \end{aligned}$$

This finishes the proof for $\mathrm{rank}_{\pi_{H_i}}(x)$. For those $\mathrm{rank}_{\pi_{i,j}}(x)$, we note that the same decomposition shows that $\mathrm{rank}_{\pi_{i,j}}(x)$ is stochastically dominated by Z_{i+1}. Since there are at most $O(\log(1/\delta)+\log\log(1/\delta))$ many j's for each i, we can simply union bound over all those j's. This finishes the proof of the lemma. ∎

Then let us prove that, conditioning on $\mathrm{rank}_{H_i}(x)$'s and $\mathrm{rank}_{\pi_{i,j}}(x)$'s being not too large, our estimate $\widehat{\mathrm{rank}_\pi(x)}$ is approximately correct with probability $1-O(\delta)$.

LEMMA A.2. *For any input stream π, assuming that the space we allocate to each H_i satisfies the premise of Lemma 4.5 (Equation (4.8)), For any query x, with probability $1-O(\delta)$, we have*

$$\left|\mathrm{rank}_\pi(x) - \sum_{i=0}^{\log_2(\epsilon n)} \textsc{Rank}_{H_i}(x)\right| \le O(\epsilon)\cdot \mathrm{rank}_\pi(x).$$

Proof. First of all, let $\ell = \log(\epsilon\cdot\mathrm{rank}_\pi(x))$. From the same logic as Remark 4.12, we know that it suffices to prove that with probability $1-O(\delta)$,

$$\left|\mathrm{rank}_\pi(x) - \sum_{i=0}^{\ell} \textsc{Rank}_{H_i}(x)\right| \le O(\epsilon)\cdot \mathrm{rank}_\pi(x).$$

We first unroll the error similar to Lemma 4.11 and Lemma A.1.

$$\begin{aligned}
&\mathrm{rank}_\pi(x) - \sum_{i=0}^{\ell} \textsc{Rank}_{H_i}(x)\\
=&\sum_{i=0}^{\ell} \mathrm{rank}_{\pi_{H_i}}(x) - \mathrm{rank}_{\pi_i'}(x) - \textsc{Rank}_{H_i}(x)\\
=&\sum_{i=0}^{\ell}\left(\left(\mathrm{rank}_{\pi_{H_i}}(x) - w_{i,0}\cdot\mathrm{rank}_{\pi_{i,0}}(x)\right) + \sum_{j=0}^{h_i} w_{i,j}\cdot\left(\mathrm{rank}_{\pi_{i,j}}(x) - 2\mathrm{rank}_{\pi_{i,j+1}}(x) - \mathrm{rank}_{\pi_{i,j}^{\mathrm{remove}}}(x)\right)\right)
\end{aligned}$$

From Lemma A.1, we know that with probability $1-O(\delta)$, Equation (A.1) holds. Conditioning on it holds and follow the same argument as Lemma A.1, we know that the absolute value of error is stochastically dominated by $|Z_\ell'| + \sum_{i=0}^{\ell}\sum_{j=0}^{h_i-1}\frac{14\mathrm{rank}_{\pi(x)}}{k}\cdot\mathbb{1}\left[w_{i,j} > w_{\ell,h_\ell'}\right]$, where Z_ℓ' is defined as following:

$$\begin{aligned}
Z_\ell' = \sum_{i=0}^{\ell}\Bigg(&\sum_{j=0}^{h_i-1} w_{i,j}\cdot\left(\sum_{t=1}^{14\cdot\mathrm{rank}_\pi(x)/(w_{i,j}\cdot k)} \mathrm{Bernoulli}(1/2) - 1/2\right)\cdot\mathbb{1}\left[w_{i,j}\le w_{\ell,h_\ell'}\right]\\
&+ w_{i,0}\cdot\left(\sum_{t=1}^{2\mathrm{rank}_\pi(x)} \mathrm{Bernoulli}(1/w_{i,0}) - 1/w_{i,0}\right)\Bigg).
\end{aligned}$$

Since Z_ℓ' is only Z_ℓ shifted, they have the same variance. We get that

$$\begin{aligned}
\mathrm{Var}\left[Z_\ell'\right] = \mathrm{Var}\left[Z_\ell\right] &\le O(1)\cdot\frac{\epsilon^2}{c\cdot\log(1/\delta)}\cdot R_\ell\cdot\mathrm{rank}_\pi(x).\\
\text{(Because } R_\ell = \mathrm{rank}_\pi(x) \text{ by defintion.)}\quad &= O(1)\cdot\frac{\epsilon^2}{c\cdot\log(1/\delta)}\cdot R_\ell^2
\end{aligned}$$

As Z_ℓ' is the sum of Bernoulli random variables, it is $\mathrm{Var}\left[Z_\ell'\right]$-subgaussian. We then apply Chernoff bound for subgaussian random variables and conclude that when constant c is picked to be large enough, we have

$$\Pr\left[Z_\ell' \ge \epsilon\cdot\mathrm{rank}_\pi(x)\right] \le \delta.$$

For the $\sum_{i=0}^{\ell}\sum_{j=0}^{h_i-1}\frac{14\mathrm{rank}_{\pi(x)}}{k}\cdot\mathbb{1}\left[w_{i,j} > w_{\ell,h_\ell'}\right]$ part, we use Equation (A.4) to conclude that it is at most $\epsilon\cdot\mathrm{rank}_\pi(x)$. This finishes the proof. ∎

Space Allocation Since now we set $k = c \cdot \frac{(\log\log(1/\delta))^2}{\epsilon}$, the condition for each compactor $C_{i,j}$ becomes as follows: Before it resets, let $s_1, s_2, \ldots, s_\ell$ be the space parameters after each resize/insert operation. Then we need $\sum_{t=1}^{\ell} 2^{-s_t/k} = \sum_{t=1}^{\ell} 2^{-c\cdot\frac{\epsilon}{(\log\log(1/\delta))^2}\cdot s_t} \le 1$. (Note this is the same condition as Lemma 4.1.)

For each subset H_i, suppose $s_1, s_2, \ldots, s_\ell$ is the sequence of space parameters after each reize / insertion from the sampler into $C_{i,0}$, instead of $\sum_{t=1}^{\ell} 2^{-\epsilon\cdot s_t} \le 0.5$ (Equation (4.8)), we now require that $\sum_{t=1}^{\ell} 2^{-c\cdot\frac{\epsilon}{(\log\log(1/\delta))^2}\cdot s_t} \le 0.5$. Same as the proof of Lemma 4.5, as long as this condition is satisfied for each H_i, the condition for each compactor $C_{i,j}$ will be then satisfied.

When allocating space, we now define

$$\widehat{s}_{i,t} = \epsilon^{-1}\cdot(\log\log(1/\delta))^3\cdot\left(\log\frac{[\![\phi_{i,j}^{(t)}]\!]}{[\![\phi_{i+1,m}^{(t)}]\!]} + 5\log(1/\epsilon) + 5\log\log n\right).$$

Here $(\log\log(1/\delta))^2$ comes from the larger value for k, the one extra $\log\log(1/\delta)$ factor is because the weight of elements in $C_{i,0}$ now reduces by a factor of $\log(1/\delta)$, so the bound in Equation (6.13) now becomes $|W_{i,j}\cap[t_{i+1,m}, t_{i+1,m+1})| \le 3\cdot\frac{\log(1/\delta)}{\epsilon^2} + 2 + O(\log^2(\epsilon n))$. Then we need the extra $\log\log(1/\delta)$ factor to account for this.

With these changes, the rest of the proofs follow from the exact same calculations as that of Section 6.4. For completeness, we repeat the calculations here:

- (The total space is bounded.) Fix a time t. Let j_i be the index of the interval $[t_{i,j_i}, t_{i,j_i+1})$ that contains t.

$$\begin{aligned}\sum_{i=0}^{\log(\epsilon n)}\widehat{s}_{i,t} &= \sum_{i=0}^{\log(\epsilon n)}\epsilon^{-1}\cdot(\log\log(1/\delta))^3\cdot\left(\log\frac{[\![\phi_{i,j}^{(t)}]\!]}{[\![\phi_{i+1,m}^{(t)}]\!]} + 5\log(1/\epsilon) + 5\log\log n\right)\\ &= O\left(\epsilon^{-1}\cdot(\log\log(1/\delta))^3\cdot\log_2\frac{\phi_{0,j_0}}{1} + \epsilon^{-1}\cdot(\log\log(1/\delta))^3\cdot\log(\epsilon n)\cdot(\log(1/\epsilon)+\log\log n)\right)\\ &= O\left(\epsilon^{-1}\cdot\log(\epsilon n)\cdot(\log\log n + \log(1/\epsilon))\cdot(\log\log 1/\delta)^3\right)\end{aligned}$$

  The last step follow from $\phi_{0,j_0} = \mathsf{poly}(\epsilon n)$ (invoking Claim 4.16 with $T = \mathsf{poly}(\epsilon n)$). The total space is then $O\left(\epsilon^{-1}\cdot\log(1/\epsilon)\cdot\log(\epsilon n)\cdot(\log\log n + \log(1/\epsilon))\cdot(\log\log 1/\delta)^3\right)$ because each H_i uses space $\lceil\log_2(1/\epsilon)\rceil\cdot\widehat{s_{i,t}}$.

- (The space sequence is feasible.) Again suppose interval $[t_{i,j}, t_{i,j+1})$ intersects with children $[t_{i+1,\ell}, t_{i+1,\ell+1})$, $[t_{i+1,\ell+1}, t_{i+1,\ell+2})$, $\ldots$, $[t_{i+1,r-1}, t_{i+1,r})$. We use the new bound

$$|W_{i,j}\cap[t_{i+1,m}, t_{i+1,m+1})| \le 3\cdot\frac{\log(1/\delta)}{\epsilon^2} + 2 + O(\log^2(\epsilon n))$$

  Then,

$$\begin{aligned}\sum_{t\in W_{i,j}} 2^{-c\cdot\frac{\epsilon}{(\log\log(1/\delta))^2}\cdot s_t} &\le \sum_{m=\ell}^{r-1}\sum_{t\in W_{i,j}\cap[t_{i+1,m},t_{i+1,m+1})} 2^{-c\cdot\frac{\epsilon}{(\log\log(1/\delta))^2}\cdot s_t}\\ (c\ge 1.)\qquad &= \sum_{m=\ell}^{r-1}\sum_{t\in W_{i,j}\cap[t_{i+1,m},t_{i+1,m+1})}\frac{[\![\phi_{i+1,m}^{(t)}]\!]}{[\![\phi_{i,j}^{(t)}]\!]}\cdot\epsilon^{-5}\cdot(\log n)^{-5}\cdot(\log(1/\delta))^{-1}\end{aligned}$$

  Exactly the same as Section 6.4, note $[\![\phi_{i,j}^{(t)}]\!]$ is monotone in t and has $O(\log(\epsilon n))$ many different values. For each possible value 2^x, we let $[a(x), b(x)] \subseteq [t_i, t_{j+1})$ be the time interval such that $[\![\phi_{i,j}^{(t)}]\!] = 2^x$, and suppose it intersects with children $[t_{i+1,\ell(x)}, t_{i+1,\ell(x)+1})$, $[t_{i+1,\ell(x)+1}, t_{i+1,\ell(x)+2})$, $\ldots$, $[t_{i+1,r(x)-1}, t_{i+1,r(x)})$.

Then for every $0 \le x \le O(\log(\epsilon n))$, we have

$$\sum_{m=\ell(x)}^{r(x)-1} \sum_{\substack{t \\ t \in W_{i,j} \cap [t_{i+1,m}, t_{i+1,m+1}) \\ \cap [a(x), b(x)]}} \frac{[\![\phi_{i+1,m}^{(t)}]\!]}{[\![\phi_{i,j}^{(t)}]\!]} \cdot \epsilon^{-5} \cdot (\log n)^{-5} \cdot (\log n)^{-5} \cdot (\log(1/\delta))^{-1}$$

(Here we use $|W_{i,j} \cap [t_{i+1,m}, t_{i+1,m+1})| \le 3 \cdot \frac{\log(1/\delta)}{\epsilon^2} + 2 + O(\log^2(\epsilon n))$.)

$$\le \sum_{m=\ell(x)}^{r(x)-1} \left(3 \cdot \log(1/\delta)/\epsilon^2 + 2 + O(\log^2(\epsilon n))\right) \cdot \frac{2 \cdot \phi_{i+1,m}^{(b(x))}}{2^x} \cdot \epsilon^{-5} \cdot (\log n)^{-5} \cdot (\log n)^{-5} \cdot (\log(1/\delta))^{-1}$$

$$\le 0.25 \cdot \frac{\sum_{m=\ell(x)}^{r(x)-1} \phi_{i+1,m}^{b(x)}}{2^x} \le 0.25$$

Thus, the space sequence is always feasible.