Space-Efficient Online Computation of Quantile Summaries

Michael Greenwald*
Computer & Information Science Department
University of Pennsylvania
200 South 33rd Street
Philadelphia, PA 19104
greenwald@cis.upenn.edu

Sanjeev Khanna†
Computer & Information Science Department
University of Pennsylvania
200 South 33rd Street
Philadelphia, PA 19104
sanjeev@cis.upenn.edu

ABSTRACT

An ϵ-approximate quantile summary of a sequence of N elements is a data structure that can answer quantile queries about the sequence to within a precision of ϵN.

We present a new online algorithm for computing ϵ-approximate quantile summaries of very large data sequences. The algorithm has a worst-case space requirement of $O(\frac{1}{\epsilon}\log(\epsilon N))$. This improves upon the previous best result of $O(\frac{1}{\epsilon}\log^2(\epsilon N))$. Moreover, in contrast to earlier deterministic algorithms, our algorithm does not require a priori knowledge of the length of the input sequence.

Finally, the actual space bounds obtained on experimental data are significantly better than the worst case guarantees of our algorithm as well as the observed space requirements of earlier algorithms.

1. INTRODUCTION

We study the problem of space-efficient computation of quantile summaries of very large data sets in a single pass. A quantile summary consists of a small number of points from the input data sequence, and uses those quantile estimates to give approximate responses to an *arbitrary* quantile query.

Summaries of large data sets have long been used by programmers motivated by limited memory resources. Elementary summaries, such as running averages or standard deviation, are typically sufficient only for simple applications. The mean and variance are often either insufficiently descriptive, or are too sensitive to outliers and other anomalous data. For such cases, online algorithms are necessary to generate quantile summaries that use little space and provide reasonably accurate approximations to the distribution function induced by the input data sequence [6, 1, 5, 13, 2].

1.1 Quantile Estimation for Database Applications

Recent work (e.g. [8, 9, 12]) has highlighted the importance of quantile estimators for database users and implementors. Quantile estimates are used to estimate the size of intermediate results, to allow query optimizers to estimate the cost of competing plans to resolve database queries. Parallel databases attempt to partition the data into value ranges such that the size of all partitions are roughly equal. Quantile estimates can be used to choose the ranges without inspecting the actual data. Quantile estimates have several other uses in databases as well. User-interfaces may estimate result sizes of queries, and provide feedback to users. This feedback may prevent expensive and incorrect queries from being issued, and may flag discrepancies between the user's model of the database and its actual content. Quantile estimates are also used by database users to characterize the distribution of real world data sets.

The existing body of work has also identified particular properties that quantileestimators require in order to be useful for these database applications — properties that may not be strictly necessary when estimating quantiles in other domains. Some of the desirable properties are as follows. (1) The algorithm should provide tunable and explicit a priori guarantees on the precision of the approximation. We say that a quantile summary is *ϵ-approximate* if it can be used to answer any quantile query to within a precision of ϵN. In other words, for any given rank r, an ϵ-approximate quantile summary returns a value whose rank r' is guaranteed to be within the interval $[r - \epsilon N, r + \epsilon N]$. (2) The algorithm should be *data independent*. Neither its guarantees should be affected by the arrival order or distribution of values, nor should it require a priori knowledge of the size of the dataset. (3) The algorithm should execute in a single pass over the data. (4) The algorithm should have as small a memory footprint as possible. We note here that the memory footprint applies to temporary storage during the computation. We can always construct an ϵ-approximate summary of size $O(1/\epsilon)$ as follows. We first construct an $\epsilon/2$-approximate summary. For i from 0 to $\frac{2}{\epsilon}$, query this

*Supported in part by DARPA under Contract #F39502-99-1-0512, and by the National Science Foundation under Grant ANI-00-81901.
†Supported in part by an Alfred P. Sloan Research Fellowship.

ACM SIGMOD 2001 May 21-24, Santa Barbara, California USA

summary for each $i\frac{\epsilon}{2}$ quantile. It is easy to see that the set of responses constitutes an ϵ-approximate summary.

1.2 Previous Work

Several earlier works have made progress towards meeting the above-mentioned requirements. Manku, Rajagopalan, and Lindsay [8] present a single-pass algorithm that constructs an ϵ-approximate quantile summary. The algorithm strictly guarantees a precision of ϵN, but it requires an advance knowledge of N, the size of the data set. It requires $O(\frac{1}{\epsilon}\log^2(\epsilon N))$ space. In [8] the same authors present an algorithm that does not require an advance knowledge of N. However, they must give up the deterministic guarantee on accuracy. Instead, they provide only a probabilistic guarantee that the quantile estimates are within the desired precision.

Gibbons, Matias, and Poosala [4] estimate quantiles under a different error metric, but their algorithm requires multiple passes over the data. Similarly, Chaudhuri, Motwani, and Narsayya [3] require multiple passes and only provide probabilistic guarantees.

Munro and Paterson [10], building on the earlier work of Pohl [7], showed that any algorithm that exactly computes the ϕ-quantile of a sequence of N data elements in only p passes, requires a space of $\Omega(N^{1/p})$. Thus the notion of approximate quantiles is inherently necessary for obtained sub-linear space algorithms.

Many researchers have also addressed the problem of determining the smallest number of comparisons that are necessary for computing a ϕ-quantile. We refer the reader to a nice survey article by Paterson [11] for an overview of results in this area.

1.3 Our Results

We design and analyze a new online algorithm for computing an ϵ-approximate quantile summary of large data sequences. The algorithm has a worst-case space requirement of $O(\frac{1}{\epsilon}\log(\epsilon N))$, thus improving upon the previous best result of $O(\frac{1}{\epsilon}\log^2(\epsilon N))$. Moreover, in contrast to earlier deterministic algorithms, our algorithm does not require a priori knowledge of the length of the input sequence.

Our approach is based on a novel data structure that effectively maintains the range of possible ranks for *each* quantile that we store. This differs from previous approaches that implicitly assumed that the error in stored quantiles was distributed roughly uniformly throughout the distribution of observed values. By explicitly maintaining the possible range of rank values for each quantile, our algorithm is able to adaptively handle new observations: values observed near tightly constrained quantiles are more likely to be dropped and new values observed near loosely constrained quantiles are more likely to be stored. Intuitively speaking, the improved behavior of our algorithm is based on the fact (which we prove) that no input sequence can be "bad" across the entire distribution at once. In other words, an input sequence cannot persistently present new observations that must be stored without allowing us to safely delete old stored observations.

We also note here that our algorithm can be parallelized in a straightforward manner to deal with the scenario where a system of P independent processors analyzes P disjoint streams derived from a parent sequence. Due to space considerations, we will omit the details of this implementation in this version.

Finally, we study the performance of our algorithm from an empirical perspective. The actual space bounds obtained on experimental data are significantly better than both the worst case guarantees of our algorithm as well as the observed space requirements of earlier algorithms. For example, when summarizing uniformly random data with $\epsilon = 0.001$ and $N = 10^7$, our algorithm used an order of magnitude less memory than the best previously known algorithm.

2. THE NEW ALGORITHM

We will assume without any loss of generality that a new observation arrives after each unit of time and thus we will use n to denote both the number of observations (elements of the data sequence) that have been seen so far as well as the current time. Our algorithm maintains a summary data structure $\mathsf{S} = \mathsf{S}(n)$ at all times, and we denote by $s = s(n)$, the total space used by it. Finally, we denote the given precision requirement by ϵ.

2.1 The Summary Data Structure

At any point in time n, the data structure $\mathsf{S}(n)$ consists of an ordered sequence of tuples which correspond to a subset of the observations seen thus far. For each observation v in S, we maintain implicit bounds on the minimum and the maximum possible rank of the observation v among the first n observations. Let $r_{min}(v)$ and $r_{max}(v)$ denote respectively the lower and upper bounds on the rank of v among the observations seen so far. Specifically, S consists of tuples $t_0, t_1, ..., t_{s-1}$ where each tuple $t_i = (v_i, g_i, \Delta_i)$ consists of three components: (i) a value v_i that corresponds to one of the elements in the data sequence seen thus far, (ii) the value g_i equals $r_{min}(v_i) - r_{min}(v_{i-1})$, and (iii) Δ_i equals $r_{max}(v_i) - r_{min}(v_i)$. We ensure that, at all times, the maximum and the minimum values are part of the summary. In other words, v_0 and v_{s-1} always correspond to the minimum and the maximum elements seen so far. It is easy to see that $r_{min}(v_i) = \sum_{j\le i} g_j$ and $r_{max}(v_i) = \sum_{j\le i} g_j + \Delta_i$. Thus $g_i + \Delta_i - 1$ is an upper bound on the *total* number of observations that may have fallen between v_{i-1} and v_i. Finally, observe that $\sum_i g_i$ equals n, the total number of observations seen so far.

Answering Quantile Queries: A summary of the above form can be used in a straightforward manner to provide ϵ-approximate answers to quantile queries. The proposition below forms the basis of our approach.

PROPOSITION 1. *Given a quantile summary* S *in the above form, a* ϕ*-quantile can always be identified to within an error of* $\max_i(g_i + \Delta_i)/2$.

PROOF. Let $r = \lceil \phi n \rceil$ and let $e = \max_i(g_i + \Delta_i)/2$. We will search for an index i such that $r - e \le r_{min}(v_i)$ and $r_{max}(v_i) \le r + e$. Clearly, such a value v_i approximates the

ϕ-quantile to within the claimed error bounds. We now argue that such an index i must always exist. First, consider the case $r > n - e$. We have $r_{min}(v_{s-1}) = r_{max}(v_{s-1}) = n$, and therefore $i = s - 1$ has the desired property. Otherwise, when $r \leq n - e$, we choose the smallest index j such that $r_{max}(v_j) > r + e$. It follows that $r - e \leq r_{min}(v_{j-1})$. If $r - e > r_{min}(v_{j-1})$ then $r_{max}(v_j) = r_{min}(v_{j-1}) + g_j + \Delta_j > r_{min}(v_{j-1}) + 2e$; a contradiction to our assumption that $e = \max_i(g_i + \Delta_i)/2$. By assumption, $r_{max}(v_{j-1}) \leq r + e$, therefore $j - 1$ is an example of an index i with the above described property. □

The following is an immediate corollary.

COROLLARY 1. *If at any time n, the summary $\mathsf{S}(n)$ satisfies the property that $\max_i(g_i + \Delta_i) \leq 2\epsilon n$, then we can answer any ϕ-quantile query to within an ϵn precision.*

At a high level, our algorithm for maintaining the quantile summary proceeds as follows. Whenever the algorithm sees a new observation, it inserts in the summary a tuple corresponding to this observation. Periodically, the algorithm performs a sweep over the summary to "merge" some of the tuples into their neighbors so as to free up space. The heart of the algorithm is in the merge phase where we maintain several conditions that allow us to bound the space used by S at any time. By Corollary 1, it suffices to ensure that at all times $max_i(g_i + \Delta_i) \leq 2\epsilon n$. Motivated by this consideration, we will say that an individual tuple is *full* if $g_i + \Delta_i = \lfloor 2\epsilon n \rfloor$. The *capacity* of an individual tuple is the maximum number of observations that can be counted by g_i before the tuple becomes full.

Bands: In order to minimize the number of tuples in our summary, our general strategy will be to delete tuples with small capacity and preserve tuples with large capacity. The merge phase will free up space by merging tuples with small capacities into tuples with "similar" or larger capacities. We say that two tuples, t_i and t_j, have similar capacities, if $\log \text{capacity}(t_i) \approx \log \text{capacity}(t_j)$.

This notion of similarity partitions the possible values of Δ into *bands*. Roughly speaking, we try to divide the Δs into bands that lie between elements of $(0, \frac{1}{2}2\epsilon n, \frac{3}{4}2\epsilon n, \ldots \frac{2^i-1}{2^i}2\epsilon n, \ldots\ 2\epsilon n - 1, 2\epsilon n)$. (These boundaries correspond to capacities of $2\epsilon n, \epsilon n, \frac{1}{2}\epsilon n, \ldots\ \frac{1}{2^i}\epsilon n, \ldots, 8, 4, 2, 1.$) As we will see shortly, it is useful to define bands in a way that ensures the property that if two Δs are ever in the same band, they never appear in different bands as n increases. Therefore, for α from 1 to $\lceil \log 2\epsilon n \rceil$, we let $p = \lfloor 2\epsilon n \rfloor$ and we define band_α to be the set of all Δ such that $p - 2^\alpha - (p \bmod 2^\alpha) < \Delta \leq p - 2^{\alpha-1} - (p \bmod 2^{\alpha-1})$. The $(p \bmod 2^\alpha)$ term holds the borders between bands static as n increases. We define band_0 to simply be p. As a special case, we consider the first $1/2\epsilon$ observations, with $\Delta = 0$, to be in a band of their own. Figure 1 shows the band boundaries as $2\epsilon n$ goes from 24 to 34. We will denote by $\text{band}(t_i, n)$ the band of Δ_i at time n, and by $\text{band}_\alpha(n)$ all tuples (or equivalently, the Δ values associated with these tuples) that have a band value of α.

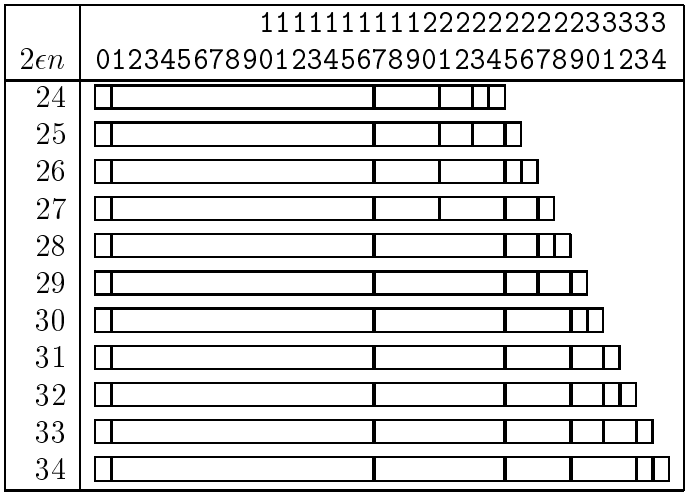

Figure 1: Band boundaries as $2\epsilon n$ progresses from 24 to 34. The rightmost band in each row is band 0.

PROPOSITION 2. *At any point in time n and for any $\alpha \geq 1$, $band_\alpha(n)$ contains either 2^α or $2^{\alpha-1}$ distinct values of Δ.*

PROOF. The $\text{band}_\alpha(n)$ is bounded below by $2\epsilon n - 2^\alpha - (2\epsilon n \bmod 2^\alpha)$ and above by $2\epsilon n - 2^{\alpha-1} - (2\epsilon n \bmod 2^{\alpha-1})$. If $2\epsilon n \bmod 2^\alpha < 2^{\alpha-1}$, then $2\epsilon n \bmod 2^\alpha = 2\epsilon n \bmod 2^{\alpha-1}$, and $\text{band}_\alpha(n)$ contains $2^\alpha - 2^{\alpha-1} = 2^{\alpha-1}$ distinct values of Δ. If $2\epsilon n \bmod 2^\alpha \geq 2^{\alpha-1}$, then $2\epsilon n \bmod 2^\alpha = 2^{\alpha-1} + (2\epsilon n \bmod 2^{\alpha-1})$, and $\text{band}_\alpha(n)$ contains $2^{\alpha-1} + 2^{\alpha-1} = 2^\alpha$ distinct values of Δ. □

A Tree Representation: We will find it useful to impose a tree structure over the tuples. Given a summary $\mathsf{S} = \langle t_0, t_1, ..., t_{s-1} \rangle$, the tree T associated with S contains a node V_i for each t_i and a special root node R. The parent of a node V_i is the node V_j such that j is the least index greater than i with $\text{band}(t_j) > \text{band}(t_i)$. If no such index exists, then the node R is set to be the parent. All children (and all descendants) of a given node V_i have Δ values larger than Δ_i. The following two properties of T can be easily verified.

PROPOSITION 3. *The children of any node in T are always arranged in non-increasing order of band in S.*

PROPOSITION 4. *For any node V, the set of all its descendants in T forms a contiguous segment in S.*

2.2 Operations

We now describe the various operations that we perform on our summary data structure. We start with a description of external operations:

2.2.1 External Operations

QUANTILE(ϕ) To compute an ϵ-approximate ϕ-quantile from the summary $\mathsf{S}(n)$ after n observations, compute the rank, $r = \lceil \phi n \rceil$. Find i such that both $r - r_{min}(v_i) \leq \epsilon n$ and $r_{max}(v_i) - r \leq \epsilon n$ and return v_i.

INSERT(v) Find the smallest i, such that $v_{i-1} \leq v < v_i$, and insert the tuple $(v, 1, \lfloor 2\epsilon n \rfloor)$, between t_{i-1} and t_i. Increment s. As a special case, if v is the new minimum or the maximum observation seen, then insert $(v, 1, 0)$.

INSERT(v) maintains correct relationships between g_i, Δ_i, $r_{min}(v_i)$ and $r_{max}(v_i)$. Consider that if v is inserted before v_i, the value of $r_{min}(v)$ may be as small as $r_{min}(v_{i-1}) + 1$, and hence $g_i = 1$. Similarly, $r_{max}(v)$ may be as large as the *current* $r_{max}(v_i)$, which in turn is bounded by $\lfloor 2\epsilon n \rfloor$. Note that $r_{min}(v_i)$ and $r_{max}(v_i)$ get increased by 1 after insertion.

```
COMPRESS()
  for i from s − 2 to 0 do
    if ((BAND(Δ_i, 2εn) ≤ BAND(Δ_{i+1}, 2εn)) &&
        (g_i^* + g_{i+1} + Δ_{i+1} < 2εn)) then
      DELETE all descendants of t_i and the tuple t_i itself;
    end if
  end for
end COMPRESS
```

Figure 2: Pseudo-code for COMPRESS

2.2.2 Internal Operations

DELETE(v_i) To delete the tuple (v_i, g_i, Δ_i) from S, replace (v_i, g_i, Δ_i) and $(v_{i+1}, g_{i+1}, \Delta_{i+1})$ by the new tuple $(v_{i+1}, g_i + g_{i+1}, \Delta_{i+1})$, and decrement s.

DELETE() correctly maintains the relationships between g_i, Δ_i, $r_{min}(v_i)$ and $r_{max}(v_i)$. Deleting v_i has no effect on $r_{min}(v_{i+1})$ and $r_{max}(v_{i+1})$, so DELETE(v_i) should simply preserve $r_{min}(v_{i+1})$ and $r_{max}(v_{i+1})$. The relationship between $r_{min}(v_{i+1})$ and $r_{max}(v_{i+1})$ is preserved as long as Δ_{i+1} is unchanged. Since $r_{min}(v_{i+1}) = \sum_{j \leq i+1} g_j$, and we delete g_i, we must increase g_{i+1} by g_i to keep $r_{min}(v_{i+1})$. All other entries are unaltered by this operation.

COMPRESS() The operation COMPRESS tries to merge together a node and all its descendants into either its parent node or into its right sibling. The property that we must ensure is that the tuple that results after this merging is not full. By Proposition 4, we know that a node and its children always form a contiguous sequence of tuples in $\mathsf{S}(n)$. Let g_i^* denote the sum of g-values of the tuple t_i and all its decendants in T. It is easy to see that merging t_i and its descendants (by DELETEing them) into t_{i+1} would result in t_{i+1} being updated to $(v_{i+1}, g_i^* + g_{i+1}, \Delta_{i+1})$. We would like to ensure that this resulting tuple is not full. We say that a pair of adjacent tuples $t_i, t_{i+1} \in \mathsf{S}(n)$ is *mergeable* if $(g_i^* + g_{i+1} + \Delta_{i+1} < 2\epsilon n)$ and $\text{band}(t_i, n) \leq \text{band}(t_{i+1}, n)$. At a high level, the COMPRESS operation iterates over the tuples in $\mathsf{S}(n)$ from right to left, and whenever it finds a mergeable pair t_i, t_{i+1}, it merges t_i as well as all tuples that are descendants of t_i in $\mathsf{T}(n)$ into t_{i+1}. Note that pairs of tuples that are not mergeable at some point in time may become so at a later point in time as the term $\lfloor 2\epsilon n \rfloor$ increases over time. Figure 2 gives pseudo-code describing this operation.

Note that since DELETE() and COMPRESS() never alter the Δ of surviving tuples, it follows that Δ_i of any quantile entry remains unchanged once it has been inserted.

COMPRESS() inspects tuples from right (highest index) to left. Therefore, it first combines children (and their entire subtree of descendants) into parents. It combines siblings only when no more children can be combined into the parent.

```
Initial State
  S ← ∅; s = 0; n = 0.
Algorithm
To add the n + 1st observation, v, to summary S(n):

  if (n ≡ 0 mod 1/(2ε)) then
    COMPRESS();
  end if
  INSERT(v);
  n = n + 1;
```

Figure 3: Pseudo-code for the algorithm

2.3 Analysis

It is easy to see that the data structure above maintains an ϵ-approximate quantile summary at each point in time. The INSERT as well as COMPRESS operations always ensure that $g_i + \Delta_i \leq 2\epsilon n$ at any point in time. We will now establish that the total number of tuples in the summary S after n observations have been seen is bounded by $(11/2\epsilon)\log(2\epsilon n)$.

We start by defining a notion of coverage. We say that a tuple t in the quantile summary S covers an observation v at any time n if either the tuple for v has been directly merged into t_i or a tuple t that covered v has been merged into t_i. Moreover, a tuple always covers itself. It is easy to see that the total number of observations covered by t_i is exactly given by $g_i = g_i(n)$. The lemmas below give some simple properties concerning coverage observations by various tuples.

LEMMA 1. *At no point in time, a tuple from* band α covers *an observation from a band* $> \alpha$.

PROOF. Suppose at some time n, the event described in the lemma occurs. The COMPRESS subroutine never merges a tuple t_i into an adjacent tuple t_{i+1} if the band of t_i is greater than the band of t_{i+1}. Thus the only way in which this event can occur is if it at some point in time, say m, we have $\text{band}(t_i, m) \leq \text{band}(t_{i+1}, m)$ and at the current time n, we have $\text{band}(t_i, n) > \text{band}(t_{i+1}, n)$. We now argue that this cannot occur since if at any point in time ℓ, $\text{band}(t_i, \ell) = \text{band}(t_{i+1}, \ell)$, then for all $n \geq \ell$, we must have $\text{band}(t_i, n) = \text{band}(t_{i+1}, n)$. The borders between bands are static, except when two bands combine (forever). Band 0 is always new. If $2\epsilon n \equiv 2^{\alpha-1} \bmod 2^\alpha$, then α and $\alpha + 1$ combine into the $\alpha+1$ band (α is a unique band for given n). All bands $> \alpha + 1$ remain the same. Because band 0 is always new, all bands $\beta < \alpha$ become $\beta + 1$. In other words, borders are always removed, never added. □

LEMMA 2. *At any point in time n, and for any integer α, the total number of observations covered cumulatively by all tuples with band values in $[0..\alpha]$ is bounded by $2^\alpha/\epsilon$.*

PROOF. By Proposition 2, each $\text{band}_\beta(n)$ contains at most 2^β distinct values of Δ. There are no more than $1/2\epsilon$ observations with any given Δ, so at most $2^\beta/2\epsilon$ observations were inserted with $\Delta \in \text{band}_\beta$. By Lemma 1, no observations from bands $> \alpha$ will be covered by a node from α. Therefore the nodes in question can cover, at most, the total number of observations from all bands $\leq \alpha$. Summing over all $\beta \leq \alpha$ yields an upper bound of $2^{\alpha+1}/2\epsilon = 2^\alpha/\epsilon$. □

The next lemma shows that for any given band value α, only a small number of nodes can have a child with that band value.

LEMMA 3. *At any time n and for any given α, there are at most $3/2\epsilon$ nodes in $\mathsf{T}(n)$ that have a child with band value of α. In other words, there are at most $3/2\epsilon$ parents of nodes from $\text{band}_\alpha(n)$.*

PROOF. Let $m_{\min}$ and $m_{\max}$, respectively denote the earliest and the latest times at which an observation in $\text{band}_\alpha(n)$ could be seen. It is easy to verify that $m_{\min} = (2\epsilon n - 2^\alpha - (2\epsilon n \bmod 2^\alpha))/2\epsilon$ and $m_{\max} = (2\epsilon n - 2^{\alpha-1} - (2\epsilon n \bmod 2^{\alpha-1}))/2\epsilon$. Thus, any parent of a node in $\text{band}_\alpha(n)$ must have $\Delta_i < 2\epsilon m_{\min}$.

Fix a parent node V_i with at least one child in $\text{band}_\alpha(n)$ and let V_j be the rightmost such child. Denote by m_j the time at which the observation corresponding to V_j was seen.

We will show that at least a $(2\epsilon/3)$-fraction of all observations that arrived after time $m_{\min}$ can be uniquely mapped to the pair(V_i, V_j). This in turn implies that no more than $3/2\epsilon$ such V_i's can exist, thus establishing the lemma. The main idea underlying our proof is that the fact that COMPRESS() did not merge V_j into V_i implies there must be a large number of observations that can be associated with the parent-child pair (V_i, V_j).

We first argue that $g^*_j(n) + \sum_{k=j+1}^{i-1} g_k(n) \geq g^*_{i-1}(n)$. If $j = i-1$, it is trivially true. Otherwise, observe that any tuple t_k that lies between t_j and t_i must belong to a band less than or equal to α — else V_k, and not V_i, would be the parent of V_j. Therefore, $\sum_{k=j+1}^{i-1} g_k(n) \geq g^*_{i-1}(n)$ and the claim follows..

Now since COMPRESS() did not merge V_j into V_i, it must be the case that $g^*_{i-1}(n) + g_i(n) + \Delta_i > 2\epsilon n$. Using the claim above, we can conclude that $g^*_j(n) + \sum_{k=j+1}^{i-1} g_k(n) + g_i(n) + \Delta_i > 2\epsilon n$. Also, at time m_j, we had $g_i(m_j) + \Delta_i < 2\epsilon m_j$. Since m_j is at most $m_{\max}$, it must be that

$$g^*_j(n) + \sum_{k=j+1}^{i-1} g_k(n) + (g_i(n) - g_i(m_j)) > 2\epsilon(n - m_{\max}).$$

Finally observe that for any other such parent-child pair $V_{i'}$ and $V_{j'}$, the observations counted above by (V_j, V_i) and $(V_{j'}, V_{i'})$ are distinct. Since there are at most $n - m_{\min}$ total observations that arrived after $m_{\min}$, we can bound the total number of such pairs by $(n - m_{\min})/(2\epsilon(n - m_{\max}))$ which is easily verified to be at most $3/2\epsilon$. □

Given a full pair of tuples (t_{i-1}, t_i), we say that the tuple t_{i-1} is a *left partner* and t_i is a *right partner* in this full pair.

LEMMA 4. *At any time n and for any given α, there are at most $4/\epsilon$ tuples from $\text{band}_\alpha(n)$ that are right partners in a full tuple pair.*

PROOF. Let X be the set of tuples in $\text{band}_\alpha(n)$ that participate as a right partner in some full pair. We first consider the case when tuples in X form a single contiguous segment in $\mathsf{S}(n)$. Let $t_i, ..., t_{i+p-1}$ be a maximal contiguous segment of $\text{band}_\alpha(n)$ tuples in $\mathsf{S}(n)$. Since these tuples are alive in $\mathsf{S}(n)$, it must be the case that

$$g^*_{j-1} + g_j + \Delta_j > 2\epsilon n \qquad i \leq j < i+p.$$

Adding over all j, we get

$$\sum_{j=i}^{i+p-1} g^*_{j-1} + \sum_{j=i}^{i+p-1} g_j + \sum_{j=i}^{i+p-1} \Delta_j > 2p\epsilon n.$$

In particular, we can conclude that

$$2\sum_{j=i-1}^{i+p-1} g^*_j + \sum_{j=i}^{i+p-1} \Delta_j > 2p\epsilon n.$$

The first term in the LHS of the above inequality counts twice the number of observations covered by nodes in $\text{band}_\alpha(n)$ or by one of its descendants in the tree $\mathsf{T}(n)$. Using Lemma 2, this sum can be bounded by $2(2^\alpha/\epsilon)$. The second term can be bounded by $p(2\epsilon n - 2^{\alpha-1})$ since the largest possible Δ value for a tuple with a band value of α or less is $(2\epsilon n - 2^{\alpha-1})$. Substituting these bounds, we get

$$\frac{2^{\alpha+1}}{\epsilon} + p(2\epsilon n - 2^{\alpha-1}) \quad > \quad 2p\epsilon n$$

Simplifying above, we get $p < 4/\epsilon$ as claimed by the lemma. Finally, the same argument applies when nodes in X induce multiple segments in $\mathsf{S}(n)$; we simply consider the above summation over all such segments. □

LEMMA 5. *At any time n and for any given α, the maximum number of tuples possible from each $\text{band}_\alpha(n)$ is $11/2\epsilon$.*

PROOF. By Lemma 4 we know that the number of $\text{band}_\alpha(n)$ nodes that are right partners in some full pair can be bounded

$N \downarrow$	Our Algorithm (tuple count)					Our Algorithm (space requirement)					MLR Algorithm				
$\epsilon \rightarrow$	.1	.05	.01	.005	.001	.1	.05	.01	.005	.001	.1	.05	.01	.005	.001
10^5:	61	120	496	902	3290	183	360	1488	2706	9870	275	468	1519	2859	8334
10^6:	76	156	664	1230	4983	228	468	1992	3690	14949	378	702	2748	4664	15155
10^7:	94	185	835	1578	6662	282	555	2505	4734	19986	600	1032	3708	7000	27475
10^8:	110	224	1067	2063	9148	330	672	3201	6189	27444	765	1477	5960	10320	37026
10^9:	124	266	1249	2407	11074	372	798	3747	7221	33222	924	1880	7650	14742	59540

Table 1: Number of tuples stored and space requirements for "hard input" sequences. For MRL algorithm, we assume that each quantile stored takes only one unit of space.

by $4/\epsilon$. Any other $\text{band}_\alpha(n)$ node either does not participate in any full pair or occurs only as a left partner. We first claim that each parent of a $\text{band}_\alpha(n)$ node can have at most one such node in $\text{band}_\alpha(n)$. To see this, observe that if a pair of non-full adjacent tuples t_i, t_{i+1}, where $t_{i+1} \in \text{band}_\alpha(n)$, is not merged then it must be because $\text{band}(t_i, n)$ is greater than α. But Proposition 3 tells us that this event can occur only once for any α, and therefore, V_{i+1} must be the unique $\text{band}_\alpha(n)$ child of its parent that does not participate in a full pair. It is also easy to verify that for each parent node, at most one $\text{band}_\alpha(n)$ can participate only as a left partner in a full pair. Finally, observe that only one of the above two events can occur for each parent node. By Lemma 3, there are at most $3/2\epsilon$ parents of such nodes, and thus the total number of $\text{band}_\alpha(n)$ nodes can be bounded by $11/2\epsilon$. □

THEOREM 1. *At any time n, the total number of tuples stored in $\mathsf{S}(n)$ is at most $(11/2\epsilon)\log(2\epsilon n)$.*

PROOF. There are at most $1 + \lfloor \log 2\epsilon n \rfloor$ bands at time n. There can be at most $3/2\epsilon$ total tuples in $\mathsf{S}(n)$ from bands 0 and 1. For the remaining bands, Lemma 5 bounds the maximum number of tuples in each band. The result follows. □

3. EMPIRICAL MEASUREMENTS

We now describe some empirical results concerning the performance of our algorithm in practice. We experimented with three different classes of input data: (1) A "hard case" for our algorithm, (2) "sorted" input data, and (3) "random" input data. The "sorted" and "random" input sequences were chosen for two reasons. First, "random" should yield some insight into the behavior of this algorithm on "average" inputs, or after some randomization. Second, these two scenarios were used to produce the experimental results in [8]. The MRL algorithm [8] is the best previously known algorithm.

We observed during these runs that, in practice, the algorithm used substantially less space than indicated by our analysis from the previous section. The observed space requirements also turn out to be better than those required by the MRL algorithm. Moreover, when we run our algorithm with the same space as used by the MRL algorithm, the observed error is significantly better than that of the MRL algorithm. We will refer to this later variant as the *pre-allocated* variant of our algorithm. In contrast, we will refer to the basic version of the algorithm where we allocate a new quantile entry only when the observed error is about to exceed the desired ϵ, as the *adaptive* variant.

Our implementation of the algorithm differed slightly from that described in Section 2 in two ways. First, new observations were inserted as a tuple $(v, 1, g_i + \Delta_i - 1)$ rather than as $(v, 1, \lfloor 2\epsilon n \rfloor)$. The latter approach is used in the previous section strictly to simplify theoretical analysis of the space complexity. Second, rather than running COMPRESS after every $1/2\epsilon$ observations, instead, for each observation inserted into S, one tuple was deleted, when possible. When no tuple could be deleted without causing its successor to become overfull, the size of S grew by 1. Note that by preallocating a large enough number of stored quantiles, no increase in space need *ever* take place, assuming you know N in advance.

For each experiment we measured both the maximum space used to produce the summary, and the observed precision of the results. We measured space consumption by counting the number of stored tuples. When comparing our space consumption to the MRL algorithm, we pessimistically multiplied the number of stored tuples by 3 to account for our recording the value and both the min and max rank of each stored element.

3.1 Hard Input

We construct here data sequences in adversarial manner for our algorithm. At each time step, we generate the next observation so that it falls in the largest current "gap" in our quantile summary.

We successively fed observations to our summary, with no advance hint about the total number of observations to be seen. We measured the maximum amount of space required as the size of the input sequence increased to 10^9. Table 1 reports the results of this experiment for N ranging over powers of 10 from 10^5 to 10^9.

Note that the required number of quantiles stored is approximately a factor of 11 lower than the worst-case bound we computed in the previous section of this paper. Also note that the number of quantiles we store is significantly lower than the number used by the MRL algorithm. Even after multiplying our tuple count by a factor of 3, we almost always require less space than MRL. The only exception is in $\epsilon = .001$ and $N = 10^5$, where the space cost of our algorithm exceeds that of the MRL algorithm.

3.2 Sorted Input

The second scenario, "sorted", measures the behavior of the summary when the data arrives in sorted order. We fixed $\epsilon = .001$ and constructed summaries of sorted sequences of sizes $10^5, 10^6$, and 10^7. We computed the actual maximum

	Observed ϵ								
$q_i \downarrow$	MRL			Our algorithm, Preallocated			Our algorithm,Adaptive		
$N \rightarrow$	10^5	10^6	10^7	10^5	10^6	10^7	10^5	10^6	10^7
$\|S\|$	8334	15155	27475	2778	5052	9158	756	756	756
Max ϵ	0.00035	0.000194	0.000167	0.00027	0.000128	0.000090	0.00095	0.000899	0.000819
1	0.00015	0.000199	0.000091	0.00021	0.000020	0.000077	0.00074	0.000057	0.000618
2	0.00006	0.000050	0.000120	0.00024	0.000056	0.000009	0.00039	0.000259	0.000203
3	0.00006	0.000210	0.000062	0.00010	0.000052	0.000031	0.00010	0.000744	0.000665
4	0.00024	0.000161	0.000001	0.00001	0.000016	0.000005	0.00040	0.000860	0.000002
5	0.00002	0.000033	0.000070	0.00002	0.000092	0.000050	0.00016	0.000494	0.000230
6	0.00022	0.000166	0.000053	0.00012	0.000048	0.000014	0.00027	0.000716	0.000632
7	0.00000	0.000037	0.000085	0.00024	0.000060	0.000066	0.00007	0.000388	0.000488
8	0.00010	0.000084	0.000043	0.00012	0.000096	0.000035	0.00021	0.000829	0.000090
9	0.00019	0.000207	0.000095	0.00006	0.000124	0.000014	0.00033	0.000000	0.000038
10	0.00013	0.000060	0.000100	0.00012	0.000088	0.000050	0.00055	0.000036	0.000354
11	0.00005	0.000098	0.000013	0.00002	0.000000	0.000014	0.00005	0.000542	0.000185
12	0.00004	0.000096	0.000001	0.00008	0.000004	0.000022	0.00017	0.000093	0.000010
13	0.00006	0.000107	0.000045	0.00014	0.000008	0.000044	0.00039	0.000263	0.000220
14	0.00002	0.000116	0.000038	0.00020	0.000008	0.000056	0.00022	0.000732	0.000665
15	0.00003	0.000098	0.000049	0.00023	0.000028	0.000041	0.00008	0.000316	0.000425

Table 2: Space and precision measurements for "sorted" case.

error over *all* possible quantile queries, and chose to query 15 quantiles at rank $\frac{q_i}{16}N$, for $q_i = [1..15]$, to study the behavior at specific quantiles.

We compared three algorithms for constructing the summary. First, we used the MRL algorithm to compute a summary where we preallocated the storage required by MRL as a function of N and ϵ. Second, we pre-allocated the same amount of storage required by MRL (1/3 as many stored quantiles as MRL, though), and ran our algorithm without allocating any more quantiles. Finally, we ran our algorithm in the adaptive mode; we started with $\frac{1}{2\epsilon}$ stored quantiles and only allocated extra storage if it was impossible to delete existing quantiles without exceeding a precision of $.001n$.

Table 2 reports the results of this experiment. $|S|$ reports the number of stored quantiles needed to achieve the desired precision. The row labeled "max" reports the maximum error of *all* possible quantile queries on the summary. In order to give an indication of the behavior of this algorithm for specific quantiles, the remaining rows list the approximation error of the response to the query for the $q_i/16$th quantile.

To interpret the entries in Table 2, consider the .5 quantile (50th pctile, or 8/16). For a sequence of 10^5 elements, the adaptive algorithm uses only 756 tuples, but returns a value with an approximation error of .00021. MRL stores over eight times as many quantiles, and returns a value with error .00010, almost twice as accurate. Our preallocated algorithm stores only one third as many tuples as MRL, but returns a value with an approximation error of .00012 – comparable accuracy but using only one third the number of tuples.

In fact, however, the error on any individual quantile is not representative of the error as a whole — had we chosen to inspect the 1/4 quantile instead of 1/2, then our algorithm would have been 24 times as accurate as MRL! Had we chosen 3/4, then MRL would have been twice as accurate as ours. Of the 15 quantiles we sampled, we outperformed MRL on 6 out of 15 for a sequence of size 10^5, 10 out of 15 for size 10^6, and 11 out of 15 for 10^7. Individual queries are highly sensitive to how close the quantile query happens to be to some single stored quantile. On average, in comparison to MRL using the same storage, our algorithm reported better worst-case observed error, and comparable observed error (we perform slightly worse for $N = 10^5$, but slightly better for $N = 10^6$ and 10^7). Both algorithms achieved higher precision than demanded by the a priori specification.

The most interesting result is that our adaptive algorithm seems to require only 756 stored quantiles, regardless of the size of the input sequence. Closer experimentation revealed that the algorithm only needs all 756 stored quantiles at a fairly early stage in the computation — the excess storage reduces the observed error, slightly. One can see this by observing the maximum error in Table 2. For a desired $\epsilon = .001$, one would expect that the maximum observed error would be approximately equal to .001, too. However, for 10^5 the maximum error is only .000955 and as N gets larger the maximum error gets smaller.

The maximum error offers another interesting insight into the behavior of our algorithm. Note that the optimal value for maximum error in all cases is $1/(2|S|)$ (this occurs only if the stored quantiles are distributed evenly among all values, and we know their rank precisely). For example, for 756 quantiles, the optimal max error is .00066. For 2778 quantiles, the ideal maximum error is .00018. Our algorithm delivers a maximum error within a factor of 2 of optimal. In contrast, the optimal max error of 8334 stored quantiles is 5.99×10^-5, yet the MRL algorithm delivers a max error 6 times as large. In fact, for MRL, the discrepancy between the ideal max error and observed max error seems to grow as N (and $|S|$) gets larger; for $N = 10^7$, the observed max error is more than 9 times the optimal value.

3.3 Random Input

The third scenario, "random", selects each datum by selecting an element (without replacement) from a uniform distribution of all the remaining elements in the data set.

That is, the values in the data set can have an arbitrarily skewed distribution, but the *order* in which the values are observed by the summary is chosen by the uniform random process.

As in the sorted case, we fixed $\epsilon = .001$ and summarized sequences of lengths 10^5, 10^6, and 10^7. We again computed the maximum error, the quantiles at rank $\frac{q_i}{16}N$, for $q_i = [1..15]$, and measured the actual maximum storage requirement to compute the summary. In contrast to the sorted input case where a single experiment was sufficient to determine the expected behavior, random input requires running several trials to illuminate expected behavior. We ran each experiment 50 times and report the min, max, mean and standard deviation for every measurement. Tables 3 through 5 report these results.

The observed ϵ of our preallocated algorithm is roughly twice as accurate as MRL, although our advantage seems to increase steadily as N gets larger. Not surprisingly, the observed ϵ of our adaptive algorithm stays close to 0.001 regardless of how large N gets. The observed storage requirements, however, may be surprising. These are once again the most interesting results of our "random" scenario. It appears that for uniformly random input the required space is independent of N, the size of the dataset, and dependent only upon ϵ. In all our experiments, a .001-approximate summary of a random input was achieved with roughly 920 tuples.

4. CONCLUDING REMARKS

We presented a new online algorithm for computing quantile summaries of very large sequences of data in a space-efficient manner. Our algorithm improves upon the earlier results in two significant ways. First, it improves the space complexity by a factor of $\Omega(\log(\epsilon N))$. Second, it does not require a priori knowledge of the parameter N — that is, it allocates more space dynamically as the data sequence grows in size. An obvious question is whether or not the space complexity achieved by our algorithm is asymptotically optimal. We believe that the answer is in the affirmative indeed.

Our empirical study of the new algorithm provides evidence that our algorithm compares favorably with the previous algorithms in practice as well. A curious trend observed in our experiments is that on random inputs, the space requirements of the algorithm seem only to depend on the error parameter ϵ and become independent of the sequence length N. It will be interesting to analytically verify this behavior and to understand the minimal characteristics of the data sequences that lead to such improved space requirements.

5. REFERENCES

[1] Rakesh Agrawal and Arun Swami. A one-pass space-efficient algorithm for finding quantiles. *Proc. 7th Int. Conf. Management of Data, COMAD*, 28–30 December 1995.

[2] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. A one-pass algorithm for accurately estimating quantiles for disk-resident data. *Proceedings of the 23rd Intl. Conference on Very Large Data Bases, Athens, Greece, 26–29 August 1997*, pages 346–355, Los Altos, CA 94022, USA, 1997. Morgan Kaufmann Publishers.

[3] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Random sampling for histogram construction: how much is enough? In *ACM SIGMOD '98*, volume 28, pages 436–447, Seattle, WA, June 1–4, 1998.

[4] Phillip B. Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms. In *Proceedings of the 23rd Intl. Conf. Very Large Data Bases, VLDB*, pages 466–475. Morgan Kaufmann, 25–27 August 1997.

[5] Michael B. Greenwald. Practical algorithms for self scaling histograms or better than average data collection. *Performance Evaluation*, 27&28:19–40, October 1996.

[6] R. Jain and I. Chlamtac. The P^2 algorithm for dynamic calculation of quantile and histograms without storing observations. *Communications of the ACM*, 28(10):1076–1085, October 1986.

[7] I. Pohl. A minimum storage algorithm for computing the median. *IBM Research Report* RC 2701, November 1969.

[8] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD '98*, volume 28, pages 426–435, Seattle, WA, June 1998.

[9] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *ACM SIGMOD '99*, volume 29, pages 251–262. Philadelphia, PA, June 1999.

[10] J. I. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, vol. 12: 315–323; 1980.

[11] M.S. Paterson. Progress in selection. *Technical Report*, University of Warwick, Coventry, UK, 1997.

[12] Viswanath Poosala, Venkatesh Ganti, and Yannis E. Ioannidis. Approximate query answering using histograms. *Bulletin of the IEEE Technical Committee on Data Engineering*, 22(4):6–15, December 1999.

[13] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. Improved histograms for selectivity estimation of range predicates. In *ACM SIGMOD 96*, volume 26, pages 294–305, Montreal, Quebec, Canada, June 4–6, 1996.

$q_i \downarrow$	MRL		Our Algorithm, Preallocated		Our Algorithm, Adaptive	
$\lvert S\rvert \rightarrow$	8334		2778		[898-939], 919.18±8.63	
	[range ($\times 10^{-4}$)]	avg±stdev	[range ($\times 10^{-4}$)]	avg±stdev	[range ($\times 10^{-4}$)]	avg±stdev
Max ϵ	[4.3-5.2]	0.0004698±2.02e-05	[2.9-2.95]	0.0002920±0.24e-05	[8.25-8.70]	0.0008487±0.91e-05
1	[0.0-3.2]	0.0000928±7.38e-05	[0.1-2.5]	0.0001074±7.19e-05	[0.1-7.8]	0.0003222±1.88e-04
2	[0.0-3.0]	0.0001130±7.58e-05	[0.2-2.5]	0.0001216±6.42e-05	[0.1-7.0]	0.0003216±1.88e-04
3	[0.0-3.5]	0.0001104±8.86e-05	[0.0-2.7]	0.0001220±7.36e-05	[0.2-7.7]	0.0003406±2.07e-04
4	[0.0-2.8]	0.0001040±6.93e-05	[0.0-2.7]	0.0001236±7.44e-05	[0.1-7.6]	0.0002952±1.98e-04
5	[0.0-3.7]	0.0001172±8.81e-05	[0.0-2.6]	0.0000844±6.07e-05	[0.1-6.6]	0.0003102±1.88e-04
6	[0.1-3.0]	0.0001046±7.69e-05	[0.0-3.3]	0.0000912±7.41e-05	[0.2-6.7]	0.0002986±1.64e-04
7	[0.2-3.6]	0.0001346±7.97e-05	[0.0-2.5]	0.0001078±6.45e-05	[0.0-6.9]	0.0003090±1.89e-04
8	[0.1-3.8]	0.0000982±8.86e-05	[0.0-3.1]	0.0001134±7.08e-05	[0.0-7.7]	0.0002910±1.94e-04
9	[0.0-2.7]	0.0001222±7.37e-05	[0.0-2.5]	0.0001074±7.62e-05	[0.0-6.6]	0.0002910±1.75e-04
10	[0.0-3.4]	0.0001278±7.68e-05	[0.0-2.3]	0.0000912±6.01e-05	[0.0-7.0]	0.0002740±1.69e-04
11	[0.1-3.1]	0.0001204±7.87e-05	[0.0-2.8]	0.0000954±7.31e-05	[0.1-6.9]	0.0002790±1.84e-04
12	[0.1-2.4]	0.0001040±6.83e-05	[0.0-2.4]	0.0000940±6.71e-05	[0.2-8.2]	0.0003566±2.32e-04
13	[0.0-3.0]	0.0000878±6.83e-05	[0.0-2.3]	0.0001114±6.49e-05	[0.2-7.6]	0.0003446±2.01e-04
14	[0.0-3.1]	0.0000982±8.05e-05	[0.0-2.5]	0.0001196±6.80e-05	[0.4-8.2]	0.0003424±1.99e-04
15	[0.0-2.8]	0.0001000±7.12e-05	[0.0-2.8]	0.0001330±8.24e-05	[0.1-6.2]	0.0002952±1.86e-04

Table 3: $N = 100,000$**; Samples = 50; random order.**

$q_i \downarrow$	MRL		Our Algorithm, Preallocated		Our Algorithm, Adaptive	
$\lvert S\rvert \rightarrow$	15155		5052		[900-939] 919.38±8.92	
	[range ($\times 10^{-4}$)]	avg±stdev	[range ($\times 10^{-4}$)]	avg±stdev	[range ($\times 10^{-4}$)]	avg±stdev
Max ϵ	[3.02-3.63]	0.0003275±1.44e-05	[1.495-1.520]	15.04e-05±0.06e-05	[7.835-8.215]	0.0008004±0.82e-05
1	[0.02-3.00]	0.0001194±7.88e-05	[0.05-1.41]	5.41e-05±3.37e-05	[0.00-7.78]	0.0003173±2.12e-04
2	[0.09-3.19]	0.0001248±7.69e-05	[0.04-1.41]	5.79e-05±3.65e-05	[0.06-6.94]	0.0003259±1.80e-04
3	[0.01-2.90]	0.0001253±7.27e-05	[0.01-1.28]	5.73e-05±3.71e-05	[0.15-7.11]	0.0003172±1.87e-04
4	[0.01-2.71]	0.0001092±7.47e-05	[0.02-1.43]	5.57e-05±3.46e-05	[0.07-7.04]	0.0003546±1.97e-04
5	[0.12-2.84]	0.0001260±7.44e-05	[0.03-1.36]	5.45e-05±3.59e-05	[0.02-7.06]	0.0002907±1.78e-04
6	[0.01-3.20]	0.0000984±7.68e-05	[0.01-1.22]	5.89e-05±3.26e-05	[0.29-6.57]	0.0002972±1.76e-04
7	[0.01-2.79]	0.0001256±7.52e-05	[0.01-1.38]	5.03e-05±3.58e-05	[0.09-6.30]	0.0002951±1.60e-04
8	[0.05-3.27]	0.0001299±6.03e-05	[0.01-1.21]	4.55e-05±3.37e-05	[0.11-7.10]	0.0002892±1.73e-04
9	[0.22-3.27]	0.0001268±7.75e-05	[0.05-1.24]	5.88e-05±3.57e-05	[0.04-7.15]	0.0003015±2.04e-04
10	[0.13-3.74]	0.0001389±8.64e-05	[0.03-1.61]	7.14e-05±3.88e-05	[0.02-7.07]	0.0002924±2.04e-04
11	[0.09-3.01]	0.0001431±7.67e-05	[0.00-1.38]	5.81e-05±3.58e-05	[0.11-6.43]	0.0002989±2.01e-04
12	[0.03-3.32]	0.0001446±8.64e-05	[0.00-1.46]	4.86e-05±3.33e-05	[0.20-6.71]	0.0003378±1.66e-04
13	[0.04-2.84]	0.0001339±7.25e-05	[0.00-1.34]	5.30e-05±3.42e-05	[0.04-6.69]	0.0003128±1.70e-04
14	[0.04-2.74]	0.0001288±8.91e-05	[0.03-1.43]	5.65e-05±3.60e-05	[0.02-7.03]	0.0003146±1.86e-04
15	[0.02-2.92]	0.0001284±8.82e-05	[0.02-1.67]	5.45e-05±3.86e-05	[0.05-6.46]	0.0002797±1.72e-04

Table 4: $N = 1,000,000$**; Samples = 50; random order.**

$q_i \downarrow$	MRL		Our Algorithm, Preallocated		Our Algorithm, Adaptive	
$\lvert S\rvert \rightarrow$	27475		9158		[899-939] 918.42±8.71	
	[range ($\times 10^{-4}$)]	avg±stdev	[range ($\times 10^{-4}$)]	avg±stdev	[range ($\times 10^{-4}$)]	avg±stdev
Max ϵ	[2.032-2.641]	2.35e-04±1.18e-05	[0.799-0.806]	8.01e-05±1.8e-07	[7.628-8.016]	7.82e-04±9.75e-06
1	[0.026-1.466]	4.98e-05±3.29e-05	[0.002-0.712]	2.74e-05±1.96e-05	[0.187-6.123]	2.87e-04±1.65e-04
2	[0.022-1.922]	6.32e-05±4.98e-05	[0.001-0.764]	2.94e-05±2.22e-05	[0.166-6.814]	3.04e-04±1.80e-04
3	[0.019-1.750]	5.90e-05±4.62e-05	[0.002-0.656]	2.93e-05±1.80e-05	[0.008-7.040]	3.68e-04±1.91e-04
4	[0.024-1.953]	6.19e-05±4.37e-05	[0.003-0.615]	2.98e-05±1.65e-05	[0.096-7.149]	2.98e-04±1.81e-04
5	[0.022-1.892]	7.02e-05±5.03e-05	[0.011-0.722]	2.99e-05±1.63e-05	[0.111-7.297]	2.56e-04±1.80e-04
6	[0.026-1.766]	6.61e-05±4.65e-05	[0.008-0.655]	2.60e-05±1.86e-05	[0.021-6.618]	3.27e-04±1.72e-04
7	[0.038-1.987]	5.75e-05±4.33e-05	[0.025-0.688]	3.30e-05±1.63e-05	[0.009-5.620]	2.14e-04±1.47e-04
8	[0.004-1.801]	5.69e-05±4.29e-05	[0.006-0.712]	2.69e-05±2.01e-05	[0.043-7.718]	3.17e-04±1.96e-04
9	[0.012-2.252]	6.47e-05±4.19e-05	[0.003-0.675]	2.90e-05±1.83e-05	[0.116-7.167]	2.83e-04±1.93e-04
10	[0.011-1.840]	6.11e-05±4.28e-05	[0.006-0.649]	2.64e-05±1.67e-05	[0.050-7.225]	3.09e-04±1.83e-04
11	[0.010-1.640]	6.67e-05±4.41e-05	[0.005-0.727]	2.99e-05±1.78e-05	[0.231-6.606]	2.60e-04±1.66e-04
12	[0.013-1.847]	6.09e-05±4.69e-05	[0.013-0.686]	2.68e-05±1.71e-05	[0.018-6.639]	2.95e-04±1.51e-04
13	[0.005-1.747]	5.80e-05±3.87e-05	[0.015-0.680]	2.82e-05±1.93e-05	[0.014-6.518]	3.06e-04±1.90e-04
14	[0.026-1.853]	7.12e-05±5.07e-05	[0.000-0.671]	3.43e-05±1.84e-05	[0.051-7.385]	2.69e-04±1.99e-04
15	[0.022-1.510]	5.57e-05±3.56e-05	[0.019-0.775]	2.91e-05±1.83e-05	[0.029-6.415]	2.74e-04±1.80e-04

Table 5: $N = 10,000,000$**; Samples = 50; random order.**