Secure Merge in Linear Time and $O(\log \log N)$ Rounds [*†]

Mark Blunk, Paul Bunn, Samuel Dittmer, Steve Lu, Rafail Ostrovsky

Abstract

Secure merge considers the problem of combining two sorted lists (which are either held separately by two parties, or held by two parties in some privacy-preserving manner, e.g. via secret-sharing), and outputting a single merged (sorted) list in a privacy-preserving manner (typically the final list is encrypted or secret-shared amongst the original two parties). Just as algorithms for *insecure* merge are faster than comparison-based sorting ($\Theta(n)$ versus $\Theta(n \log n)$ for lists of size n), we explore protocols for performing a *secure* merge that are more performant than simply invoking a secure sort protocol. Namely, we construct a semi-honest protocol that requires $O(n)$ communication and computation and $O(\log \log n)$ rounds of communication. This matches the metrics of the insecure merge for communication and computation, although it does not match the $O(1)$ round-complexity of insecure merge. Our protocol relies only on black-box use of basic secure primitives, like secure comparison and shuffle.

Our protocol improves on previous work of [FNO22], which gave a $O(n)$ communication and $O(n)$ round complexity protocol, and other "naive" approaches, such as the shuffle-sort paradigm, which has $O(n \log n)$ communication and $O(\log n)$ round complexity. It is also more efficient for most practical applications than either a garbled circuit or fully-homomorphic encryption (FHE) approach, which each require $O(n \log n)$ communication or computation and have $O(1)$ round complexity.

There are several applications that stand to benefit from our result, including secure sort (in cases where two or more parties have access to their own list of data, secure sort reduces to secure merge since the parties can first sort their own data locally), which in-turn has implications for more efficient private set intersection (PSI) protocols; as well as secure mutable database storage and search, whereby secure merge can be used to insert new rows into an existing database.

In building our secure merge protocol, we develop several subprotocols that may be of independent interest. For example, we develop a protocol for secure *asymmetric* merge (where one list is much larger than the other), which matches theoretic lower-bounds for all three metrics (assuming the ratio of list sizes is small enough).

Keywords. Secure Merge, Secure Sort, Secure Database, Private Set Intersection.

[*]This work was performed under the following financial assistance award number 70NANB21H064 from U.S. Department of Commerce, National Institute of Standards and Technology. The statements, findings, conclusions, and recommendations are those of the author(s) and do not necessarily reflect the views of the National Institute of Standards and Technology or the U.S. Department of Commerce.

[†]Submitted to a conference in April 2022

1 Introduction

It is well-known that sorting a list of length n using only comparisons requires $\Theta(n \log n)$ time, while merging two already sorted lists requires only $\Theta(n)$ time, see e.g., [MSS08, Ski98]. Namely, by taking advantage of the structure of the input (the ordering on the two lists), merge algorithms can out-perform their "*input-agnostic*" counterparts (sorting algorithms) by a factor of $\log n$ in run-time. This reflects and reinforces a general intuition that it is more efficient to maintain and update an existing structure than it is to tear down the structure and rebuild it from the ground up. In the area of database management, this general intuition (applied here to the specific example of merging versus sorting) applies directly to tables that need to be kept in a sorted order (e.g., to efficiently query order statistics). It is far more efficient to maintain a table in a sorted order by periodically merging in new data entries than it is to re-sort the entire table for each update cycle. Another frequently encountered scenario when a sorted list is desired occurs when data values are held by two or more parties who wish to produce a sorted, merged copy of their data. Each party can separately invoke an (insecure) sort algorithm locally on their own data, so that (the more costly) *secure* operations are required only for merging the lists. For example, many private set intersection (PSI) protocols (see e.g., [CHI+19, PSSZ15, HEK12]) are bottlenecked by an underlying invocation of secure sort, and consequently (in appropriate settings) could benefit by a factor of $\log n$ in performance by using secure merge instead.

Motivated by the above intuition, we investigate in this paper whether the theorized $\log n$ run-time improvement for merge versus sort can be realized in the *secure* setting as well. We answer this question in the affirmative, producing a protocol that instantiates the following:

Theorem 1.1 (*Informal*)**.** *There exists a two-party secure merge protocol with $\Theta(n)$ run-time (computation and communication) and $\Theta(\log \log n)$ rounds that relies only on black-box access to the secure functionalities defined in §3.7: comparison, shuffle, open, and conditional-addition.*

1.1 Paper Structure

In Section 2 we summarize previous work targeting the secure sort/merge problem, and provide a comparison table of our main result with relevant earlier works in Table 1. In Section 3, we set notation and provide a high-level overview of the techniques and tools our protocols build on. Then in Section 4, we present our main secure merge protocol, as well as the sub-protocols that it builds upon. We give formal statements and proofs in Section 5.1, and describe our constant round secure merge in the asymmetric case where one list has size n^α, for fixed $\alpha < 1$, in Section 6.

Supplementary material (instantiations of secure primitives, full proofs, and expanded discussion) can be found in Appendices A-D.

2 Previous Work

As the merging/sorting of lists is a fundamental problem in computer science, there has been enormous research in this area. Not surprisingly, many of the protocols for *securely* merging/sorting lists draw inspiration from these *insecure* counterparts (not to mention the generic approach of converting insecure protocols to secure protocols e.g. via garbled circuit or fully-homomorphic encryption). We discuss below[1] previous work for both the insecure and secure variants, and for the sake of comparison summarize the most relevant results to our main result in Table 1 below.

[1] As mentioned, the work in this area is extensive, so our summary of results is necessarily non-exhaustive. Please see cited works, and references therein, for a more complete overview and discussion.

2.1 Insecure Merge Algorithms

2.1.1 Sorting Networks

The relationship between secure merging and secure sorting can be traced back to [Bat68], which built a sorting network of size $O(n \log^2 n)$ from $\log n$ merging networks, each of size $O(n \log n)$. Although an asymptotically optimal sorting network of size $O(n \log n)$ was later achieved by Ajtai, Komlos, and Szemeredi [AKS83], merging networks cannot match the combinatorially optimal $O(n)$ size. A merging network on lists of size $\Theta(n)$ require $\Omega(n \log n)$ comparators, as shown by Yao and Yao [YY76], and depth of $\Omega(\log n)$, as shown by Hong and Sedgewick [HS82].

2.1.2 RAM and PRAM

The classical merge algorithm requires $O(n)$ work on a RAM machine, see e.g., [MSS08]. As discussed below, the approach of [FNO22] uses an approach that is inspired by this classical merge algorithm, and achieves matching asymptotic work (albeit with $O(n)$ round-complexity).

A parallel RAM machine (PRAM) allows multiple processors to act on the same set of memory in parallel. We are in particular interested in Concurrent-Read-Exclusive-Write (CREW) PRAM, where all processors can read the same memory simultaneously, but processors cannot write to the same memory address at the same time, see e.g., [BH85] for more formal definitions and a discussion of various PRAM models.

For PRAM machines, Valiant showed in [Val75] that $O(n)$ processors could merge two lists of size $O(n)$ in time $O(\log \log n)$. This was improved to $O(n/\log \log n)$ processors by Borodin and Hopcroft [BH85], who also showed that the $O(\log \log n)$ time bound is optimal when limited to $O(n)$ processors. As a rough heuristic, we expect the number of processors and total work done by a PRAM algorithm to serve as a lower bound for the round complexity and communication complexity of a corresponding secure protocol, motivating the following:

> **Conjecture**. *Any linear-time protocol for two parties to securely merge their lists (each of size $\Theta(n)$) requires $\Theta(\log \log n)$ rounds of communication.*

Notice that if the above conjecture is valid, then our secure merge protocol (Section 4) is asymptotically optimal in all three metrics: computation, communication, and round complexity.

More formally, if a 2-party protocol Π securely realizes some functionality $\mathcal{F}$ in R rounds and C communication, and each party can execute each round of the protocol in $O(1)$ time on a CREW PRAM with C processors, then there exists an algorithm A that realizes $\mathcal{F}$ on a single CREW PRAM machine (playing the role of all parties) with C processors, $O(RC)$ total work, and $O(R)$ time. This observation immediately implies that the conjecture is true for secure merge protocols with the property that each round of the protocol can be executed in $O(1)$ time on a CREW PRAM with $O(n)$ processors.

Both the Valiant and the Borodin-Hopcroft algorithms rely on the following basic construction: Split each list into blocks of size $\sqrt{n}$, with $\sqrt{n}$ pivots. Running all pairwise comparisons between pivots identifies which block of the opposite list each pivot is mapped to; then another round of pairwise comparisons identifies the exact position the pivot is mapped to within that block. This partitions the merge problem into $\sqrt{n}$ subproblems of size approximately $\sqrt{n}$, giving the recurrence $c(n) = \sqrt{n} \cdot c(\sqrt{n})$, if $c(n)$ is the cost of merging two lists of length n. With sufficiently many processors, this recurrence gives a run-time of $O(\log \log n)$. Our secure merge protocols (Section 4) are similar in spirit, although some adaptation is required to deal with blocks from one list that cover more than one block of the other list.

A follow-up work by Hayashi, Nakano, and Olariu [HNO98] demonstrates a protocol with $O(\log \log n + \log k)$ run-time for merging k lists of size $O(n)$. Following the intuition that run-time in the PRAM model roughly corresponds to (a lower-bound on) round-complexity in the secure setting,

this suggests that a linear-time secure merge protocol for k parties could have round complexity $O(\log k)$. For example, this would imply that an optimal secure merge protocol for $k = O(\log n)$ parties (each with a list of size $O(n)$) could have linear run-time and $O(\log \log n)$ round complexity. However, because our secure merge protocol (Section 4) relies on black-box use of a constant-round secure shuffle subprotocol with $O(n)$ communication, and since currently known instantiations of secure shuffle with linear communication have *linear* or worse dependence on the number of parties (in terms of round-complexity) [FO21, LWZ11], direct extension of our secure merge protocol to $k > 2$ parties would have $O(\log \log n + k)$ round complexity.

2.2 Secure Merge Algorithms

2.2.1 Security via Generic Transformation

We explore here two naïve solutions for transforming an insecure merge algorithm to a secure one (see Table 1 for a succinct comparison of our secure merge protocol to these naïve solutions):

Garbled Circuits.

By choosing any (insecure) sorting algorithm that can be represented as a circuit, the parties can use garbled circuits to (securely) sort their list in $O(1)$ rounds. As discussed above in §2.1.1, for comparison-based merging networks, $\Omega(n \log n)$ comparisons and $\Omega(\log n)$ depth are necessary, and achievable by the Batcher merging network [Bat68].

Additionally, we note that obtaining κ bits of computational security when merging lists whose elements have size W bits requires $\kappa \cdot W$ bits, or κ words of communication for each comparison. Thus, obtaining secure merge via a garbled circuit approach (for a circuit representing a merging network) would result in a constant-round protocol with $\Omega(\kappa \cdot n \log n)$ communication.

On the other hand, using GMW-style circuit evaluation [GMW87] instead of garbled circuits can reduce the overhead of each comparison (from κ down to constant), but incurs a hit in round complexity (proportional to the depth of the circuit instead of constant-round).

Fully Homomorphic Encryption. We can also construct a $O(1)$ round protocol by having one party encrypt their inputs under fully homomorphic encryption (FHE) and send the result to the other party, who performs the desired calculations on ciphertexts. The other party then subtracts a vector of random values $\mathbf{r}$ from the (encrypted) merged list and sends the result back to the first party, who decrypts to obtain the merged list shifted by $\mathbf{r}$. Now the parties hold an additive sharing of the sorted list.

As with garbled circuits, however, the calculations the second party performs on the ciphertexts must be input-independent (in order to avoid information leakage), and so the calculation must be represented by a circuit. This means that communication is (asymptotically) lower than the garbled circuit approach, since the first party need only provide ciphertexts of his list (which correspond to *inputs* to the circuit), as opposed to providing information for each *gate* of the circuit. Therefore the communication required for the FHE approach is only $O(\beta^* n)$, where β^* is the ciphertext expansion for the FHE scheme, which approaches 1 as the word size W grows. However, while communication in the FHE approach may be (asymptotically) reduced (compared to the garbled circuit approach), notice that the computation is still $O(\gamma^* n \log n)$, where γ^* is the cost of a multiplication under FHE, as the circuit requires $O(n \log n)$ comparison (multiplication) operations. Additionally, since the circuit has depth $\Omega(\log n)$, the FHE scheme will require bootstrapping to avoid ciphertext blowup, which will be expensive in practice.

2.2.2 Shuffle-Sort Paradigm

One challenge facing any *secure* merge (or sort) protocol is that the results of each comparison must be kept secret from each party, or else security is lost. One approach to allowing the results of the comparisons to be known *without* information leakage is to first *shuffle* (in an oblivious manner) the

input lists. However, since this shuffling inherently destroys the property that the two input lists are already sorted, this approach reduces a secure *merge* problem to a secure *sort* problem, hence incurring the $\log n$ efficiency loss.

Comparison-based sorting requires $O(n \log n)$ communication and computation as well as $O(\log n)$ rounds, while secure shuffling requires $O(n)$ communication and $O(1)$ rounds, see e.g., [HKI+12, FO21]. Therefore a secure sort using the shuffle-sort paradigm requires $O(n \log n)$ communication and computation and $O(\log n)$ rounds, which is worse than our secure merge approach on all three metrics; see Table 1.

2.2.3 Oblivious Sort

Because the AKS-constant is too large for practical applications, a number of other approaches to secure sorting have been explored. The shuffle-sort paradigm mentioned above is one example of a large family of *oblivious* sort protocols, which allow for a variable memory access pattern as long as it is independent of the underlying list values, or *data oblivious*. We mention here the radix sort of Hamada, Ikarashi, Chida, and Takahashi [HICT14] achieves $O((W \log W + W)n + n \log n)$ communication (in memory words) and $O(1)$ round complexity in the 3-party honest majority setting with constant bit lengths of elements, later improved by Chida, Hamada, Ikarashi, Kikuchi, Kiribuchi, and Pinkas [CHI+19], to $O(n \log n)$ memory words. However, the round complexity depends linearly on W, so when $W \approx \log n$, this matches the round complexity of the other protocols.

2.2.4 Secure Merge Protocols

There are several works that investigate secure merge directly. The first, due to Falk and Ostrovsky [FO21], achieves $O(n \log \log n)$ communication complexity with $O(\log n)$ round complexity. The second, due to Falk, Nema, Ostrovsky [FNO22], achieves the asymptotically optimal $O(n)$ communication complexity (and with small constants), but requires $O(n)$ rounds of communication. For many cryptographic applications, a high round complexity causes more of a bottleneck than a high communication complexity. Therefore, the secure merge protocol of [FNO22], while both simple and asymptotically optimal in terms of communication, may still not be practical for many applications.

A comparison of these results to our secure merge protocol can be found in Table 1 below.

2.3 Comparison of Results

The black box functionalities introduced in our main theorem can be efficiently implemented using additive secret sharing, GMW, and an additively homomorphic rerandomizable public key cryptosystem such as Paillier or lattice-based LWE, as discussed in §3.7.

To make the comparison with our protocols as concrete as possible, we introduce the following constants: κ is a computational security parameter for use in Yao's garbled circuits. β is the ciphertext expansion in our chosen additively homomorphic cryptosystem, while γ is the cost of decryption in that cryptosystem.

Asymptotically $\kappa \gg \log n$ is necessary so that κ cannot be guessed in the time it takes to traverse the list. In practice $\kappa = 128$ is standard. Asymptotically, β approaches 1 as the word size W grows. In practice for LWE cryptosystems β can be quite reasonable, e.g. $\beta = 17.7$ for shuffling 16384 29-bit plaintexts packed into a single ciphertext in the PALISADE FHE library [FO21].

Because running secure merge as a comparison-based circuit requires $\Omega(\log n)$ depth, the ciphertext expansion and computational complexity of performing the sorting over FHE will take a hit compared to the depth one public key operations required for 2-party sort. Rather than attempting to estimate this gap, we use (β^*, γ^*) to represent the same quantities over some FHE scheme.

We assume the objects to be sorted are contained in $O(1)$ memory words of size W bits. Unless explicitly stated otherwise, our communication and computational complexity numbers are given in

terms of memory words and primitive operations on memory words.

Protocol	Computation	Communication	Rounds
(Garbled) Merging Network [Folklore]	$O(\kappa \cdot n \log n)$	$O(\kappa \cdot n \log n)$	$O(1)$
(GMW) Merging Network [Folklore]	$O(n \log n)$	$O(n \log n)$	$O(\log n)$
(FHE) Merging Network [Folklore]	$O(\gamma^* n \log n)$	$O(\beta^* n)$	$O(1)$
Shuffle-Sort [HKI+12]	$O(n \log n)$	$O(n \log n)$	$O(\log n)$
Oblivious Radix Sort [CHI+19]	$O(n \log n)$	$O(n \log n)$	$O(W)$
Secure Merge of [FO21]	$O(n \log \log n + \gamma n)$	$O(n \log \log n + \beta n)$	$O(\log n)$
Secure Merge of [FNO22]	$O(\gamma n)$	$O(\beta n)$	$O(n)$
Our Secure $(\mathbf{n}^\alpha, \mathbf{n})$ Merge Protocol	$O(2^{1/(1-\alpha)^3}\gamma n)$	$O(2^{1/(1-\alpha)^3}\beta n)$	$O(1/(1-\alpha)^3)$
Our Secure Merge Protocol	$O(\gamma n)$	$O(\beta n)$	$O(\log \log n)$

Table 1: Comparison of our secure merge protocol with other existing solutions, where n is the list size, W is the size of a memory word, list elements fit into $O(1)$ memory words, κ is a computational security parameter for Yao's garbled circuits, and β and γ are the ciphertext expansion and computational cost of a decryption operation, respectively, over some additively homomorphic cryptosystem, while (β^*, γ^*) represent the same costs over FHE. In our (n^α, n) merge protocol, $\alpha < 1$ is some fixed constant.

3 Overview of Techniques

3.1 Notation

For two lists $A = (a_1, \ldots, a_k)$ and $B = (b_1, \ldots, b_k)$, we denote the *zip* of A and B as $A \bowtie B = ((a_1, b_1), \ldots, (a_k, b_k))$. For any two sorted lists L_1 and L_2, let $\bigsqcup$ denote the "merge" of two lists (i.e. $\bigsqcup$ is functionally equivalent to (multi-)set union followed by sort): $L_1 \bigsqcup L_2 = Sort(L_1 \cup L_2)$. For any sorted list L_j of size n, and for any $k|n$, let $\mathcal{M}_{j,k}$ denote the k "medians" of L_j. Namely, if list $L_j = \{u_1, \ldots, u_n\}$, then:

$$\mathcal{M}_{j,k} := \{u_{j \cdot \frac{n}{k}}\}_{j=1}^{k} \tag{1}$$

Basic properties that follow from the definition of medians can be found in Appendix D.

3.2 Symmetric vs. Asymmetric Merge

Throughout the paper we distinguish between secure symmetric merges $\mathbf{\Pi_{SSM}}$ and secure asymmetric merges $\mathbf{\Pi_{SAM}}$. In a symmetric merge, the lists are of roughly the same size, or differ by at most a constant factor (since then one list can be padded with dummies to match the length of the other). In an asymmetric merge, the ratio of list sizes is larger than constant.

Both flavors of merge, symmetric and asymmetric can be produced using the other flavor of merge as a subprotocol, and we will see in our constructions. In particular, the use of asymmetric merge as a subprotocol is a valuable tool for separating the lists into blocks and aligning blocks.

3.3 Aligning blocks

As in the PRAM merge algorithm discussed in §2.1.2, our secure merge protocols work by partitioning one list based on the values of the k-medians of the other list, which gives a collection of k subproblems, for some choice of k.

However, following the PRAM merge algorithm exactly would leak the size of these subproblems and in turn would leak the number of elements of L_2 lying between successive k-medians of L_1. Instead, we need tools for *obliviously* aligning blocks. Our protocols rely on two primary strategies.

First, our *lossy* strategy works by using an asymmetric merge on medians to count the number of elements from one list that would be assigned to each block of the other list. When the number of elements is within a constant factor of the block size, these elements can be matched with the corresponding block, creating subproblems for which secure symmetric merge can be applied. For blocks which have too many elements matched to them, the elements are extracted into an overflow list, which is treated separately. Our approach guarantees that the overflow list is sufficiently small, and so we apply a less efficient merge protocol to the entirety of the overflow list.

Second, our *lossless* strategy begins by producing, from two lists of size n, two expanded lists of size $2n$, with n dummy elements inserted into each list. These expanded lists are constructed so that they are "aligned": we can partition both lists into blocks of size n/k, run secure merge separately on these blocks, and simply concatenate the outputs (and remove dummy elements) to obtain the final merged list. More concretely, the dummy elements inserted into each list represent duplicates of the k medians from the opposite list, inserted into the appropriate position. We describe this insertion procedure further and show that it guarantees aligned blocks in Appendix D.

3.4 The Tag-Shuffle-Reveal Paradigm

We make repeated use of the *tag-shuffle-reveal* paradigm, which should be considered analogous to the shuffle-sort paradigm of prior sorting protocols, see e.g. [HKI+12], or an extension of the shuffle-reveal paradigm of [HICT14]. Our tag-shuffle-reveal paradigm is used implicitly in [FO21], but we expand its use here, and so describe it in more detail.

In the tag-shuffle-reveal paradigm, each element of a list is (obliviously) tagged with some label. This label can be (a secret sharing of) its current index, or it can be the result of some multiparty computation, for example a bit representing the output of a comparison against another value. Then, after shuffling the list, the tag or some part of the tag is opened, and the list entries are rearranged accordingly.

Because the shuffle step ensures that the tags are randomly ordered, The only requirement to ensure security is to ensure that the set of values the opened tags take on do not depend on the underlying data. The main application of this tag-shuffle-reveal paradigm is our extraction protocols, described below.

3.5 Extraction Protocols

Our construction relies on a collection of tools for extracting marked elements. The marked elements can be extracted and kept in their original order, or extracted and shuffled, or extracted into bins based on a tag they are marked with. The protocol for extracting marked elements and shuffling them is denoted $\mathbf{\Pi_{Extract}}$ (§B.1); the protocol for extracting marked elements in order is denoted $\mathbf{\Pi_{Extract-Ord}}$ (§B.2); and the protocol for extracting marked elements into bins is denoted $\mathbf{\Pi_{Extract-Bin}}$ (§B.3).

Each of these protocols rely on the same fundamental approach. First, the protocol obtains a count of the number of marked elements, either by local computation or as an input. Then, using secure comparisons, they add additional dummy elements, some marked, and some unmarked, so that the total number of marked elements is input independent. Then the list is shuffled, the tags are revealed, and elements are placed in their correct location. Finally, if we wish to extract the elements in order, we also tag them with their original location, perform secure computations (before or after shuffling) to adjust this tag to their correct final locations, and then open this tag as well.

Both of our strategies for aligning blocks mentioned above require insertion and removal of dummy elements. In the lossy strategy, elements are sorted into bins based on which block from

the opposite list they are matched with, and elements that would overflow their bins are extracted into a separate list. This requires a call to $\mathbf{\Pi_{Extract-Bin}}$ to obtain the elements for each bin, and a separate call to $\mathbf{\Pi_{Extract-Ord}}$ to produce an ordered list of all overflow elements.

In the lossless strategy, dummy elements representing n/k copies of the k-medians of each list are inserted into the opposite list, and then removed after merging on each subproblem. This requires a call to $\mathbf{\Pi_{Extract-Ord}}$ to remove the dummies from the final list while keeping the ordering of the non-dummy elements unaltered.

3.6 Prepared vs. Executed Secure Merge

A number of our protocols require us to tag elements with their new locations after performing a merging subprotocol. In order to clarify the notation and avoid requiring excessive unnecessary shuffling, we treat separately the act of tagging each element with its eventual (merged) position, referred to as *prepared* merge ($\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}}$), versus the act of actually moving elements as per their merge, referred to as *executed* merge ($\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{execute}}$).

A prepared or executed merge protocol can be transformed into the other variant using a single secure shuffle. To go from executed merge to prepared merge, we tag the elements in the original lists with their original indices, and then after merging tag them with their corresponding indices in the destination list. Then the parties shuffle the destination list, open the original indices, and return the elements to their original positions. Meanwhile, to go from prepared merge to executed merge, the parties merely shuffle together both tagged lists, open the tags, and move the elements to the indices indicated by their tags. We give the details in Appendix A.

3.7 Primitive Functionalities

Our protocols are realized with black box calls to the following "primitive" functionalities: $\mathbf{\Pi_{Open}}$, $\mathbf{\Pi_{Comp}}$, $\mathbf{\Pi_{Sel}}$, $\mathbf{\Pi_{Shuffle}}$ that act on additively shared secret values. We explain these functionalities here and describe how they might be realized.

In the functionality $\mathbf{\Pi_{Open}}$, the parties allow each other to learn the true value of any encrypted value. Under GMW-style MPC, each party sends their share to the other. Under homomorphic encryption, each party sends their share to the party who can decrypt, and then they broadcast the opened values.

In $\mathbf{\Pi_{Comp}}$, the parties learn shares of a bit denoting the result of a comparison operator $[x > y]$, $[x \geq y]$ or $[x == y]$. These comparisons can be computed by converting to shares of bits and applying garbled circuits, which requires at least $O(W \log W)$ boolean gates on words of W bits. A promising alternative is the GMW-based approach of Nishide and Ohta [NO07], which requires $O(W)$ bits of communication and $O(1)$ rounds of communication and is concretely efficient.

In $\mathbf{\Pi_{Sel}}$, the parties perform multiplication of two values $[b]$ and $[x]$, where b is known to be 0 or 1, and x can be any value. Equivalently, the parties compute shares of the ternary operator $b\ ?\ x : 0$. The value b can be XOR shared or additively shared over a larger field. $\mathbf{\Pi_{Sel}}$ can be realized using any standard MPC multiplication, or by using string-OTs with some rerandomizable encryption.

In $\mathbf{\Pi_{Shuffle}}$, the parties begin with shares of a given list, and end with shares of the same list, in some totally unknown order. To get the desired asymptotics, we require that the shuffle take $O(n)$ communication and $O(1)$ rounds of communication. Such protocols exist and can be realized via LWE with reasonably good ciphertext expansion [FO21]. At a high level, the shuffle works by allowing each party to hold an encryption of the list under the other party's secret key, and then shuffling and re-randomizing; see [FO21] for the full protocol and a more thorough treatment of it.

We recall the secure merge protocol from [FNO22] discussed above, which requires $O(n)$ communication and $O(n)$ rounds, which we call $\mathbf{\Pi_{SM-FNO}}$ in this work. Additionally, we use the trivial $O(n^2)$ communication, $O(1)$ round merge protocol, which works by making every possible pair of comparisons. We call this protocol $\mathbf{\Pi_{SM-ALL}}$ and give it formally in §C.1 for completeness.

4 Overview of Merge Protocols

Our main protocol is built from the four sub-protocols described below in §4.1, §4.2, §4.3, and §4.4.

4.1 Abstract Secure Symmetric Merge

Our Abstract Secure Symmetric Merge $\mathbf{\Pi_{SSM}}(n)$, which we give in Section 5.1, follows the *lossless* strategy for aligning blocks discussed in §3.3. In order to determine where the k medians of each list map to in the other list, we require a Secure Asymmetric Merge protocol $\mathbf{\Pi_{SAM}}(k, n)$, discussed below in §4.2 and given in full in Section 6. After determining where the k medians map to, we make n/k copies of each median, use a local protocol to determine each list element's new destination index, and the tag-shuffle-reveal paradigm to route all elements to their destination in the new, expanded list. After this, we run $\mathbf{\Pi'_{SSM}}(n/k)$ in parallel on each block.

We prove that this technique of expanding medians actually produces fully aligned blocks in Lemma D.1. We call this protocol *Abstract* Secure Symmetric Merge because it requires a choice of subprotocols to implement it. For our main result, we take $k = n/\log\log n$, and for $\mathbf{\Pi'_{SSM}}$ we use $\mathbf{\Pi_{SSM-FNO}}$.

Abstract Secure Symmetric Merge Protocol $\mathbf{\Pi_{SSM}}(n)$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share two *sorted* lists L_1 and L_2, each of size n.
Also as input, parameter k with $k|n$, and specifications of subprotocols $\mathbf{\Pi_{SSM'}}(n/k)$ and $\mathbf{\Pi_{SAM}}(k, n)$.

Output. The two lists have been merged (i.e. combined so that the final list is sorted) into an output list $L_1 \bigsqcup L_2$, which has size $2n$ and is (additively) secret-shared amongst the two parties.

RCost. $O(1) + \mathsf{RCost}(\mathbf{\Pi_{SSM'}}(n/k)) + \mathsf{RCost}(\mathbf{\Pi_{SAM}}(k, n))$.

CCost. $2k \cdot \mathsf{CCost}(\mathbf{\Pi_{SSM'}}(n/k)) + 2 \cdot \mathsf{CCost}(\mathbf{\Pi_{SAM}}(k, n)) + 8 \cdot \mathsf{CCost}(\mathbf{\Pi_{Op}})$.

Protocol.

1. Merge $\mathcal{M}_{2,k}$ with L_1 by invoking $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SAM}}(k, n)$. Denote the outputs as $(\widehat{\mathcal{M}_{2,k}}, \widehat{L}_1)$ (which are (additively) secret-shared amongst the two parties).
2. Run the $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{Dup}}(k, \widehat{\mathcal{M}}_{2,k}, \widehat{L}_1)$ protocol (see Figure 13), which creates k copies of $\mathcal{M}_{2,k}$ and assigns all elements a destination index. Call $\mathbf{\Pi_{Shuffle}}$ on the output, open the destination indices, and move the elements into a new list L'_1 as indicated by their destination indices.
3. Repeat Steps 1-2 above, with the roles of L_1 and L_2 swapped. Let L''_2 denote the final output (which has size $2n$ and is (additively) secret-shared amongst the two parties).
4. Partition L'_1 and L''_2 into $2k$ blocks, each of size n/k, and run $\mathbf{\Pi_{SSM'}}(n/k)$ on each block. Concatenate the results from each block, and let $\widehat{L}$ denote the output (which has size $4n$ and is (additively) secret-shared amongst the two parties).
5. Run the $\mathbf{\Pi_{Extract-Ord}}(\widehat{L} \bowtie ([\ell_i \neq 0])_i, 2n)$ protocol (see Figure 10) to remove the duplicates added in Steps 2 and 3 above. The output list has size $2n$, and is exactly (a secret-sharing of) $L_1 \bigsqcup L_2$.

Figure 1: Abstract Secure Symmetric Merge Protocol. In our main protocol, we take $k = \log\log n$, $\mathbf{\Pi_{SSM'}}(n/k) = \mathbf{\Pi_{SSM-FNO}}$, and $\mathbf{\Pi_{SAM}} = \mathbf{\Pi_{SAM}}(n, k, \mathbf{\Pi_{SSM-\log\log n}})$.

4.2 Abstract Secure Asymmetric Merge from Secure Symmetric Merge

We describe our Secure Asymmetric Merge protocol in Figure 2, and give proofs and analysis in §5.2. This protocol follows the lossy strategy for aligning blocks discussed in §3.3, and uses two auxiliary Secure Symmetric Merge protocols $\mathbf{\Pi_{SSM'}}, \mathbf{\Pi_{SSM''}}$. The protocol begins by calling $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SSM''}}(k)$ on the smaller list L_1 and the k medians of L_2. The parties now collectively hold information about which blocks of L_2 each value of L_1 is assigned to, as well as how many elements of L_1 will be assigned to each block of L_2.

Whenever at most n/k elements of the smaller list L_1 map to a block of L_2, we use $\mathbf{\Pi_{Extract-Bin}}$ to extract these elements, giving k subproblems of size n/k. We collect the remaining elements of

L_1 together with all of the corresponding blocks of L_2. By a counting argument, there are at most k elements in the extracted blocks of L_2, so this subproblem can also be solved by calling $\mathbf{\Pi}''_{\mathbf{SSM}}(k)$.

Finally, on the k blocks where we have n/k elements in a block of L_2 matched with at most n/k elements of L_1, we call $\mathbf{\Pi}_{\mathbf{SSM}'}(n/k)$. Through careful application of the tag-shuffle-reveal paradigm, we can now compute the destination indices for all list elements.

Secure Asymmetric Merge Protocol $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SAM}}(k, n)$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share *sorted* list L_1 of size k and L_2 of size n (for $k|n$). Also as input, specification of subprotocols $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SSM}'}(n/k)$ and $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SSM}''}(k)$.

Output. The two lists have been merged (i.e. combined so that the final list is sorted) into an output list $\overline{L_1 \bigsqcup L_2}$, which has size $k+n$ and is (additively) secret-shared amongst the two parties.

RCost. $O(1) + \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{SSM}'}(n/k)) + 2 \cdot \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{SSM}''}(k))$.

CCost. $(3n+k)\mathsf{CCost}(\mathbf{\Pi}_{\mathbf{Sel}}) + (5n+7k)\mathsf{CCost}(\mathbf{\Pi}_{\mathbf{Comp}}) + (7n+8k)\mathsf{CCost}(\mathbf{\Pi}_{\mathbf{Open}}) + 4\mathsf{CCost}(\mathbf{\Pi}_{\mathbf{Shuffle}}) + k \cdot \mathsf{CCost}(\mathbf{\Pi}_{\mathbf{SSM}'}(n/k)) + 2 \cdot \mathsf{CCost}(\mathbf{\Pi}_{\mathbf{SSM}''}(k))$.

Protocol.

1. Call inplace merge on L_1 and the k-th medians of L_2: $(\widehat{L}_1, \widehat{L}_2) := \mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SSM}''}(L_1, \mathcal{M}_{2,k})$.
   Write $\widehat{L}_j := L_j \bowtie (\iota_{i,j})_i$ and define $(\kappa_{i,j})_i := \iota_{i,j} - i$.
2. Define $([\sigma_i])_i := (i + [\kappa_{i,1}] \cdot n/k)_i$, so that σ_i is the correct index of $\ell_{i,1}$ if $\ell_{i,1}$ is less than all elements of the corresponding block of L_2.
3. Choose random secrets $([r_i])$ and define $\overline{L}_1 := L_1 \bowtie ([\sigma_i])_i \bowtie ([r_i])_i \bowtie ([e_i])_i$.
4. Each party computes shares of $[\lambda_i] := [\kappa_{i,1}] \cdot [(\kappa_{i,1} - \kappa_{i-n/k,1}) \neq 0]$. In other words, we tag the first n/k elements of L_1 which belong between v_{j-1} and v_j in $\mathcal{M}_{2,k}$ with the label j.
5. Write $(L^A_{1,m})_m := \mathbf{\Pi}_{\mathbf{Extract-Bin}}(\overline{L}_1 \bowtie (\lambda_i)_i, n/k, [\kappa_{i-1,2}] + 1, [\min\{\kappa_{i,2} - \kappa_{i-1,2}, n/k\}])$, for $m = 1, \ldots, k$, that is, $L^A_{1,m}$ holds all elements tagged with label m, in order, followed by dummies, for $1 \leq m \leq k$, and $|L^A_{1,m}| = n/k$.
6. Construct a shared list $T := ([\kappa_{i,2} - \kappa_{i-1,2} > n/k]) \bowtie ([\kappa_{i,2}])$ of length k, and extend it to length n by duplicating each share n/k times to give an expanded list $T_n := (\tau_i)_i \bowtie (\kappa_{i,T})_i$.
7. Define $[e_i] := i$ for all i and $\overline{L}_2 := L_2 \bowtie ([\kappa_{i,T}]) \bowtie ([e_i])_i$.
8. Call the extraction algorithm and compute $L^B_2 := \mathbf{\Pi}_{\mathbf{Extract-Ord}}(\overline{L}_2 \bowtie ([\tau_i])_i, k)$, that is, L^B_2 contains every block of L_2 with at least n/k elements of L_1 mapped to it, zipped with the flags $\kappa_{i,T}$ and e_i.
9. Compute $L^A_2 := [\overline{L}_2 \cdot (1 - \tau_i)]$, that is, L^A_2 contains all blocks of $\overline{L}_2$ not extracted into L^B_2, in their original positions. Partition L^A_2 into blocks $(L^A_{2,m})_h$ each of size n/k.
10. Compute $\overline{L}^A_{1,m} := L^A_{1,m} \cdot (1 - \tau_{mn/k})$ to replace all blocks $L^A_{1,m}$ with dummies if the corresponding block $L^A_{2,m}$ is dummy.
11. For $m = 1, \ldots, k$, perform $(\widehat{L}^A_{1,m}, \widehat{L}^A_{2,m}) := \mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SSM}'}(L^A_{1,m}, L^A_{2,m})$ on each of the k blocks of size n/k, with $\widehat{L}^A_{j,m} := L^A_{j,m} \bowtie (\mu^A_{i,j,m})_i$.
12. Let $L^B_1 := L_1$ and perform $(\widehat{L}^B_1, \widehat{L}^B_2) := \mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SSM}''}(L^B_1, L^B_2)$, with $\widehat{L}^B_j := L^B_j \bowtie (\mu^B_{i,j})_i$
13. Define:
$$L^C_1 = \bigcup_m \left(L^A_{1,m} \bowtie ([\mu^A_{i,1,m} - i + [\sigma_i])_i \bowtie (1)_i \right) \bigcup L^B_1 \bowtie ([(\mu^B_{i,1} - i) \mod n/k] + [\sigma_i])_i \bowtie (2)_i$$
14. Take $\widehat{L}^C_1 := \mathbf{\Pi}_{\mathbf{Shuffle}}(L^C_1)$. Open the secrets $[r_i]$ and the tags 1 or 2 to match elements marked 1 from a set $L^A_{1,m}$ with elements marked 2 from the set L^B_1. Construct a new list L^D_1 by using $\mathbf{\Pi}_{\mathbf{Sel}}$ to copy in the element marked 1 if it is nondummy and the paired element marked 2 otherwise.
15. Define:
$$\widehat{L} = L^D_1 \bigcup \left(L^B_2 \bowtie ([\mu^B_{i,2}] - i + [e_i])_i \right) \bigcup_m \left(L^A_{2,m} \bowtie ([\mu^A_{i,2,m}] - i + [\kappa_{i,T}] + [e_i])_i \right)$$
   Compute $\overline{L} := L \bowtie ([e_i])_i := \mathbf{\Pi}_{\mathbf{Extract}}(\widehat{L} \bowtie ([\widehat{\ell}_i \neq 0]), n+k)$.
16. Open the values $[e_i]$, and re-arrange the entries of L to match the opened values $[e_i]$.

Figure 2: Secure Asymmetric Merge Protocol. In our main protocol, we take $k = \log\log n$, $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SSM}'} = \mathbf{\Pi}_{\mathbf{SSM-FNO}}$, and $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SSM}''} := \mathbf{\Pi}_{\mathbf{SSM}-\log\log\mathbf{n}}$.

4.3 Building Block: Secure Merge with $O(n \log\log n)$ Communication

As an ingredient for secure asymmetric merge, we need a secure symmetric merge with $O(n \log\log n)$ communication and $O(\log\log n)$ rounds, which we call $\mathbf{\Pi_{SSM-\log\log n}}$ (Figure 3). This protocol in turn builds off of the $(n^{1/3}, n)$ asymmetric merge below, and follows the lossy strategy for aligning blocks described in §3.3.

As in $\mathbf{\Pi_{SAM}}(n, k)$, taking the k-medians of L_1 (in this case, with $k = n^{1/3}$) and using the $\mathbf{\Pi_{SAM-n^{1/3}}}$ protocol to merge with L_2 allows us to obtain destination indices for each element of the merged list, and counts of the number of elements from L_2 mapped to each block of L_1. We can similarly do the asymmetric merge of the k-medians of L_2 with L_1 to count the number of elements from L_1 mapped to each block of L_2.

For each block of L_1 that maps entirely inside a block of L_2 in the merged list, we extract that block together with all (less than n/k) corresponding elements from L_2, to get a subproblem of size $n/k = n^{2/3}$. After this process is complete, each block of L_2 has at most $2n/k$ elements mapped to it. Extracting these elements gives subproblems of size $2n/k$, which can be cut in half to give subproblems of size n/k.

When we count subproblems in this way, we have too many for the recursion to work correctly. However, as we show in Lemma 5.1, most of the subproblems created using a naive approach would contain all dummy elements. Shuffling and removing fully-dummy subproblems at each step gives us the recursion $\mathsf{CCost}(\mathbf{\Pi_{SSM-\log\log n}})(n) = O(n) + n^{1/3}\mathsf{CCost}(\mathbf{\Pi_{SSM-\log\log n}})(n^{2/3})$, from which the desired bounds follow immediately.

4.4 Building Block: Secure Asymmetric Merge on $(n^{1/3}, n)$

For the special case where $|L_1| = O(n^{1/3})$ and $|L_2| = O(n)$, we present in Figure 4 an asymmetric merge protocol with communication complexity $O(n)$ and round complexity $O(1)$. By calling $\mathbf{\Pi_{Compare-ALL}}$ on L_1 and the $n^{2/3}$ medians of L_2, we can identify every block of L_2 which contains an element of L_1 after merging the two lists. Because there are $O(n^{1/3})$ elements of L_1, there are $O(n^{1/3})$ such blocks of L_2. After extracting these blocks, we run $\mathbf{\Pi_{Compare-ALL}}$ again on L_1 and the $O(n^{2/3})$ elements of the extracted blocks of L_2. After some careful accounting of the index shifts during these steps, we get the destination indices for all the elements, which gives the desired merge protocol.

In Section 6, we give an extension of this protocol, which uses recursion to construct an asymmetric merge protocol with communication complexity $O(n)$ and round complexity $O(1)$ for a list L_1 of size $O(n^\alpha)$ and any fixed $\alpha < 1$, where the implied constants depend on α.

Secure Symmetric Merge $\mathbf{\Pi_{SSM\text{-}\log\log}}(n)$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share two *sorted* lists L_1 and L_2 of size n.

Output. The two lists have been merged (i.e. combined so that the final list is sorted) into an output list $\overline{L_1 \bigsqcup L_2}$, which has size $2n$ and is (additively) secret-shared amongst the two parties.

RCost. $O(\log\log n)$ rounds.

CCost. $O(n \log\log n) \cdot \mathsf{CCost}(\mathbf{\Pi_{Open}} + \mathbf{\Pi_{Comp}} + \mathbf{\Pi_{Sel}}) + O(\log\log n)\mathsf{CCost}(\mathbf{\Pi_{Shuffle}}(n))$.

Protocol.

1. Compute $d = \frac{\log\log n}{\log(3/2)} - O(1)$ such that $8 \le n^{(2/3)^d} < 16\sqrt{2}$.
2. Define $A_0 := \{(L_1, L_2, 0)\}$, to be the top-level array of sub-problems of size n. Each subproblem consists of two lists L_1, L_2 and a final offset ω. Then, for $k = 0, \ldots, d$, do Steps 3 through 18.
3. For each pair $(L_1, L_2, \omega) \in A_{k-1}$, do Steps 4 through 16.
4. Call $(L_1 \bowtie \iota_{i,1}, \mathcal{M}_{2,n^{1/3}} \bowtie \theta_{i,2}) := \mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SAM-n^{1/3}}}(L_1, \mathcal{M}_{2,n^{1/3}})$ and $(L_2 \bowtie \iota_{i,2}, \mathcal{M}_{1,n^{1/3}} \bowtie \theta_{i,1}) := \mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SAM-n^{1/3}}}(L_2, \mathcal{M}_{1,n^{1/3}})$.
5. Compute interactively $[s_i] := ([\theta_{(i+1)n^{2/3},1} - \theta_{in^{2/3},1} == 0])$ for $i = 1, \ldots, n^{1/3}$. The indicator s_i denotes blocks of L_1 which map entirely inside of a block of L_2 during secure merge.
6. For $1 \le i \le n^{1/3}$ and $1 \le m \le n^{2/3}$, compute interactively $[\tau_{(i-1)n^{2/3}+m}] := [\iota_{in^{2/3},2} - \iota_{(i-1)n^{2/3},2} - n^{2/3} - 1]$ and $[t_i] := [\tau_i > 0]$. Then re-define $[\tau_i] := [\tau_i] \cdot [t_i]$. The indicator t_i denotes blocks of L_2 which envelop entire blocks of L_1 during secure merge, while τ_i counts the number of such blocks enveloped by the ith block of L_2.
7. Generate $\overline{L}_2 := L_2 \bowtie (u_i)_i$ by generating (locally) uniformly random tags u_i for each element of L_2.
8. Call $\overline{L}_2' := \mathbf{\Pi_{Extract-Ord}}(\overline{L}_2 \bowtie ([t_i])_i \bowtie ([t_i] \cdot ([\iota_{i,2}] - i))_i)$.
9. Call $(B_j)_j := \mathbf{\Pi_{Extract-Bin}}(\overline{L}_2')_i, 2n^{2/3}, [\theta_{i,1} - i], [\theta_{i+1,1} - \theta_{i,1} - n^{2/3}])$, where the output is a series of lists B_j, for $1 \le j \le n^{1/3}$.
10. Delete the last $n^{2/3}$ entries of each B_j.
11. Partition L_1 into blocks based on $\mathcal{M}_{1,n^{1/3}}$, and call the resulting blocks $(C_j)_j$, for $1 \le j \le n^{1/3}$. Append to A_{k+1} every pair $(B_j \cdot [s_j], C_j \cdot [s_j], \mathsf{ind}(B_j) + \mathsf{ind}(C_j))$, where $\mathsf{ind}(X)$ is the tag ω of subproblem X.
12. Call $\mathbf{\Pi_{Shuffle}}$ on each of $\overline{L}_2 \bowtie (i)_i$ and $\cup_j (B_j \bowtie (s_j)_i)$. Open all values u_i from each list, and match corresponding elements, then transfer the values $(s_j)_i$ to the shuffled list of $\overline{L}_2 \bowtie (i)_i$. In other words, set $\widehat{L}_2$ equal to some permutation of $L_2 \bowtie (i)_i \bowtie (s_j)_i$.
13. Perform $\mathbf{\Pi_{Shuffle}}$ again, open $(i)_i$, and rearrange to give $L_2'' := L_2 \cdot [1 - (\widehat{s}_j)_i]$. In other words, L_2'' holds every element of L_2 not already entered into A_{k+1}.
14. Partition L_2'' into $n^{1/3}$ blocks $(D_j)_j$ along the $n^{1/3}$-medians, as in Step Z.
15. Call $(E_j)_j := \mathbf{\Pi_{Extract-Bin}}(L_1 \bowtie ([1 - s_i] \cdot ([\iota_{i,1}] - i))_i, 2n^{2/3}, [\theta_{i,2} - i - n^{2/3}\tau_i], [\theta_{i+1,2} - \theta_{i,2} - n^{2/3}(1 + \tau_{i+1})])$.
16. Append every pair $(D_j, E_j, \mathsf{ind}(D_j) + \mathsf{ind}(E_j))$ to A_{k+1}.
17. Compute
$$S_k := \sum_{i=1}^{k} 4^{1+k-i} n^{1-\frac{1}{2^i}}.$$
18. Set $A_{k+1} = \mathbf{\Pi_{Extract}}(A_{k+1} \bowtie ([\sum [a_i \ne 0] \ne 0]), S_k)$.
19. Perform $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SM}}(L_1, L_2, \mathbf{\Pi_{SM-ALL}})$ on every pair $(L_1, L_2, [\omega]) \in A_d$, and add $[\omega]$ to all resulting shares $[\iota_{i,j}]$.

Figure 3: $O(n \log\log n)$ Secure Symmetric Merge Protocol

Cubic Secure Asymmetric Merge $(n^{1/3}, n)$ Protocol $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SAM\text{-}n^{1/3}}}(n^{1/3}, n)$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share *sorted* list L_1 of size $n^{1/3}$ and L_2 of size n.

Output. The two lists have been merged (i.e. combined so that the final list is sorted) into an output list $L_1 \bigsqcup L_2$, which has size $n + n^{1/3}$ and is (additively) secret-shared amongst the two parties.

RCost. $O(1)$.

CCost. $(3n + 2n^{2/3} + 2n^{1/3})\mathsf{CCost}(\mathbf{\Pi_{Comp}}) + (4n + 2n^{2/3})\mathsf{CCost}(\mathbf{\Pi_{Open}}) + 3\mathsf{CCost}(\mathbf{\Pi_{Shuffle}})(n + n^{2/3})$.

Protocol.

1. Run compare all on the list L_1 and the $n^{2/3}$-medians of L_2, that is,
$$(\widehat{L}_1, \widehat{L}_2) := \mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SM-ALL}}(L_1, \mathcal{M}_{2,n^{2/3}}).$$
Write elements of $\widehat{L}_j$ as $(a_{i,j}, \iota_{i,j})$ so that $\widehat{L}_j = L_j \bowtie (\iota_{i,j})_i$.
2. Construct a shared list $T := ([\iota_{i,2}] - i) \bowtie ([\iota_{i,2} - \iota_{i-1,2} > 1])$ of length $n^{2/3}$, and a second list T_n formed by copying each element of T a total of $n^{1/3}$ times, so that T_n has total length n. Define τ_i to be i plus the first coordinate of the ith element of T_n, that is, write $T_n := ([\tau_i] - i) \bowtie (\upsilon_i)$. Note that $[\tau_i]$ denotes the correct index of $a_{i,2}$ after merging if no elements of L_1 lie in the block to which the ith element of L_2 belongs, and υ_i is an indicator for whether $a_{i,2}$ lies in a block that contains an element of L_1 after merging.
3. Define $e_i := i$ for all i, and construct $\overline{L}_2 := L_2 \bowtie ([\tau_i]) \bowtie ([e_i]) \bowtie ([\upsilon_i])$, a sequence of ordered 4-tuples.
4. Construct $\overline{L}_2'$ similarly, but reverse the condition tested by T, that is, replace each element $([\tau_i], [\upsilon_i]) \in T_n$ with $([\tau_i], 1 - [\upsilon_i])$.
5. Define $L_2^A := \mathbf{\Pi_{Extract}}(\overline{L}_2, n^{2/3})$ and $L_2^B := \mathbf{\Pi_{Extract}}(\overline{L}_2', n)$, so that the union of the nun-dummy elements of L_2^A and L_2^B is $\overline{L}_2$.
6. Run compare all on the lists L_1 and L_2^A, that is,
$$(\widehat{\widehat{L}}_1, \widehat{L}_2^A) := \mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SM-ALL}}(L_1, L_2^A).$$
Write elements of $\widehat{\widehat{L}}_1$ as $([a_{i,1}], [\kappa_{i,1}])$ and elements of $\widehat{L}_2^A$ as $([a_i^A], [\tau_{i,2}], [e_{i,2}], [\kappa_{i,2}])$.
7. Define $\overline{L}_2^A := ([a_i^A], [e_{i,2}] + [\kappa_{i,2}] - i, [e_{i,2}])$.
8. Construct the list $L_2^C := \mathbf{\Pi_{Shuffle}}(\overline{L}_2^A \cup L_2^B) := (\ell_i) \bowtie (\phi_{i,2}) \bowtie (e_{i,2})$.
9. Open the values $e_{i,2}$. Discard the dummy elements with $e_{i,2} = 0$ and arrange the remaining values $(\ell_i, \phi_{i,2})$ in order based on these values. Call the resulting list L_2^D.
10. Return $(L_1, i + n^{1/3} \cdot ([\iota_{i,1}] - i) + [\kappa_{i,1} \mod n^{1/3}])$ and L_2^D.

Figure 4: Cubic Secure Asymmetric Merge $(n^{1/3}, n)$ Protocol

5 Analyses of Protocols in Section 4

5.1 Analysis of Abstract Secure Symmetric Merge Protocol $\mathbf{\Pi_{SSM}}(n)$ of §4.1

5.1.1 Security

The security of the $\mathbf{\Pi_{SSM}}(n)$ protocol follows immediately from the security of the underlying $\mathbf{\Pi_{Dup}}(k,m,n)$, $\mathbf{\Pi_{Shuffle}}$, $\mathbf{\Pi_{Extract-Ord}}$, $\mathbf{\Pi_{SSM'}}(n/k)$, and $\mathbf{\Pi_{SAM}}(k,n)$ subprotocols.

5.1.2 Correctness

Assuming correctness of the $\mathbf{\Pi_{Dup}}(k,m,n)$, $\mathbf{\Pi_{Shuffle}}$, $\mathbf{\Pi_{Extract-Ord}}$, $\mathbf{\Pi_{SSM'}}(n/k)$, and $\mathbf{\Pi_{SAM}}(k,n)$ subprotocols, we need only demonstrate that the concatenation done in Step 4 above is correct, that is, that the blocks "align" as per the partitioning. Namely, this follows from Lemma D.1, which demonstrates that lists L_1'' and L_2'' have the same list of $2k$ medians (both equal $\mathcal{M}_{1,k} \bigsqcup \mathcal{M}_{2,k}$).

5.1.3 Cost

- Step (1) has $\mathsf{RCost}(\mathbf{\Pi_{SAM}}(k,n))$ and $\mathsf{CCost}(\mathbf{\Pi_{SAM}}(k,n))$.
- Step (2) has $\mathsf{RCost}(\mathbf{\Pi_{Shuffle}}(n)+O(1))$ and $\mathsf{CCost}(\mathbf{\Pi_{Shuffle}}(2n)+2n\cdot\mathbf{\Pi_{Open}})$, since $\mathbf{\Pi_{Dup}}$ can be performed locally (see Figures 13).
- Step (3) repeats the costs of (1) and (2).
- Step (4) has $\mathsf{RCost}(\mathbf{\Pi_{SSM'}}(n/k))$ and $2k\cdot\mathsf{CCost}(\mathbf{\Pi_{SSM'}}(n/k))$.
- Step (5) has $\mathsf{RCost}(\mathbf{\Pi_{Extract-Ord}}(4n,2n))$ and $\mathsf{CCost}(\mathbf{\Pi_{Extract-Ord}}(4n,2n))$.

Using the cost of $\mathbf{\Pi_{Extract-Ord}}$, the total cost is:

$$\begin{aligned}
\mathsf{RCost}(\mathbf{\Pi_{SSM}}(n)) &= \mathsf{RCost}(\mathbf{\Pi_{SAM}}(k,n)) + \mathsf{RCost}(\mathbf{\Pi_{Shuffle}}(2n)) + \\
&\quad \mathsf{RCost}(\mathbf{\Pi_{SSM'}}(n/k)) + \mathsf{RCost}(\mathbf{\Pi_{Extract-Ord}}(4n,2n)) \qquad (2)\\
&= O(1) + \mathsf{RCost}(\mathbf{\Pi_{SAM}}(k,n)) + \mathsf{RCost}(\mathbf{\Pi_{SSM'}}(n/k)) \\
\mathsf{CCost}(\mathbf{\Pi_{SSM}}(n)) &= 2\left[\mathsf{CCost}(\mathbf{\Pi_{SAM}}(k,n)) + \mathsf{CCost}(\mathbf{\Pi_{Shuffle}}(2n)) + 2n\cdot\mathsf{CCost}(\mathbf{\Pi_{Open}})\right] + \\
&\quad 2k\cdot\mathsf{CCost}(\mathbf{\Pi_{SSM'}}(n/k)) + \mathsf{CCost}(\mathbf{\Pi_{Extract-Ord}}(4n,2n)) \qquad (3)\\
&= 3\cdot\mathsf{CCost}(\mathbf{\Pi_{Shuffle}}(2n)) + 12n\cdot\mathsf{CCost}(\mathbf{\Pi_{Open}}) + 2n\cdot\mathsf{CCost}(\mathbf{\Pi_{Comp}}) + \\
&\quad 2\cdot\mathsf{CCost}(\mathbf{\Pi_{SAM}}(k,n)) + 2k\cdot\mathsf{CCost}(\mathbf{\Pi_{SSM'}}(n/k))
\end{aligned}$$

Plugging in $\mathbf{\Pi_{SM-FMO}}$ for $\mathbf{\Pi_{SSM'}}(n/k)$ in $\mathbf{\Pi_{SSM}}$ and $\mathbf{\Pi_{SAM}}$, and plugging in $\mathbf{\Pi_{SM-\log\log n}}$ for $\mathbf{\Pi_{SSM''}}(k)$ in $\mathbf{\Pi_{SAM}}$, (3) becomes:

$$\begin{aligned}
\mathsf{RCost}(\mathbf{\Pi_{SSM}}(n)) &= \mathsf{RCost}(\mathbf{\Pi_{SSM''}}(\tfrac{n}{k})) + 2\cdot\mathsf{RCost}(\mathbf{\Pi_{SSM'''}}(k)) \\
&= O(1) + O(\log\log n) + O(\log\log n) = O(\log\log n). \qquad (4)\\
\mathsf{CCost}(\mathbf{\Pi_{SSM}}(n)) &= O(n)
\end{aligned}$$

5.2 Analysis of Abstract Secure Asymmetric Merge $\mathbf{\Pi_{SAM}}(k,n)$ Protocol of §4.2

Security.
A simulator for the protocol for either party calls the simulator for each subprotocol, and then in Step 14 generates a random permutation of k elements tagged 1 and k elements tagged 2, and in step 16 generates random permutations of $\{1,\ldots,k\}$ and $\{1,\ldots,n\}$. By the correctness of the protocol, shown below, and the correctness of $\mathbf{\Pi_{Shuffle}}$ and $\mathbf{\Pi_{Extract}}$, Steps 14 and 16 do output uniformly random permutations of these elements during an honest run of the protocol. Also by the

correctness of the protocol, the required bounds for each call to an extraction protocol are satisfied, so these protocols can be simulated without leaking information. Security follows from the security of the underlying subprotocols.

Correctness.
To verify correctness, we proceed through the protocol, checking correctness of each claim about the intermediate tags used, the required bounds on number of dummy elements for the extraction protocols, and finally, verifying that the tags ι_i opened at the end give the correct destination indices.

In Step 2, $\kappa_{i,1}$ counts the number of medians from $\mathcal{M}_{2,k}$ less than $\ell_{i,1}$, so $\kappa_{i,1} \cdot n/k$ counts the number of elements in the blocks of L_2 which are placed entirely to the left of $\ell_{i,1}$, and σ_i holds the index of $\ell_{i,1}$ in the merged list, assuming that $\ell_{i,1}$ is less than all elements in the block of L_2 to which it is matched, as claimed.

In Step 4, $\kappa_{i,1} - \kappa_{i-n/k,1} = 0$ if and only if $\ell_{i,1}$ and $\ell_{i-n/k,1}$ are mapped into the sample block of L_2, so that there are more than n/k elements to the left of $\ell_{i,1}$ mapped into the same block. Thus $\lambda_i \neq 0$ precisely for the first n/k elements of L_1 mapped to each bucket of L_2, as claimed.

In Step 5, the above construction ensures that there are at most n/k elements tagged with each label, and they are consecutively placed within L_1. Because $\kappa_{i-1,2}$ counts the number of elements of L_1 less than the $i-1$st median of $\mathcal{M}_{2,k}$, $\kappa_{i-1,2}+1$ gives the first index of L_1 greater than the $i-1$st median of $\mathcal{M}_{2,k}$, that is, the index of the first element which maps into the ith block of L_2. Finally, $\kappa_{i,2} - \kappa_{i-1,2}$ counts the total number of elements which map into the ith block of L_2, which is equal to the number of elements of L_1 tagged with i unless there are more than n/k of them, in which case exactly n/k elements are tagged with i. Thus the conditions to call $\mathbf{\Pi_{Extract-Bin}}$ are satisfied.

In Step 6, the expression $\kappa_{i,2} - \kappa_{i-1,2} > n/k$ is an indicator that checks if more than n/k elements of L_1 map to the ith block of L_2. After duplicating n/k times to give the list T_n, the indicators τ_i correspond to elements which belong to blocks with this property.

In Step 8, let B be the number of blocks of L_2 that have more than n/k elements mapped to them. Then $n/kB < |L_1| = k$, so $B < k^2/n$, and the number of elements with $\tau_i = 1$ is less than $k^2/n \cdot n/k = k$, and the conditions to call $\mathbf{\Pi_{Extract-Ord}}$ are satisfied.

In Step 9, all elements not marked by τ are copied in place via calls to $\mathbf{\Pi_{Sel}}$. In Step 10, the corresponding blocks $L^A_{1,m}$ are copied in place. Since mn/k is of course an index from the mth block of L_2, $1 - \tau_{mn/k}$ is the desired indicator.

Now, since the tags $[e_i]$ opened in Step 16 allows all elements to be returned to their original positions, we must now only verify that the tags zipped in on Steps 13, 14, and 15 give the desired destination indices.

For non-dummy elements $\ell \in L^A_1$, the expression $\mu^A_{i,1,m} - i$ counts the number of elements from the mth block of L_2 less ℓ, so the tag $\mu^A_{i,1,m} - i + [\sigma_i]$ gives the correct destination index of ℓ. Any element ℓ of L_1 which is dummy in L^A_1 maps to one of the blocks that makes up L^B_2. Thus $\mu^B_{i,1} - i$ counts the total number of elements of L^B_2 less than ℓ, which is equal to n/k times the number of whole blocks of L^B_2 less than ℓ plus the number of elements from the block to which ℓ is mapped. Thus reducing $\mu^B_{i,1} - i$ modulo n/k gives the desired count of elements from L_2 in the matched block, and so adding σ_i gives the desired destination index. The procedure in Step 14 ensures that the computed destination index from L^A_1 is taken unless the element is dummy, in which case the computed destination index from L^B_1 is taken, as desired.

Elements of L_2 are rerouted into exactly one of $L^A_{2,m}, L^B_2$. For elements $\ell \in L^A_{2,m}$, the value $\kappa_{i,T}$ counts the number of elements of L_1 which sit entirely to the left of the mth block of L_2, while $\mu^A_{i,2,m} - i$ counts the number of elements from L_1 which map into the mth block of L_2, but land to the left of ℓ. Adding back in e_i gives the correct output index. Elements $\ell \in L^B_2$ are compared to the entire list L_1 in Step 12, so $\mu^B_{i,2} - i$ counts the number of elements from L_1 that land to the left, and adding back e_i gives the desired destination index.

Cost.
The cost of Steps 1, 11, and 12 are two calls to $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SSM}''}(k)$ and k calls to $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SSM}'}(n/k)$. Steps 2 and 3 can be performed locally. The cost of Step 4 is n calls to $\mathbf{\Pi_{Sel}}$ and n calls to $\mathbf{\Pi_{Comp}}$, which can be executed in parallel. The costs of Steps 5, 8, and 15 are $O(1)$ rounds and the communication is $(4n+3k)\mathsf{CCost}(\mathbf{\Pi_{Comp}}) + (6n+5k)\mathsf{CCost}(\mathbf{\Pi_{Open}}) + 3\mathbf{\Pi_{Shuffle}}(n+k)$. The cost of Step 6 is k calls to $\mathbf{\Pi_{Comp}}$ which can be executed in parallel. Steps 7 and 8 can be computed locally. Steps 9 and 10 require $2n$ calls to $\mathbf{\Pi_{Sel}}$. Step 14 requires a call to $\mathbf{\Pi_{Shuffle}}(2k)$ and k parallel calls to $\mathbf{\Pi_{Sel}}$. Steps 14 and 16 together require $n+3k$ calls to $\mathbf{\Pi_{Open}}$.

Additionally, the call to min in Step 5 requires k calls to $\mathbf{\Pi_{Comp}}$, and the call to mod in Step 13 requires $2k$ calls to $\mathbf{\Pi_{Comp}}$ (both min and mod can be computed by conditional subtraction of a public value, which requires no communication once shares of the conditional indicator have been generated). In total, this gives a cost of

$$O(1) + \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{SSM}'}(n/k)) + 2 \cdot \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{SSM}''}(k))$$

rounds and

$$\begin{aligned}\mathsf{CCost}(\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{SAM}}(k,n)) \;&= (3n+k)\cdot\mathsf{CCost}(\mathbf{\Pi_{Sel}}) + (5n+7k)\cdot\mathsf{CCost}(\mathbf{\Pi_{Comp}}) + (7n+8k)\cdot\mathsf{CCost}(\mathbf{\Pi_{Open}}) \\ &\quad + 4\cdot\mathsf{CCost}(\mathbf{\Pi_{Shuffle}})(n+k) + k\cdot\mathsf{CCost}(\mathbf{\Pi}_{\mathbf{SSM}'}(n/k)) + 2\cdot\mathsf{CCost}(\mathbf{\Pi}_{\mathbf{SSM}''}(k)).\end{aligned}$$

communication, with $O(n)$ computation.

5.3 Analysis of the $\mathbf{\Pi}_{\mathbf{SSM}\text{-}\log\log}(n)$ Protocol of §4.3

Security.
To simulate either player's view during an execution of the protocol, a simulator can call the simulator of the sub-protocols on every step except for Step 12 and Step 13, since these are the two steps where values are opened to the parties. On Step 12, both parties only see a list of random values, and a second list that is some random permutation of the first. In Step 13, the parties only see a random permutation of $\{1, 2, \ldots, n\}$. These views can be simulated by randomly sampling from these distributions.

Correctness.

The values from $\mathcal{M}_{1,n^{1/3}}$ and $\mathcal{M}_{2,n^{1/3}}$ each partition the range of possible values into $n^{1/3}+1$ blocks. There would be at most $n^{2/3}$ elements from either list between any two adjacent values. In an insecure protocol, we could recursively call $\mathbf{\Pi_{SM}}$ on each of these sub-problems.

To make this protocol secure, we do something similar, hiding the size of the sub-problem while ensuring that values can be efficiently transferred into sub-problems without any sub-problem growing too large.

First, we take every block from $\mathcal{M}_{1,n^{1/3}}$ that sits entirely inside of a block from $\mathcal{M}_{2,n^{1/3}}$. Such blocks certainly correspond to a sub-problem of size $(n^{2/3}, n^{2/3})$ or smaller, and we extract all such sub-problems in Steps 5-11.

We then remove all of these extracted elements from our original list in Steps 12-14. Now each block from $\mathcal{M}_{2,n^{1/3}}$ only touches remaining blocks from $\mathcal{M}_{1,n^{1/3}}$ if the block from $\mathcal{M}_{1,n^{1/3}}$ covers one or more of the endpoints of the block from $\mathcal{M}_{2,n^{1/3}}$. (The only other option is that the block from L_1 lies entirely within the block from L_2, and we have already removed all of these cases). Therefore there are at most 2 blocks from L_1 that touch any block of L_2, which gives the $2n^{2/3}$ bound in Step 15.

The nonzero tags $[t_i] \cdot [\iota_{i,2} - i]$ in Step 8 are arranged in increasing order ordered. The largest possible tag is equal to $|\mathcal{M}_{1,n^{1/3}}| = n^{2/3}$. The value $\theta_{i,1} - i$ denotes the index of the first entry of

L_2 that is greater than the ith element of $\mathcal{M}_{1,n^{1/3}}$, while $\theta_{i+1,1} - \theta_{i,1} - n^{2/3}$ counts the number of elements from L_2 that map to this block of L_1. Therefore, after removing the zero tags, the list $\overline{L}'_2$ meets the requirements of $\mathbf{\Pi}_{\mathbf{Extract-Bin}}$.

The correctness of the index lists in Step 15 are similar, with the values τ_i being used to shift the starting indices by the removed blocks and to deduct the removed blocks from the total counts.

To show that there are at most S_k sub-problems generated in steps 11 and 16, we prove the following lemma:

Lemma 5.1. *At a depth of d, there are at most*

$$S_d := \sum_{i=1}^{d} 4^{1+d-i} n^{1-\frac{2^i}{3^i}}$$

sub-problems without all elements dummy. For $d = \frac{\log\log n}{\log(3/2)} - O(1)$, we have

$$S_d \leq 8n^{1-\frac{2^d}{3^d}}.$$

Proof. We begin with the case $d = 1$. In Step 11, for every j with $s_j == 1$, $B_j \cdot [s_j]$ contains no dummies, and otherwise $(B_j \cdot [s_j], C_j \cdot [s_j])$ are all dummies. In Step 16, D_j contains no dummies unless some of the entries of the jth block of L''_2 are already assigned to a pair $(B_{j'} \cdot [s_{j'}], C_{j'} \cdot [s_{j'}])$ from Step 11. Thus at least one in four non-empty sets contains $n^{2/3}$ elements, so there are at most $8n/n^{2/3} = 8n^{1/3}$ non-dummy sets, so that $|A_1| \leq 4n^{1/3}$, that is, there are at most $4n^{1/3}$ non-dummy sub-problems of size $n^{2/3}$.

Similarly, at any depth d, there can be at most $\frac{2n}{n^{\frac{2^d}{3^d}}}$ sets containing $n^{\frac{2^d}{3^d}}$ elements, all non-dummy. Again, sets can be divided into groups such that at least $1/4$ of sets are entirely non-dummy, except when the sets B_j, C_j or D_j, E_j both contain dummy elements. Because the dummy elements are all contiguous, at most two blocks from each of the L_1 and L_2 sub-lists contain a mix of dummy and non-dummy elements, and so at most four additional sub-problems arise in this way containing elements both dummy and non-dummy from each prior sub-problem. This gives the recurrence

$$S_d = \frac{4n}{n^{\frac{2^d}{3^d}}} + 4S_{d-1},$$

which gives the first equation stated in the lemma. For the second equation, note that there exists some unique value of d, with $d < \frac{\log\log n}{\log(3/2)}$, such that $8 \leq n^{1/2^d} < 16\sqrt{2}$. We thus have, for $i < d$,

$$\frac{4^{1+d-i} n^{1-\frac{2^i}{3^i}}}{4^{1+d-(i+1)} n^{1-\frac{2^{i+1}}{3^{i+1}}}} = 4n^{-\frac{2^i 1}{3^{i+1}}} \leq \frac{1}{2},$$

and so

$$S_d \leq 4n^{1-\frac{1}{2^d}} \sum_{i=1}^{d} \left(\frac{1}{2}\right)^{d-i} < 8n^{1-\frac{2^d}{3^d}},$$

as desired. □

Cost.

At a depth k, Steps 4-16 require $O(n^{(2/3)^k})$ communication and computation and $O(1)$ rounds for each element of A_{k-1}. By the lemma, at a depth of k, there are at most $O(n^{1-(2/3)^k})$ subproblems, so the total work for Steps 3 through 18 is $O(n)$ communication and computation and $O(1)$ rounds. Since this loop is called $\lg\log n - O(1)$ times, the total communication is $O(n \log\log n)$ and the round complexity is $O(\log\log n)$, as desired.

5.4 Analysis of Secure Asymmetric Merge $\Pi_{\mathbf{SM-n^{1/3}}}$ Protocol of §4.4

Security.
Security follows from the security of the underlying protocols, after verifying that the conditions for the calls to $\mathbf{\Pi_{Extract}}$ are satisfied and that nothing is revealed in the $\mathbf{\Pi_{Open}}$ call in Step 9. We verify both of these conditions in our proof of correctness, below. **Correctness**.
The indicator υ_i introduced in Step 2 tests whether any elements of L_1 map into the ith block of L_2. Because there are $n^{1/3}$ total elements of L_1, there are at most $n^{1/3}$ such blocks, and so at most $n^{2/3}$ elements in the blocks, since the blocks have size $n^{1/3}$. Thus the condition for calling $\mathbf{\Pi_{Extract}}$ to generate L_2^A in Step 5 is satisfied. Since there are n elements total in L_2, the condition for L_2^B is also satisfied.

FOr elements of L_1, $\iota_{i,1} - i$ counts the number of blocks of L_2 situated entirely to the left, and $\kappa_{i,1}$ counts the number of elements of L_2 situated to the left, plus some additional blocks that are double counted. Thus, after modding out by $n^{1/3}$, The indices for L_1 computed in Step 10 are correct.

For elements $\ell \in L_2^A$, the value $\kappa_{i,2} - i$ counts the total number of elements from all of L_1 that end left of ℓ, so adding back in e_i gives the desired destination index. For element $\ell \in L_2^B$, the value τ_i computed in Step 2 is already the correct destination index, since no elements from L_1 maybe into the block of ℓ. Thus in Step 9, all elements of L_2 are given the correct destination index.
Cost.
Step 1 requires $n^{1/3} \cdot n^{2/3} = n$ calls to $\mathbf{\Pi_{Comp}}$. Step 2 requires $n^{2/3}$ calls to $\mathbf{\Pi_{Comp}}$. Step 3 and 4 can both be computed locally. Step 5 requires $n + n^{2/3}$ comparisons, $3n + n^{2/3}$ calls to $\mathbf{\Pi_{Open}}$, and two calls to $\mathbf{\Pi_{Shuffle}}(n)$. Step 6 again requires n comparisons. Step 7 is free. Step 8 requires another call to $\mathbf{\Pi_{Shuffle}}$. Step 9 requires $n + n^{2/3}$ calls to $\mathbf{\Pi_{Open}}$. Finally, Step 10 requires $n^{1/3}$ modular reductions, which in turn requires $2n^{1/3}$ calls to $\mathbf{\Pi_{Comp}}$.

This gives a total cost of $O(1)$ rounds and communication:

$$(3n + 2n^{2/3} + 2n^{1/3})\mathsf{CCost}(\mathbf{\Pi_{Comp}}) + (4n + 2n^{2/3})\mathsf{CCost}(\mathbf{\Pi_{Open}}) + 3\mathsf{CCost}(\mathbf{\Pi_{Shuffle}})(n + n^{2/3})$$

.

6 Secure n^α Asymmetric Merge Protocols

In this section, we generalize the Secure Asymmetric Merge protocol from §4.4 to work on a pair of lists L_1 of size n^α (for *any* constant $\alpha < 1$), and L_2 of size n, that runs in time $O(n)$ with $O(1)$, for any fixed $\alpha < 1$, where the implied constants depend on α. As we show below, the implied constants are small for $\alpha = 1/3$, and bounded above by $2^{2/(1-\alpha)^3}$ in general for communication and $2/(1-\alpha)^3$ for round complexity.

Our protocol works by bootstrapping up from an initial protocol for merging $n^{1,3}, n$, and going in general from

$$\beta \to \gamma = \frac{1}{\beta^2 - 3\beta + 3}.$$

We write $\mathbf{\Pi_{SM-n^\beta}}$ for a particular secure merge protocol for the constant $\beta < 1$, and $\mathbf{\Pi_{SAM-\gamma-\beta}}$ for the protocol that bootstraps from $\mathbf{\Pi_{SM-n^\beta}}$ to $\mathbf{\Pi_{SM-n^\gamma}}$.

Note that the construction of $\mathbf{\Pi_{SM-n^{1/3}}}$, which is the only subprotocol needed for our main $\mathbf{\Pi_{SSM}}$ protocol, was presented above in §4.4, and its analysis is given in §5.4

6.1 Bootstrapping Protocol $\Pi_{\mathbf{SM}-\beta-\gamma}$ Protocol

We give in Figure 5 a protocol that bootstraps from $\mathbf{\Pi_{SM-n^\beta}}$ to $\mathbf{\Pi_{SM-n^\gamma}}$, for γ close to β, in $O(1)$ rounds and roughly twice the communication of the previous level of bootstrapping.

Secure Asymmetric Merge Recursion Step Protocol $\mathbf{\Pi^{prepare}_{SAM\text{-}\gamma\text{-}\beta}}(n^\gamma, n, \mathbf{\Pi^{prepare}_{SAM-\beta}})$

Input. Two parties $\mathcal{P}_1, \mathcal{P}_2$ (additively) secret-share *sorted* list L_1 of size n^γ and L_2 of size n. Additionally, an algorithm $\mathbf{\Pi^{prepare}_{SAM-\beta}}$) is given, with

$$\gamma = \frac{1}{\beta^2 - 3\beta + 3}.$$

Output. The two lists have been merged (i.e. combined so that the final list is sorted) into an output list $L_1 \bigsqcup L_2$, which has size $n + n^\gamma$ and is (additively) secret-shared amongst the two parties.

RCost. $O(1) + \mathsf{RCost}(\mathbf{\Pi_{SM-n^\beta}})(n)$.

CCost. $O(n) + 2\mathsf{CCost}(\mathbf{\Pi_{SM-n^\beta}})(n)$.

Protocol.

1. Run compare all on the list L_1 and the $n^{1-\gamma}$-medians of L_2, that is,
$$(\widehat{L}_1, \widehat{L}_2) := \mathbf{\Pi^{prepare}_{SM-ALL}}(L_1, \mathcal{M}_{2,n^{1-\gamma}}).$$
Write elements of $\widehat{L}_j$ as $(a_{i,j}, \iota_{i,j})$ so that $\widehat{L}_j = L_j \bowtie (\iota_{i,j})_i$.
2. Generate a random secretly shared sequence $([r_i])_i$ of length $n^{1-\gamma}$ and construct a shared list $T := ([\iota_{i,2}] - i) \bowtie ([\iota_{i,2} - \iota_{i-1,2} > 1 + n^{\beta\gamma}]) \bowtie ([r_i])$. Generate a second list T_n formed by copying each element of T a total of n^γ times, so that T_n has total length n. As in Figure 4, define $T_n := ([\tau_i] - i, [\upsilon_i], [s_i])$, so that τ_i is the correct index of $a_{i,2}$ if no elements from L_1 map into the corresponding block of L_2 to which $a_{i,2}$ belongs, $[\upsilon_i]$ is an indicator for blocks matched to more than $n^{\beta\gamma}$ elements from L_1, and $[s_i]$ is the random value $[r_j]$ from the block to which $a_{i,2}$ belongs.
3. Define $e_i := i$ for all i, and construct $\overline{L}_2 := L_2 \bowtie ([\tau_i]) \bowtie ([e_i]) \bowtie ([s_i]) \bowtie ([\upsilon_i])$.
4. Define $([t_i]) := [\min\{\iota_{i,2} - \iota_{i-1,2} - 1, n^{\beta\gamma}\}]$ and $([c_i]) := [\iota_{i-1,2}] - (i-1) + 1$.
5. Construct shares of the value $[\sigma_i] := [\iota_{i,1} - i] \cdot [1 + \iota_{i,1} - \iota_{i-n^{\gamma\beta},1} > n^{\gamma\beta}]$ and $[\rho_i] := i + n^\gamma \cdot [\iota_{i,1} - i]$.
6. Define $(L^A_{2,k})_k$ as the partition of L^A_2 into blocks of size n^γ by the medians $\mathcal{M}_{2,n^{1-\gamma}}$ and $L^B_2 := \mathbf{\Pi_{Extract-Ord}}(\overline{L}_2, n^{\gamma(2-\beta)})$.
7. Define $(L^A_{1,k})_k := \mathbf{\Pi_{Extract-Bin}}(L_1 \bowtie ([e_i]) \bowtie ([\rho_i])_i \bowtie ([\sigma_i])_i, n^{\beta\gamma}, (c_i)_i, (t_i)_i)$ and $L^B_1 := \mathbf{\Pi_{Extract-Ord}}(L_1 \bowtie ([e_i]) \bowtie ([\rho_i])_i \bowtie [\sigma_i == 0], n^\gamma)$.
8. Call $(\widehat{L}^A_{1,k}, \widehat{L}^A_{2,k}) := \mathbf{\Pi^{prepare}_{SAM-\beta}}(L^A_{1,k}, L^A_{2,k})$ in parallel on each pair of blocks, for $k = 1, \ldots, n^{1-\gamma}$, and write $\widehat{L}^A_{j,k} := L^A_{j,k} \bowtie (\kappa^A_{i,j,k})_i$.
9. Divide L^B_1 into blocks $\mathcal{B}^B_k$ of size $n^{(\gamma(3-\beta))-1}$, and perform $(\widehat{\mathcal{B}}^B_k, \widehat{L}^B_{2,k}) := \mathbf{\Pi^{prepare}_{SAM-\beta}}(\mathcal{B}^B_k, L^B_{2,k})$ for each k, in parallel, and define $\kappa^B_{i,j,k}$ as above.
10. Define $\overline{L}^B_2 := L^B_2 \bowtie \left([\tau_i] + \left[\sum_{k=1}^{n^{1-\gamma(2-\beta)}} (\kappa^B_{i,2,k} - i)\right]\right)_i := L^B_2 \bowtie (\kappa_{i,2,0})_i$.
11. Partition $\overline{L}^B_2$ into blocks of size n^γ. Call $\mathbf{\Pi_{Shuffle}}((\widehat{L}^A_{2,k})_k \cup (\overline{L}^B_{2,k})_k)$ on blocks.
12. Open the values $[s_i]$ to match blocks, and for every pair of matching blocks, replace $[\kappa_{i,2,k}]$ with $[\kappa_{i,2,k} + \kappa_{i,2,0} - e_i]$. Call the result L^C_2.
13. Call $\mathbf{\Pi_{Shuffle}}(L^C_2)$, again on blocks, and open the values $[e_i]$ and re-order the elements accordingly. Call the result L^D_2.
14. Write L^C_1 for the union of all lists $L^A_{j,k}$ and $\mathcal{B}^B_k$, and define $\widehat{L}^C_1 := L^C_1 \bowtie ([\kappa_{i,j,k}] - i + [\rho_{i,j,k}])$, where i is the index of the element within the list $L^A_{j,k}$ or $\mathcal{B}^B_k$, and $\rho_{i,j,k}$ is the tag (appended in Step 5) of the corresponding element.
15. Call $\mathbf{\Pi_{Shuffle}}(L^C_1)$. Open the values $[e_i]$, discard dummies, and re-order the remaining elements. Call the result L^D_1.
16. Return L^D_1 and L^D_2.

Figure 5: Secure Asymmetric Merge (n^γ, n) Bootstrapping Protocol

Security.
Security follows from the security of the underlying protocols, after verifying that the conditions for the calls to $\mathbf{\Pi_{Extract}}$ are satisfied and that nothing is revealed in the $\mathbf{\Pi_{Open}}$ call in Steps 12, 13, and 15. We verify both of these conditions in our proof of correctness, below. **Correctness**.

The lists $(L^A_{1,k})$ contain the first $n^{\beta\gamma}$ elements mapped to each block of size n^γ of L_2, and so we can call $\mathbf{\Pi_{SM-n^\beta}}$ on these pairs of blocks. We extract into L^B_1 all elements after the first $n^{\beta\gamma}$. There are at most n^γ such elements. We extract into L^B_2 all blocks which have more than $n^{\beta\gamma}$ elements mapped to them. There are at most $n^{\gamma(1-\beta)}$ such blocks, and so at most $n^{\gamma(2-\beta)}$ such elements.

After dividing L^B_1 into blocks in Step 9, there are $n^{1-\gamma(2-\beta)}$ subproblems each of size $(n^{\gamma(2-\beta)}, n^{(\gamma(3-\beta))-1}$. By the assumption that $\gamma = 1/(\beta^2 - 3\beta + 3)$, we have $\beta \cdot ((\gamma(3-\beta)) - 1) = \gamma(2-\beta)$, so we can apply the $\mathbf{\Pi_{SM-n^\beta}}$ protocol here as well, in Step 9.

Correctness follows from similar index chasing as in the previous protocols. **Cost**.
Step 1 requires n comparisons. Step 2 requires $o(n)$ comparisons. Step 3 is free. Step 4 requires n calls to min which gives n calls to $\mathbf{\Pi_{Comp}}$. Step 5 requires $o(n)$ calls to $\mathbf{\Pi_{Comp}}$ and $\mathbf{\Pi_{Sel}}$. Because $n^{\gamma(2-\beta)} = o(n)$, Step 6 requires $n + o(n)$ calls to $\mathbf{\Pi_{Open}}$ and $o(n)$ calls to $\mathbf{\Pi_{Comp}}$. Similarly Step 7 requires $2n \cdot \mathbf{\Pi_{Comp}} + 3n \cdot \mathbf{\Pi_{Open}} + n \cdot \mathbf{\Pi_{Sel}}$ plus $o(n)$ communication. Step 8 requires $n^{1-\gamma}$ calls to $\mathbf{\Pi_{SM-n^\beta}}(n^{\beta\gamma}, n^\gamma)$, and similarly Step 9 requires $n^{1-\gamma}$ calls to $\mathbf{\Pi_{SM-n^\beta}}(n^{\beta\gamma}, n^\gamma)$. Finally, Steps 10 through 15 require $3n + o(n)$ calls to $\mathbf{\Pi_{Open}}$ and 3 shuffles.

This gives a total cost of $O(1) + \mathsf{RCost}(\mathbf{\Pi_{SM-n^\beta}})$ rounds and

$$(4n + o(n))\mathsf{CCost}(\mathbf{\Pi_{Comp}}) + (7n + o(n))\mathsf{CCost}(\mathbf{\Pi_{Open}}) + n\mathsf{CCost}(\mathbf{\Pi_{Sel}}) + 6\mathsf{CCost}(\mathbf{\Pi_{Shuffle}})(n) + n^{1-\gamma}\mathsf{CCost}(\mathbf{\Pi_{SM-n^\beta}}(n^\gamma) + n^{1-\gamma(2-\beta)})\mathsf{CCost}(\mathbf{\Pi_{SM-n^\beta}}(n^{\gamma(2-\beta)})$$

communication, and the last two terms can be bounded above by $2\mathsf{CCost}\mathbf{\Pi_{SM-n^\beta}}(n)$.

6.2 Secure Asymmetric Merge (n^α, n) Protocol

In this section, we give the protocol for performing secure asymmetric merge in time $O(n)$ and $O(1)$ rounds for fixed $\alpha < 1$. In terms of α, we give the upper bound of $O(n2^{2/(1-\alpha)^3})$ communication and $O(1/(1-\alpha)^3)$ rounds as $\alpha \to 1$.

The correctness and security of the $\mathbf{\Pi_{SAM-n^\alpha}}$ protocol follows immediately from correctness and security of $\mathbf{\Pi_{SAM-n^{1/3}}}$ and $\mathbf{\Pi_{SAM-\gamma-\beta}}$, as long as the sequence (γ_i) constructed in Step 1 of this protocol actually terminates. We show that in fact it terminates in at most $2/(1-\alpha)^3$ steps, from which the above bounds on communication and round complexity follow from the communication and round complexity of the previous two protocols.

But indeed, for any $\beta < \alpha$, we have

$$\frac{1}{\beta^2 - 3\beta + 3} - \beta = \frac{(1-\beta)^3}{(1-\beta)(2-\beta)+1} \geq (1-\beta)^3/3 \geq (1-\alpha)^3/3.$$

Since we begin with $\beta = 1/3$, it requires at most

$$\frac{\alpha - 1/3}{(1-\alpha)^3/3} \leq \frac{2}{(1-\alpha)^3}$$

steps of the recurrence before the sequence of (γ_i)'s becomes greater than α.

Secure Asymmetric Merge (n^α, n) Protocol $\mathbf{\Pi_{SAM\text{-}n^\alpha}}(n^\alpha, n)$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share *sorted* list L_1 of size n^α and L_2 of size n (for $\alpha \leq 1/2$).

Output. The two lists have been merged (i.e. combined so that the final list is sorted) into an output list $\overline{L_1 \bigsqcup L_2}$, which has size $n + n^\alpha$ and is (additively) secret-shared amongst the two parties.

RCost. $O(1/(1-\alpha)^3)$.

CCost. $O(n \cdot 2^{2/(1-\alpha)^3})$.

Protocol.

1. Construct a finite sequence (γ_i) as follows:
   - Set $\gamma_1 := 1/3$, and $i = 1$.
   - While $\gamma_i < \alpha$, set $\gamma_{i+1} \leftarrow \frac{1}{\gamma_i^2 - 3\gamma_i + 3}$ and $i \leftarrow i + 1$.
2. Let ℓ equal the length of (γ_i). Then for $i = 1, \ldots, \ell - 1$, call $\mathbf{\Pi_{SAM-\gamma_{i+1}-\gamma_i}}(n^{\gamma_{i+1}}, n, \mathbf{\Pi_{SAM-\gamma_i}})$ to construct the protocol $\mathbf{\Pi_{SAM-\gamma_{i+1}}}$. This gives a protocol $\mathbf{\Pi_{SAM-\gamma_\ell}}$, for $\gamma_\ell > \alpha$.
3. Pad the list L_1 with $n^{\gamma_\ell} - n^\alpha$ dummy elements, and call the result $\overline{L}_1$.
4. Call $\mathbf{\Pi_{Extract}}(\mathbf{\Pi_{SAM-\gamma_\ell}}(\overline{L}_1, L_2))$ to obtain the desired result, extracting all non-dummy elements.

Figure 6: Secure Asymmetric Merge (n^α, n) Protocol

A Converting between Prepared Merge and Executed Merge

We give here the formal definitions and conversion protocols between secure *prepared merge* and secure *executed merge*. Secure executed merge, or $\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{execute}}$, is the usual secure merge functionality, where parties input their additive shares of secret sorted lists L_1, L_2 and receive back additive shares of the merged list $L_1 \bigsqcup L_2$.

Secure prepared merge is a functionality that takes the same inputs, but instead outputs to each party shares of the lists $\widehat{L}_j := L_j \bowtie (\iota_{i,j})_i$, where $\iota_{i,j}$ is the index of the ith element of L_j in $L_1 \bigsqcup L_2$. By convention, we require that, when elements from L_1 and L_2 have equal values, the elements from L_1 are to the left of the elements from L_2 in the final merged list.

Below, we give a protocol for constructing a prepared merge protocol from an existing executed merge, and a protocol for constructing an exected merge protocol from an existing prepared merge.

A.1 Prepared Merge from Executed Merge

A.1.1 Analysis

Security.
Steps 1, 3, 6, and 7 are performed locally, and so are automatically secure. The security of Steps 2 and 4 follow from the security of $\mathbf{\Pi_{Shuffle}}$ and the underlying secure merge protocol $\mathbf{\Pi_{SM}}$. For Step 5, note that for $1 \leq k \leq n_j$, there is a unique entry i with $f_i = k$ and $g_i = j$.

Therefore both players learn only a random shuffling of pairs (i, j) for $j \in \{1, 2\}$ and $1 \leq i \leq n_j$, and either party could simulate the interaction with the other by generating such a random shuffling in Step 5 randomly, and computing the other party's message from this information.

Correctness.
In step 3, the value h_i holds the destination index of each entry of the list, and steps 6 and 7 ensure that the values $(a_{i,j}, h_i) = (a_{i,j}, \iota_i)$ are placed back in their original positions in L_j.

Cost.
The only interactive cost beyond the costs of $\mathbf{\Pi_{SM}}$ and $\mathbf{\Pi_{Shuffle}}$ is the cost of Step 5, which requires $2(n_1 + n_2)\mathsf{CCost}(\mathbf{\Pi_{Op}})$ communication and $O(1)$ rounds. This gives the numbers stated in the protocol

Prepared Merge from Executed Merge $\mathbf{\Pi}_{\mathsf{execute}\to\mathsf{prepare}}(L_1, L_2, \mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{execute}})$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share *sorted* lists L_1, L_2, with elements $\ell_{i,1} \in L_1$, $\ell_{i,2} \in L_2$. Additionally, we have a secure executed merge protocol $\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{execute}}$.

Output. A pair of shared lists $\widehat{L}_1, \widehat{L}_2$, such that $\widehat{L}_j = L_j \bowtie (\iota_{i,j})_i$, and the index of $\ell_{i,j}$ in the merged list $\overline{L_1 \bigsqcup L_2}$ is $\iota_{i,j}$.

RCost. $O(1) + \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{Shuffle}}(n)) + \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{SM}}(n))$.

CCost. $\mathsf{CCost}(\mathbf{\Pi}_{\mathbf{Shuffle}}(n, 4)) + (2n_1 + 2n_2) \cdot \mathsf{CCost}(\mathbf{\Pi}_{\mathbf{Op}}) + \mathsf{CCost}(\mathbf{\Pi}_{\mathbf{SM}}(n, 3))$.

Protocol.

1. Generate the list $\overline{L_1}$ made up of elements $(\ell_{i,1}, \iota_{i,1}, 1)$, and likewise let $\overline{L_2}$ be the list of elements $(\ell_{i,2}, \iota_{i,2}, 2)$, where here $\iota_{i,j} := i$ is the index of the ith element of L_j.
2. Compute $L = \mathbf{\Pi}_{\mathbf{SM}}(\overline{L_1}, \overline{L_2})$.
3. Append to each element $([e_i], [f_i], [g_i])$ of L the element $[h_i] = i$.
4. Compute $\widehat{L} = \mathbf{\Pi}_{\mathbf{Shuffle}}(L)$.
5. Open the values $\widehat{f}_i, \widehat{g}_i$.
6. Set $\widehat{L}_1$ equal to $([\widehat{e}_i], [\widehat{h}_i])$, for all i with $\widehat{g}_i = 1$, ordering the elements of $\widehat{L}_1$ by the corresponding values of $\widehat{f}_i$.
7. Set $\widehat{L}_2$ equal to $([\widehat{e}_i], [\widehat{h}_i])$, for all i with $\widehat{g}_i = 2$, ordering the elements of $\widehat{L}_2$ by the corresponding values of $\widehat{f}_i$.

Figure 7: Secure Prepared Merge from Secure Executed Merge

A.2 Executed Merge from Prepared Merge

Exected Merge from Prepared Merge $\mathbf{\Pi}_{\mathsf{prepare}\to\mathsf{execute}}(L_1, L_2, \mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}})$

Input.Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share *sorted* lists L_1, L_2, with elements $\ell_{i,1} \in L_1$, $\ell_{i,2} \in L_2$. Additionally, we have a secure prepared merge protocol $\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}}$.

Output. Each party holds additive shares of the merged list $L = L_1 \bigsqcup L_2$.

RCost. $O(1) + \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{Shuffle}}(n)) + \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}}(n))$.

CCost. $\mathsf{CCost}(\mathbf{\Pi}_{\mathbf{Shuffle}}(n, (1, 1, 0))) + (n_1 + n_2) \cdot \mathsf{CCost}(\mathbf{\Pi}_{\mathbf{Op}}) + \mathsf{CCost}(\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}}(n, (1, 1, 0)))$.

Protocol.

1. Each party locally generates shares of the sequence $(e_i)_i$, with $e_i := i$.
2. Generate the lists $\overline{L}_j := L_j \bowtie ([e_i])_i$, that is, replace $[a_{i,j}]$ with $([a_{i,j}], [e_i])$.
3. Each party computes their share of $(\widehat{L}_1, \widehat{L}_2)$ using $\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}}(\overline{L}_1, \overline{L}_2)$. Write $\widehat{L}_j = L_j \bowtie (\widehat{\iota}_{i,j})$.
4. Set $\widehat{L} = \mathbf{\Pi}_{\mathbf{Shuffle}}(\widehat{L}_1 \cup \widehat{L}_2)$. Call the resulting elements $([a_i], [\iota_i])$.
5. Open the values ι_i.
6. In an empty list L of length $n_1 + n_2$, place at index j the (unique) $[a_i]$ with $\iota_i = j$, and return L.

Figure 8: Secure Executed Merge from Secure Prepared Merge

A.2.1 Analysis

Security.
Steps 1 and 2 are performed locally, and so are secure. The security of steps 3 and 4 follow from the security of the protocols $\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}}$ and $\mathbf{\Pi}_{\mathbf{Shuffle}}$ (in other words, a simulator for $\mathbf{\Pi}_{\mathbf{inPlace}\to\mathbf{SM}}$ can perform steps 1 and 2 as in the actual protocol, and use the simulator for the subprotocols in steps

3 and 4).

In Step 5, each party sees a uniformly random permutation of $\{1, \ldots, n_1 + n_2\}$, which can be generated at random by a simulator. Note that the correctness of $\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}}$ guarantees that each value in this range occurs exactly once as an entry of ι_i, and the security of $\mathbf{\Pi}_{\mathbf{Shuffle}}$ guarantees that these values are distributed randomly.

Correctness.
By the correctness of $\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}}$, step 6 places each value $\ell_{i,j}$ into their correct final position in the output list L.

Cost.
The only steps that requires interaction are Step 3 and Step 4, where subprotocols are called, and Step 5, where n_1+n_2 values are opened, requiring $(n_1+n_2)\mathbf{\Pi}_{\mathbf{Open}}$ communication and $O(1)$ rounds. It follows immediately that round complexity is $O(1) + \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{Shuffle}}(n)) + \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}}(n))$ and communication is $\mathsf{CCost}(\mathbf{\Pi}_{\mathbf{Shuffle}}(n,(1,1,0))) + (n_1+n_2)\cdot\mathsf{CCost}(\mathbf{\Pi}_{\mathbf{Op}}) + \mathsf{CCost}(\mathbf{\Pi}_{\mathbf{SM}}^{\mathsf{prepare}}(n,(1,1,0)))$.

B Extraction Protocols

B.1 Extracting Unordered Marked Elements

Extraction Protocol $\mathbf{\Pi}_{\mathbf{Extract}}(A, t)$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share a list A of size n of elements of the form (a_i, ι_i), of which $t' \leq t$ elements are marked by $\iota_i = 1$, and the other elements have $\iota_i = 0$.

Output. Each party holds additive shares of a list B of length t containing all elements a_i with $\iota_i = 1$, shuffled randomly.

RCost. $O(1) + \mathsf{RCost}(\mathbf{\Pi}_{\mathbf{Shuffle}}(n))$.

CCost. $\mathsf{CCost}\mathbf{\Pi}_{\mathbf{Shuffle}}(n,(1,0,1)) + t\mathsf{RCost}(\mathbf{\Pi}_{\mathbf{Comp}}) + (n+t)\mathsf{RCost}(\mathbf{\Pi}_{\mathbf{Open}})$.

Protocol.

1. Generate shares of the total number of marked elements $[t'] := [\sum_{i=1}^{n} \iota_i]$. This can be done locally.
2. For $i = n+1, \ldots, n+t$, append to A the element:
$$(d_i, \iota_i) := (0, \delta_i).$$
where δ_i is an indicator on $(i-n) > t'$. Each party sets their share of d_i to be 0, and computes their share of ι_i using parallel invocations of a comparison protocol.
3. Invoke the shuffling protocol and set $\widehat{A} := \mathbf{\Pi}_{\mathbf{Shuffle}}(A)$.
4. Open all values $\widehat{\iota_i}$.
5. Initialize B to an empty list.
6. For $i = 1, \ldots n+t$, if $\widehat{\iota_i} = 1$, each party copies their share of the value $\widehat{a_i}$ to B.

Figure 9: Extract and Shuffle Marked Elements Protocol

B.1.1 Analysis

Security.
Security of steps 1, 2, and 3 follow from security of the underlying comparison and shuffling protocols. By correctness (shown below), the values $\widehat{\iota_i}$ will contain t 1's and n 0's, and after calling the $\mathbf{\Pi}_{\mathbf{Shuffle}}$ protocol, these values will be randomly arranged in $\widehat{A}$, and so both party's view in Step 4 can be generated uniformly at random by a simulator.

Correctness.
There are exactly $t - t'$ values in $\{n+1, \ldots, n+t\}$ with $i - n > t'$, and so Step 2 adds $t - t'$ values

with $\iota_i = 1$ to the existing t' values with $\iota_i = 1$ in A. These are the values placed into B in Step 6, and they are randomly distributed by the correctness of $\mathbf{\Pi_{Shuffle}}$.

Cost.
Step 2 requires t comparison operations $\mathbf{\Pi_{Comp}}$ and Step 4 requires $n + t$ opening operations. Combining with Step 3 gives $O(1) + \mathsf{RCost}(\mathbf{\Pi_{Shuffle}}(n))$ rounds and $\mathsf{CCost}\mathbf{\Pi_{Shuffle}}(n, (1, 0, 1)) + t\mathsf{RCost}(\mathbf{\Pi_{Comp}}) + (n + t)\mathsf{RCost}(\mathbf{\Pi_{Open}})$.

B.2 Extracting Ordered Marked Elements

Extraction Protocol $\mathbf{\Pi_{Extract-Ord}}(A, t)$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share a *sorted* list A of size n of elements of the form (a_i, ι_i), of which $t' \leq t$ elements are marked by $\iota_i = 1$, and the other elements have $\iota_i = 0$.

Output. Each party holds additive shares of a list B of length t containing all elements a_i with $\iota_i = 1$, in order, followed by dummy elements.

RCost. $O(1) + \mathsf{RCost}(\mathbf{\Pi_{Shuffle}}(n))$.

CCost. $\mathsf{CCost}\mathbf{\Pi_{Shuffle}}(n, (1, 1, 1)) + t \cdot \mathsf{RCost}(\mathbf{\Pi_{Comp}}) + (n + 2t) \cdot \mathsf{RCost}(\mathbf{\Pi_{Open}})$.

Protocol.

1. Each party computes locally shares of $[e_i] := (\sum_{j \leq i}[\iota_i])$, and define $[t'] := [e_n]$ to be a share of the total number of marked elements.
2. Generate locally shares of the list: $\overline{A} := (a_i)_i \bowtie ([\iota_i]) \bowtie ([e_i])_i$.
3. For $i = n + 1, \ldots, n + t$, compute $[\delta_i]$ to be an indicator representing the comparison $(i - n) > t'$.
4. For $i = n + 1, \ldots, n + t$, append to $\overline{A}$ the element: $([a_i], [\iota_i], [e_i]) := (0, [\delta_i], [\delta_i] \cdot (i - n))$.
5. Invoke the shuffling protocol and set $\widehat{A} := \mathbf{\Pi_{Shuffle}}(\overline{A})$.
6. Open all values $\widehat{\iota_i}$.
7. Open all values $\widehat{e_i}$ for values $\widehat{\iota_i} = 1$.
8. Initialize B to an empty list.
9. For $i = 1, \ldots t$, set the i^{th} element of B to the (unique) element $[\widehat{a}_j]$ with $\widehat{e}_j = i$.

Figure 10: Stably Extract Marked Elements Protocol

B.2.1 Analysis

Security.
Security of steps 1, 3, 4, and 5 follow from security of the underlying selection, comparison, and shuffling protocols. By correctness (shown below), the values $\widehat{e_i}$ correspond to the integers from 1 to t, and after calling the $\mathbf{\Pi_{Shuffle}}$ protocol, these values will be randomly arranged in $\widehat{A}$, and so both party's view in Step 6 can be generated uniformly at random by a simulator. All remaining steps are performed locally, and can be imitated by a simulator.

Correctness.
As in $\mathbf{\Pi_{Extract}}$, there are exactly t values in $\{1, \ldots, n + t\}$ with $e_i \neq 0$, the values which will be extracted into B. For the t' non-dummy values from A with $\iota_i = 1$, the index $[e_i]$ counts the number of non-dummy values to the left of a_i in A, and so $[e_i]$ holds the destination index of a_i, and collectively, the indices $[e_i]$ take on the values $\{1, \ldots, t\}$.

The remaining $t - t'$ values in $\{n + 1, \ldots, n + t\}$ with $i - n > t'$ are in fact $n + t' + 1, \ldots, n + t$, and so the corresponding values $[e_i]$ are $t' + 1, \ldots, t$. Thus the nonzero values of e_i, for $1 \leq i \leq n + t$, collectively cover the values $\{1, \ldots, t\}$, with the first t' values being the non-dummy elements of A, in ordered, as desired.

Cost.

Note that Step 1 requires linear *computation* when implemented via the recursion $[e_i] = [\iota_i] + [e_{i-1}]$, and zero communication, since it can be performed locally.

Step 2 requires t comparison operations $\mathbf{\Pi_{Comp}}$ and Step 4 requires $n + t$ opening operations. Combining with Step 3 gives $O(1) + \mathsf{RCost}(\mathbf{\Pi_{Shuffle}}(n))$ rounds and $\mathsf{CCost}\mathbf{\Pi_{Shuffle}}(n, (1, 1, 1)) + t \cdot \mathsf{RCost}(\mathbf{\Pi_{Comp}}) + (n + 2t) \cdot \mathsf{RCost}(\mathbf{\Pi_{Open}})$.

B.3 Stably Extracting into Bins

Stable Bin Extraction Algorithm $\mathbf{\Pi_{Extract-Bin}}(A, t, C, T')$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share a *sorted* list A of size n of elements of the form (a_i, ι_i), with $\iota_i \in \{0, \ldots, k\}$, and $t'_j \leq t$ elements have $\iota_i = j$, for $j = 1, \ldots, k$, with $tk \leq \ell n$, for some given constant ℓ. Additionally, we are guaranteed that all elements with $\iota_i = j$, for $j \neq 0$, are in a contiguous block. Finally, all parties hold shares of another pair of lists (C, T'), each of length k, with the property that $[c_j]$ is the index i of the first element of A with $\iota_i = j$, and $[t'_j]$ is the number of elements of A with $\iota_i = j$. (if no such element ι_i exists, $[t'_j] := [c_j] := 0$.)

Output. Each party holds additive shares of a sequence of lists (B_j), for $j = 1, \ldots, k$, where each list B_j has length t and contains all elements a_i with $\iota_i = j$, in the same order that they occur in A, followed by dummy elements.

RCost. $O(1) + \mathsf{RCost}(\mathbf{\Pi_{Shuffle}}(n))$.

CCost. $\mathsf{CCost}(\mathbf{\Pi_{Shuffle}}((1 + \ell)n, (1, 2, 0))) + (1 + 2\ell) \cdot \mathsf{CCost}(\mathbf{\Pi_{Comp}}) + (2 + 2\ell)n \cdot \mathsf{CCost}(\mathbf{\Pi_{Open}}) + (1 + \ell)n \cdot \mathsf{CCost}(\mathbf{\Pi_{Sel}})$.

Protocol.

1. Append tk elements to A, where for $n + t(j-1) < i \leq n + tj$, we define
$$(a_i, \iota_i) := (0, [i - (n + t(j-1)) > t'_j]).$$
2. Generate the shared list $A' = A \bowtie ([\kappa_i])_i$, where κ_i is equal to i reduced modulo t. This can be done locally by $\mathcal{P}_1$ setting their share equal to κ_i and all other parties setting their share equal to zero.
3. Invoke the shuffling algorithm and set $\widehat{A} := \mathbf{\Pi_{Shuffle}}(A')$.
4. Open all values $\widehat{\iota_i}$.
5. For $i = 1, \ldots, n + tk$, let $j = \widehat{\iota_i}$, and set $[\widehat{\kappa_i}] \leftarrow ([\widehat{\kappa_i}] - [c_j - 1] \cdot [\widehat{a_i} \neq 0]) \pmod t$.
6. Open all values $\widehat{\kappa_i}$.
7. Intialize (B_j) as a sequence of k lists of length t.
8. Set the ℓ-th element of B_j equal to the (unique) element of $\widehat{A}$ with $\widehat{\iota_i} = j$ and $\widehat{\kappa_i} = \ell$.

Figure 11: Stably Extract Marked Elements By Bin Protocol

B.3.1 Analysis

Security.
A simulator can simulate the view of either party during Steps 1, 3, and 5 by invoking a simulator of the underlying protocols, and can imitate Steps 2, 7, and 8 exactly, since these steps are performed locally. Security for these steps follows from the security of hte underlying protocol.

We show in our discussion of correctness that, in Step 6, the parties learn a random permutation of the tk ordered pairs (i, j) with $1 \leq i \leq t$ and $1 \leq j \leq k$, while in Step 4, the parties learn the first coordinates of all terms in that permutation. During Step 4 of a simulated execution of the protocol, the simulator generates and stores the permutation for Step 6 uniformly at random, and then computes the resulting values for Step 4.

Correctness.

As in $\mathbf{\Pi_{Extract}}$ and $\mathbf{\Pi_{Extract-Ord}}$, the comparison operation in Step 1 adds $t - t'_j$ elements with $\iota_i = j$. The sequence of values κ_i for these dummy values is $\{t'_j + 1, \ldots, t-1, 0\}$, since the term with $\kappa_i = t$ is reduced modulo t.

Before Step 5, the remaining non-dummy values with $\iota_i = j$ have values κ_i equal to the sequence $\{c_j, c_j + 1, \ldots, c_j + t'_j - 1\}$, taken modulo t, where entry $c_j + (h-1)$ corresponds to the hth element of A with $\iota_i = j$. Therefore, after Step 5, the values κ_i take on the values $\{1, 2, \ldots, t'_j\}$, where entry k corresponds to the hth element of $\iota_i = j$.

Therefore, for $1 \le h \le t$ and $1 \le j \le k$, every pair (h, j) occurs exactly once in Step 6, and, for a fixed j, the first t'_j entries of B_j are the non-dummy elements of A with $\iota_i = j$, in order, as desired.

Cost.
Step 1 requires $tk \le \ell n$ comparisons. Steps 4 and 6 require $(1+\ell)n$ opening of secret values. Step 5 requires k subtractions, which of course can be performed locally, $(1+\ell)n$ comparisons (to zero) and $(1+\ell)n$ selection operations. Steps 2, 7, and 8 are performed locally. Each of Steps 1,4,5, and 6 therefore require $O(1)$ rounds.

This gives a total round cost of $O(1) + \mathsf{RCost}(\mathbf{\Pi_{Shuffle}}(n)$ and a total communication cost of $\mathsf{CCost}(\mathbf{\Pi_{Shuffle}}((1+\ell)n, (1,2,0))) + (1+2\ell) \cdot \mathsf{CCost}(\mathbf{\Pi_{Comp}}) + (2+2\ell)n \cdot \mathsf{CCost}(\mathbf{\Pi_{Open}}) + (1+\ell)n \cdot \mathsf{CCost}(\mathbf{\Pi_{Sel}})$, with linear total computation by each party.

C Other Sub-Protocols

In this section, we present other sub-protocols invoked by any of the Secure Merge protocols above.

C.1 Secure Symmetric Merge via Compare All

The following protocol (Figure 12) is a naïve protocol that simply performs all n^2 possible comparisons (securely) and is useful for terminating iterative/recursive processess when the reduced list size n is sufficiently small.

Secure Merge - Compare All $\mathbf{\Pi^{prepare}_{SM\text{-}ALL}}(n_1, n_2)$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share two *sorted* lists L_1 and L_2, of size n_1, n_2, with elements $\ell_{i,j}$, for $j \in \{1,2\}$. When $n_1 = n_2 = n$, we write $\mathbf{\Pi^{prepare}_{SM\text{-}ALL}}(n)$.

Output. Lists $\widehat{L}_1$, $\widehat{L}_2$, with $\widehat{L}_j = L_j \bowtie (\iota_{i,j})_i$, where $\iota_{i,j}$ denotes the index of $\ell_{i,j}$ in $L_1 \bigsqcup L_2$.

RCost. $O(1)$ rounds.

CCost. $n_1 n_2 \cdot \mathsf{CCost}(\Pi_{Op})$.

Protocol.

1. Compute shares $[e_{i,j}] = [(\ell_{i,1} > \ell_{j,2})]$ for every pair $(i,j) \in \{1, \ldots, n_1\} \times \{1, \ldots, n_2\}$.
2. Locally compute $[\iota_{i,1}] = \sum_{j=1}^{n_2} [e_{i,j}]$ and $[\iota_{i,2}] = n_1 - \sum_{j=1}^{n_1} [e_{j,i}]$.
3. Return $(\widehat{L}_1, \widehat{L}_2) := (L_1 \bowtie (i + \iota_{i,1})_i, L_2 \bowtie (i + \iota_{i,2})_i)$

Figure 12: Secure Merge - Compare All Protocol

C.1.1 Analysis of the $\mathbf{\Pi^{prepare}_{SM\text{-}ALL}}(n_1, n_2)$ Protocol

Security.
Security follows from the security of the underlying comparison protocol used in Step 1, since Steps 2 and 3 are performed locally.

Correctness.
By convention, we require elements from L_1 to be to the left of elements from L_2 if their values are equal.

The value $e_{i,j}$ is nonzero precisely when $\ell_{i,1} > \ell_{j,2}$, so $\iota_{i,1}$ counts the number of such values $\ell_{j,2}$. In the merged list $L_1 \bigsqcup L_2$, the entry $\ell_{i,1}$ will have $(i-1)$ elements from L_1 to its left, and $\iota_{i,1}$ elements from L_2 to its left, giving its final position as $i + \iota_{i,1}$.

Similarly $\sum_{j=1}^{n_1} e_{j,i}$ counts values $\ell_{j,1} > \ell_{i,2}$, and so $n_2 - \sum e_{j,i}$ counts the number of values $\ell_{j,1} \leq \ell_{i,2}$, and $i + \iota_{i,2}$ gives the destination index of $\ell_{i,2}$.

Cost.

- Step (1) incurs $O(1)$ round cost (the cost of the underlying comparison protocol) and $2n_1n_2\mathsf{CCost}(\mathbf{\Pi_{Comp}})$, since the shares $[e_{i,j}]$ and $[f_{i,j}]$ can all be computed in parallel, and there are $2n_1n_2$ such comparisons to compute.
- Steps (2) and (3) can be performed locally.

C.2 Duplicate Values In Place $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{Dup}}(k, L_1 \bowtie (\iota_{i,1})_i, L_2 \bowtie (\iota_{i,2})_i)$

Duplicate Values In Place $\mathbf{\Pi}^{\mathsf{prepare}}_{\mathbf{Dup}}(k, L_1 \bowtie (\iota_{i,1})_i, L_2 \bowtie (\iota_{i,2})_i)$

Input. Two parties $\mathcal{P}_1$, $\mathcal{P}_2$ (additively) secret-share two *sorted* lists $\widehat{L}_1$ and $\widehat{L}_2$, of sizes n_1, n_2 respectively, with $\widehat{L}_j := L_j \bowtie (\iota_{i,j})_i$ with the property that $\iota_{i,j}$ is the index of $\ell_{i,j}$ in the merged list $L_1 \bigsqcup L_2$.

Output. Two (sorted, additively secret-shared) output lists $\widehat{L}'_1, \widehat{L}'_j$ of size kn_1 and n_2, respectively, with the property that $\widehat{L}'_j := L'_j \bowtie (\iota'_{i,j})_i$, $L'_2 = L_2$, and L'_1 consists of k copies of each entry of L_1, sorted in place. The values $\iota'_{i,j}$ give the index of the entires $\ell'_{i,j}$ in the merged list $L'_1 \bigsqcup L'_2$ of size $n_2 + k \cdot n_1$.

RCost. None.

CCost. None. (i.e. this protocol requires only $O(n_2 + kn_1)$ local computations).

Protocol.

1. Each party constructs their share of the list L'_1 by replacing each share $[\ell_{i,1}]$ with k copies, now in positions $[\ell_{k\cdot(i-1)+m}]$, for $1 \leq m \leq k$.
2. Each party generates their shares of the list $(\iota'_{i,1})_i$ by setting $[\iota'_{k\cdot(i-1)+m}] := ([\iota_i] - i) + k \cdot (i-1) + m$, for $1 \leq m \leq k$.
3. Each party generates their shares of the list $(\iota'_{i,2})_i$ by setting $[\iota'_{i,2}] := k \cdot ([\iota_i] - i) + i$.
4. Return $(L'_1 \bowtie (\iota'_{i,1})_i, L_2 \bowtie (\iota'_{i,2})_i)$.

Figure 13: Duplicate Values In Place Protocol

C.2.1 Analysis

Security.
This protocol is executed locally, and is therefore secure.

Correctness.
Each entry $\ell_{i,1}$ in L_1 has $\iota_{i,1} - i$ elements from L_2 to its left in $L_1 \bigsqcup L_2$, since it has $\iota_{i,1} - 1$ elements total to its left, of which $i-1$ come from L_1. The corresponding element $\ell'_{k\cdot(i-1)+m}$, for $1 \leq m \leq k$, will likewise have $\iota_{i,1} - i$ elements from L_2 to its left in $L'_1 \bigsqcup L'_2$, and so a total of $([\iota_i] - i) + k \cdot (i-1) + m - 1$ elements to its left.

Similarly, each entry $\ell_{i,2}$ in L_2 has $\iota_{i,2} - i$ elements from L_1 to its left in $L_1 \bigsqcup L_2$, and $k \cdot (\iota_{i,2} - i)$ elements to its left in $L'_1 \bigsqcup L'_2$, and correctness follows.

Cost.
Because this protocol is executed locally, it requires zero rounds and no communication. We need to check only that it requires $O(n_2 + kn_1)$ computation. Steps 1 and 2 each require $O(kn_1)$ computation, and Step 3 requires $O(n_2)$ computation.

D Results on Medians

Notice that if we write $\mathcal{M}_{j,k} = \{v_1, v_2, \ldots, v_k\}$, then:

(a)	$\lvert\mathcal{M}_{j,k}\rvert = k$	
(b)	If $k = 1$, then:	$\mathcal{M}_{j,1} = \{u_n\}$ (the last element of L_j)
(c)	If $k = n = \lvert L_j\rvert$, then:	$\mathcal{M}_{j,n} = L_j$
(d)	$\forall v_i \in \mathcal{M}_{j,k}$:	There are at least $i \cdot \left(\frac{n}{k}\right)$ items in L_j that are $\leq v_i$
(e)	$\forall v_i \in \mathcal{M}_{j,k}$ and $v_i > v \in L_j$:	There are fewer than $i \cdot \left(\frac{n}{k}\right)$ items in L_j that are $\leq v$
(f)	$\forall v_i \in \mathcal{M}_{j,k}$:	There are at least $1 + (k-i) \cdot \left(\frac{n}{k}\right)$ items in L_j that are $\geq v_i$
(g)	$\forall v_i \in \mathcal{M}_{j,k}$ and $v_i < v \in L_j$:	There are fewer than $1 + (k-i) \cdot \left(\frac{n}{k}\right)$ items in L_j that are $\geq v$
(h)	$v_k = u_n$	(i.e. the last element of $\mathcal{M}_{j,k}$ is the last element of L_j) (5)

For any value $v \in S$, where S is a totally ordered set, let:

$$\begin{aligned}
LT_{L_j}(v) &= \text{Number of elements in } L_j \text{ that are less than } v \\
LTE_{L_j}(v) &= \text{Number of elements in } L_j \text{ that are less than or equal to } v \\
GT_{L_j}(v) &= \text{Number of elements in } L_j \text{ that are greater than } v \\
GTE_{L_j}(v) &= \text{Number of elements in } L_j \text{ that are greater than or equal to } v
\end{aligned}$$

Note that each of the above four functions are monotone step functions with respect to v (increasing for LT and LTE and decreasing for GT and GTE), with jumps at each $v \in L_j$. Then for any $v \in L_j$ and any $v_i \in \mathcal{M}_{j,k}$:

$$n = LTE_{L_j}(v) + GT_{L_j}(v) = LT_{L_j}(v) + GTE_{L_j}(v) \tag{6a}$$

$$LTE_{L_j}(v_i) \geq i \cdot \left(\frac{n}{k}\right) \tag{6b}$$

$$LTE_{L_j}(v) < i \cdot \left(\frac{n}{k}\right) \quad \Leftrightarrow \quad v_i > v \tag{6c}$$

$$LTE_{L_j}(v) \geq i \cdot \left(\frac{n}{k}\right) \quad \Leftrightarrow \quad v_i \leq v \tag{6d}$$

$$LTE_{L_j}(v) = i \cdot \left(\frac{n}{k}\right) \quad \Rightarrow \quad v_i = v \tag{6e}$$

$$GTE_{L_j}(v_i) \geq 1 + (k-i) \cdot \left(\frac{n}{k}\right) \tag{6f}$$

$$GTE_{L_j}(v) < 1 + (k-i) \cdot \left(\frac{n}{k}\right) \quad \Leftrightarrow \quad v_i < v \tag{6g}$$

$$GTE_{L_j}(v) \geq 1 + (k-i) \cdot \left(\frac{n}{k}\right) \quad \Leftrightarrow \quad v_i \geq v \tag{6h}$$

$$GTE_{L_j}(v) = 1 + (k-i) \cdot \left(\frac{n}{k}\right) \quad \Rightarrow \quad v_i \leq v \tag{6i}$$

where (6a) is by definition; (6b) and (6f) are simply (5.d) and (5.f), respectively; the left implication of (6c) and (6g) are simply (5.e) and (5.g), respectively, and the right implication is from (6b) and (6f) together with the fact that LTE and GTE are monotone (increasing/decreasing, respectively) functions in v; (6d) and (6h) are the contrapositive of (6c) and (6g), respectively; and (6e) is because $v_i \leq v$ (by (6c)) and also $v_i \geq v$ by (6h) together with (6a) (and similarly (6i) follows from (6a), (6g), and (6d)).

The correctness of the above protocols will rely on the "alignment" property claimed in the Secure Symmetric Merge rubric above. Formally:

Lemma D.1. *Let L_1 and L_2 denote two (sorted) lists of size n, and let $\mathcal{M}_{1,k}$ and $\mathcal{M}_{2,k}$ denote their k medians (as per (1)). Let $L_1' = L_1 \bigsqcup \mathcal{M}_{2,k}$ denote the list (of size $2n$) resulting from merging $\mathcal{M}_{2,k}$ with L_1, with each element in $\mathcal{M}_{2,k}$ duplicated n/k times in L_1'. Let $\mathcal{M}_{2k}'$ denote the $2k$ medians of L_1'. Then the $2k$ medians of L_1' are exactly the (merged) k medians of L_1 and L_2:*

$$\mathcal{M}_{2k}' = \mathcal{M}_{1,k} \bigsqcup \mathcal{M}_{2,k}$$

Proof. Denote the lists of k medians as:

$$\mathcal{M}_{1,k} = \{u_1, u_2, \ldots, u_k\}$$
$$\mathcal{M}_{2,k} = \{v_1, v_2, \ldots, v_k\}$$

Let $\mathcal{M}_{2k} = \mathcal{M}_{1,k} \bigsqcup \mathcal{M}_{2,k}$, and denote $\mathcal{M}_{2k}$ and $\mathcal{M}_{2k}'$ as:

$$\mathcal{M}_{2k} = \{w_1, w_2, \ldots, w_{2k}\}$$
$$\mathcal{M}_{2k}' = \{z_1, z_2, \ldots, z_{2k}\}$$

Then the claim is that $\mathcal{M}_{2k}' = \mathcal{M}_{2k}$, which we argue by demonstrating that $z_i = w_i$ for all $1 \le i \le 2k$. Notice that $|L_1'| = 2n$ and $|\mathcal{M}_{2k}'| = 2k$, and by definition $\mathcal{M}_{2k}'$ evenly partitions L_1' into $2k$ blocks, with each block of size n/k. Furthermore, notice that L_1' is the merge of two lists:

- n elements are L_1
- n elements are the k elements of $\mathcal{M}_{2,k}$, each duplicated n/k times.

In particular, if we write $L_2' := \bigcup_{\frac{n}{k}} \mathcal{M}_{2,k}$ (i.e. L_2' is $\mathcal{M}_{2,k}$ duplicated n/k times), then observe that the k medians of L_2' are exactly the same as the k medians of L_2, namely $\mathcal{M}_{2,k}$. Therefore, if we write (as *multi-sets*): $L_1' = L_1 \cup L_2'$, then (as per (5)):

$$\begin{array}{lll} \text{For any } u_i \in \mathcal{M}_{1,k}\text{:} & \text{There are at least } i \cdot \left(\frac{n}{k}\right) & \text{items in } L_1', \textbf{\textit{from}}\ \boldsymbol{L_1}, \text{ that are} \le u_i. \\ & \text{There are at least } 1 + (k-i) \cdot \left(\frac{n}{k}\right) & \text{items in } L_1', \textbf{\textit{from}}\ \boldsymbol{L_1}, \text{ that are} \ge u_i. \\ \text{For any } v_i \in \mathcal{M}_{2,k}\text{:} & \text{There are at least } i \cdot \left(\frac{n}{k}\right) & \text{items in } L_1', \textbf{\textit{from}}\ \boldsymbol{L_2'}, \text{ that are} \le v_i. \\ & \text{There are at least } (1 + k - i) \cdot \left(\frac{n}{k}\right) & \text{items in } L_1', \textbf{\textit{from}}\ \boldsymbol{L_2'}, \text{ that are} \ge v_i. \end{array} \tag{7}$$

Notice that the last statement in (7) (unlike the second statement) does not come directly from (5.e). In particular, it relies on the fact that the medians of L_2' are exactly described by $\mathcal{M}_{2,k}$ (the medians of L_2), and each such median appears a total of n/k times in L_2'. This is why the '1' appears *inside* the parentheses (as a multiplicative factor of the n/k) as opposed to outside (as an additive constant). We conclude the proof by showing that $z_i = w_i$ for all $1 \le i \le 2k$ by showing both inequalities:

> $\underline{z_i \ge w_i}$. By (6d), this will follow if we can show that there are at least $i \cdot \frac{2n}{2k}$ values in L_1' that are less than or equal to w_i. Let $x = x_i$ denote the maximal index such that $u_x \in \mathcal{M}_{1,k}$ appears *among the first i coordinates* of $\mathcal{M}_{2k}$. Similarly, let $y = y_i$ denote the the maximal index such that $v_y \in \mathcal{M}_{2,k}$ appears *among the first i coordinates* of $\mathcal{M}_{2k}$. By definition of $\mathcal{M}_{2k}$ (as the merge of $\mathcal{M}_{1,k}$ and $\mathcal{M}_{2,k}$), we have that $x_i + y_i = i$. And then by (7) above, we have that there are at least $x_i \cdot \frac{n}{k}$ elements in L_1 that are less than or equal to u_x, and since u_x is in the first i coordinates of $\mathcal{M}_{2k}$, we have that $u_x \le w_i$, and hence (by monotonicity of LTE) there are at least $x_i \cdot \frac{n}{k}$ elements in L_1 that are less than or equal to w_i. Similarly, there are at least $y_i \cdot \frac{n}{k}$ elements in L_2' that are less than or equal to w_i. Consequently, there are at least $(x_i + y_i) \cdot \frac{n}{k} = i \cdot \frac{n}{k}$ values in L_1' that are less than or equal to w_i, as required.
>
> $\underline{z_i \le w_i}$. By (6h), this will follow if we can show that there are at least $1 + (2k - i) \cdot \frac{2n}{2k}$ values in L_1' that are greater than or equal to w_i. Let $x = x_i$ denote the *minimal* index such that $u_x \in \mathcal{M}_{1,k}$ appears *at or after coordinate i* of $\mathcal{M}_{2k}$ (or define $x_i = k + 1$ if none of the values

in $\mathcal{M}_{1,k}$ appear at or after coordinate i of $\mathcal{M}_{2k}$). Similarly, let $y = y_i$ denote the the minimal index such that $v_y \in \mathcal{M}_{2,k}$ appears *at or after coordinate* i of $\mathcal{M}_{2k}$ (or define $y_i = k+1$ if none of the values in $\mathcal{M}_{2,k}$ appear at or after coordinate i of $\mathcal{M}_{2k}$). By definition of $\mathcal{M}_{2k}$ (as the merge of $\mathcal{M}_{1,k}$ and $\mathcal{M}_{2,k}$), we have that $x_i + y_i = i + 1$. By (7) above, we have that there are at least $\max(0, 1 + (k - x_i) \cdot \frac{n}{k})$ elements in L_1 that are greater than or equal to u_x, and since u_x has index at least i in $\mathcal{M}_{2k}$, we have that $u_x \geq w_i$, and hence (by monotonicity of GTE) there are at least $\max(0, 1 + (k - x_i) \cdot \frac{n}{k})$ elements in L_1 that are greater than or equal to w_i. Similarly, there are at least $(1 + k - y_i) \cdot \frac{n}{k}$ elements in L'_2 that are greater than or equal to w_i. Consequently, there are at least $1 + (1 + 2k - (x_i + y_i)) \cdot \frac{n}{k} = 1 + (2k - i) \cdot \frac{n}{k}$ values in L'_1 that are greater than or equal to w_i, as required. □

References

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An o(n log n) sorting network. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 1–9. ACM, 1983.

[Bat68] Kenneth E. Batcher. Sorting networks and their applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book Company, Washington D.C., 1968.

[BH85] Allan Borodin and John E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *J. Comput. Syst. Sci.*, 30(1):130–145, 1985.

[CHI+19] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Naoto Kiribuchi, and Benny Pinkas. An efficient secure three-party sorting protocol with an honest majority. *IACR Cryptol. ePrint Arch.*, page 695, 2019.

[FNO22] Brett Hemenway Falk, Rohit Nema, and Rafail Ostrovsky. A linear-time 2-party secure merge protocol. *IACR Cryptol. ePrint Arch.*, 2022.

[FO21] Brett Hemenway Falk and Rafail Ostrovsky. Secure merge with o (n log log n) secure operations. In *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[HICT14] Koki Hamada, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Oblivious radix sort: An efficient sorting algorithm for practical secure multi-party computation. *IACR Cryptol. ePrint Arch.*, page 121, 2014.

[HKI+12] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Practically efficient multi-party sorting protocols from comparison sort algorithms. In *International Conference on Information Security and Cryptology*, pages 202–216. Springer, 2012.

[HNO98] Tatsuya Hayashi, Koji Nakano, and Stephan Olariu. Work-time optimal k-merge algorithms on the PRAM. *IEEE Trans. Parallel Distributed Syst.*, 9(3):275–282, 1998.

[HS82] Zhu Hong and Robert Sedgewick. Notes on merging networks (preliminary version). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 296–302. ACM, 1982.

[LWZ11] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-efficient oblivious database manipulation. In Xuejia Lai, Jianying Zhou, and Hui Li, editors, *Information Security, 14th International Conference, ISC 2011, Xi'an, China, October 26-29, 2011. Proceedings*, volume 7001 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2011.

[MSS08] Kurt Mehlhorn, Peter Sanders, and Peter Sanders. *Algorithms and data structures: The basic toolbox*, volume 55. Springer, 2008.

[NO07] Takashi Nishide and Kazuo Ohta. Constant-round multiparty computation for interval test, equality test, and comparison. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 90-A(5):960–968, 2007.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 515–530. USENIX Association, 2015.

[Ski98] Steven S Skiena. *The algorithm design manual*, volume 2. Springer, 1998.

[Val75] Leslie G. Valiant. Parallelism in comparison problems. *SIAM J. Comput.*, 4(3):348–355, 1975.

[YY76] Andrew Chi-Chih Yao and Foong Frances Yao. Lower bounds on merging networks. *J. ACM*, 23(3):566–571, 1976.