Efficient Solutions to Variants of Inversion Problems of Range Minimum Queries

Souta Kobayashi[1], Dominik Köppl[2], Ryo Yoshinaka[1], and Ayumi Shinohara[1]

[1] Tohoku University, Japan
kobayashi.souta.t8@dc.tohoku.ac.jp,{ryoshinaka,ayumis}@tohoku.ac.jp
[2] University of Yamanashi, Japan, dkppl@yamanashi.ac.jp

Abstract. Given a set of results from range minimum queries (RMQs), our task is to construct a sequence that is consistent with the results of the queries. We study two types of RMQs: a *value-based* RMQ returns the minimum value and an *index-based* RMQ returns the index of the minimum. While the value-based version has been discussed informally in the context of competitive programming, the index-based version appears to be unexplored. In this paper, we provide a survey and unified analysis of the value-based version, and we propose efficient algorithms for the index-based version. These include algorithms for computing the lexicographically smallest consistent sequence and permutation, as well as an enumeration algorithm that outputs all consistent permutations with constant delay.

1 Problem Definition

Given an integer array $A[1:n]$ of n integers, a *range minimum query* (RMQ) asks, for a given interval $[l,r]$ to report the smallest entry in the sub-array $A[l:r]$, i.e., $\min_{k\in[l,r]} A[k]$. Literature gives two types of data structures that deal with this problem differently: encoding and indexing data structures. Encoding data structures [7,5,11,3], mostly based on the equivalence with the lowest common ancestor problem, need A only during their construction – at query time they can report the answer directly. The other type, the indexing data structures [15,9,10], needs to consult A even during query time. While the latter type seems to be weaker, it allows squeezing the constructed data structure in only a few bits of space. Indeed, indexing data structures that report only the index i of the minimum need $2n + o(n)$ bits of space, which was reached by Fischer and Heun [10]. In particular, when only the index of the minimum is needed, indexing data structures for RMQs are the way to go. Therefore, it has become natural to think about RMQs as either of two types: returning the value of the minimum or the index at which the minimum is stored. We call the former type *value-based* and the latter *index-based*.

Now suppose that instead of having A or an RMQ data structure on A, we are given some answers to either value-based or index-based RMQs and try to reverse-engineer A. We should find an array $A'[1:n]$ that is consistent with the original A, i.e., answering each of the RMQs on A or A' leads to the same results. While there can be infinitely many arrays satisfying the requirements, this paper focuses on reconstructing specific types of arrays as representatives: namely, lexicographically smallest arrays and permutations. Those variants of the value-based version of this problem have occasionally appeared in competitive programming contests [23,21,25,19,18], with several informal explanatory articles available online [24,22,20,26]. In contrast, to the best of the authors' knowledge, the index-based version has not been investigated.

Regarding the reverse engineering of index-based RMQs, we present linear-time algorithms for computing the smallest positive integer sequences and permutations, together with an algorithm that enumerates all consistent permutations with constant delay, in Section 3. Since index-based RMQs do not retain the actual array values, consistent permutations can be viewed as representing all consistent arrays without duplicated values. The value-based counterparts of these problems are discussed in Section 4. After surveying the existing results, we present a new linear-time algorithm for computing the smallest consistent permutation.

2 Preliminaries

Let $\mathbb{N}$ and $\mathbb{N}_+$ denote the sets of non-negative integers and positive integers, respectively. The interval of two non-negative integers l and r is denoted by $[l, r] = \{ i \in \mathbb{N} \mid l \leq i \leq r \}$. For a sequence $A = (a_1, \ldots, a_n)$ of integers, $A[i] = a_i$ and $A[i : j] = (a_i, \ldots, a_j)$ if $i \leq j$. In the case where $i > j$, $A[i : j]$ is the empty sequence. For a set J of indices of A, we write $A(J) = \{ A[i] \mid i \in J \}$. Thus, $A[i : j]$ is a sequence and $A([i, j])$ is a set. Since this paper is concerned with sequences of positive integers only, an *(integer) sequence* refers to a sequence of positive integers, unless otherwise noted. A sequence A of length n is *(lexicographically) smaller* than another B of the same length if there is $i \in [1, n]$ such that $A[1 : i-1] = B[1 : i-1]$ and $A[i] < B[i]$.

2.1 Useful queries and data structures

Our algorithms need to solve two types of problems, for which we use data structures from literature.

Interval Union-Find Problem
Consider manipulating an interval partition $\mathcal{I}$ of the set $[1, n]$. Initially, $\mathcal{I}$ is the discrete partition $\mathcal{I} = \{ [i, i] \mid i \in [1, n] \}$. We allow two kinds of operations:

1. $\mathcal{I}.link(i)$ with $i \in [1, n]$ merges the intervals $[l, i], [i+1, r] \in \mathcal{I}$ into $[l, r]$ if such intervals exist in $\mathcal{I}$;
2. $\mathcal{I}.find(i)$ with $i \in [1, n]$ returns $r \in [1, n]$ such that $i \in [l, r] \in \mathcal{I}$.

The above is a specialization of the union-find problem commonly studied in the literature. Our specialization allows a more efficient solution.

Theorem 1 ([16,12]). *One can perform m operations of link and find over $[1, n]$ given in an online manner in $O(n + m \cdot \alpha(m + n, n))$ time, where α is the inverse of Ackermann's function. Moreover, if the machine word size is $\Omega(\log n)$ bits, the factor $\alpha(m + n, n)$ can be replaced by a constant.*

Here, by an *online manner*, we mean that we perform each operation without knowing the succeeding operations.

Offline Minimum Problem
Consider manipulating a semi-dynamic set $S \subseteq [1, n]$ of integers. Initially, S is empty ($S = \emptyset$). We allow two kinds of operations:

1. $S.insert(i)$ with $i \in [1, n]$ updates the set S to $S \cup \{i\}$;
2. $S.extract()$ returns and deletes the minimum of S, unless S is empty.

Theorem 2 ([1,12]). *One can perform (at most $2n$) operations of insert and extract over $[1, n]$ given in an offline manner in $O(n \cdot \alpha(n, n))$ time if insert(i) is called at most once for each $i \in [1, n]$. Moreover, if the machine word size is $\Omega(\log n)$ bits, the factor $\alpha(n, n)$ can be replaced by a constant.*

Here, by an *offline manner*, we mean that we are given the sequence of operations before processing the first operation.

3 Index-based RMQs

This section is concerned with integer sequences consistent with given index-based RMQs and their answers. An *index-based RMQ with answer (iRMQA)* is a triple of positive integers (l, r, k) such that $l, r, k \in [1, n]$ with $l \leq k \leq r$ for some fixed n. We formally define our problem as follows.

iRMQA Reversal Problem
Input: a finite set $Q \subseteq \{\, (l, r, k) \in \mathbb{N}_+^3 \mid 1 \le l \le k \le r \le n \,\}$ of iRMQAs;
Output: a sequence $A[1:n]$ such that $\min\{\, i \in [l, r] \mid A[i] = \min A[l:r] \,\} = k$ for all $(l, r, k) \in Q$ or a report that no such sequence exists.

In this model, when there are multiple indices that achieve the minimum value within the queried range, the query result is the smallest such index.

Throughout this section, we fix the length n of the underlying sequences and a finite set Q of iRMQAs, and let $q = |Q|$.

Example 1. Given $n = 9$ and $Q_1 = \{(1,2,1), (1,3,3), (4,7,5), (6,8,8), (5,9,9)\}$, one of the solutions is $B = (7,7,5,9,5,5,5,3,1)$. In fact,

$$\begin{aligned}
&\min\{\, i \in [1,2] \mid B[i] = \min B[1:2] = 7 \,\} = \min\{1,2\} = 1,\\
&\min\{\, i \in [1,3] \mid B[i] = \min B[1:3] = 5 \,\} = \min\{3\} = 3,\\
&\min\{\, i \in [4,7] \mid B[i] = \min B[4:7] = 5 \,\} = \min\{5,6,7\} = 5,\\
&\min\{\, i \in [6,8] \mid B[i] = \min B[6:8] = 3 \,\} = \min\{8\} = 8,\\
&\min\{\, i \in [5,9] \mid B[i] = \min B[5:9] = 1 \,\} = \min\{9\} = 9.
\end{aligned}$$

If two iRMQAs (l, r, k) and (l', r', k) with the same k are given, they can be merged into $(\min(l, l'), \max(r, r'), k)$ without changing the admissible consistent sequences. We call an iRMQA set *canonical* if it has just one iRMQA of the form (l_k, r_k, k) for each k. One can convert Q into canonical form by letting

$$\begin{aligned}
l_k &= \min(\{k\} \cup \{\, l \mid (l, r, k) \in Q \text{ for some } r \,\}),\\
r_k &= \max(\{k\} \cup \{\, r \mid (l, r, k) \in Q \text{ for some } l \,\}).
\end{aligned}$$

We store those values in two arrays of length n by a preprocessing taking $O(q+n)$ time. Hereafter, we assume that the given iRMQA set is canonical.

3.1 Permutations consistent with iRMQAs

This subsection is concerned with the problems of enumerating all permutations and finding the smallest permutation consistent with the given iRMQA set Q.

We represent the conditions defined by the iRMQAs of Q with a directed graph $G_Q = ([1,n], E_Q)$, where

$$E_Q = \{\, (i, j) \mid i \neq j \text{ and } l_i \le j \le r_i \text{ for } (l_i, r_i, i) \in Q \,\}.$$

Proposition 1. *A permutation P is consistent with an iRMQA set Q if and only if its inverse permutation P^{-1} is a topological ordering of the graph G_Q.*

Thus, finding a permutation consistent with the given iRMQAs is equivalent to finding a topological sort on the graph G_Q. In particular, Q has a consistent permutation if and only if G_Q is acyclic. However, constructing the graph G_Q can take quadratic time. Instead, we compute a subgraph G'_Q, which has less than $2n$ edges and retains the same admissible topological orderings as G_Q. We first observe some properties of the graph G_Q. Let $\widehat{E}_Q$ be the transitive closure of E_Q; that is, $(i, j) \in \widehat{E}_Q$ if and only if there is a path from i to j in G_Q.

Lemma 1. *If $i < j < k$ and $(i, k) \in E_Q$, then $(i, j) \in E_Q$. As a corollary, if $i < j < k$ and $(i, k) \in \widehat{E}_Q$, then $(i, j) \in \widehat{E}_Q$. The same holds when $i > j > k$.*

We call an edge (i, j) *leftward* if $i > j$ and *rightward* if $i < j$. Lemma 1 implies that if we have $(i, k), (j, k) \in E_Q$ with $i < j < k$, deletion of (i, k) does not affect the reachability from i to k since $(i, j) \in E_Q$. Therefore, it is enough to keep the closest predecessors $par_{\mathsf{L}}[k]$ and $par_{\mathsf{R}}[k]$ on

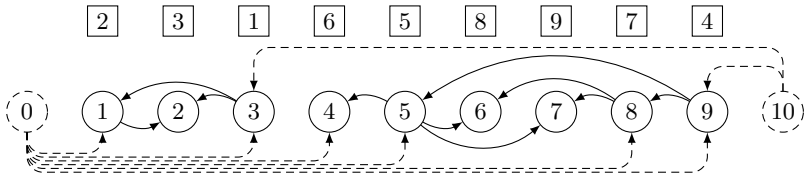

Fig. 1. Graph G'_{Q_1} for Q_1 in Example 1. Edges of E_L and E_R are drawn by solid leftward and rightward arrows, respectively. The vertex 10 and its outgoing edges drawn with dashed lines appear in T_L only. Similarly, 0 appears in T_R. The boxed numbers show the preorder numbering in left-to-right depth-first search of the tree T_L, which gives the smallest permutation consistent with Q_1.

the left and the right, respectively, for each vertex k. We call $par_\mathsf{L}[k]$ and $par_\mathsf{R}[k]$, if they exist, the *left* and the *right* parents of k, respectively. Formally, define $G'_Q = ([1,n], E'_Q)$ whose edge set $E'_Q \subseteq E_Q$ keeps only edges of the forms $(par_\mathsf{L}[k], k)$ and $(par_\mathsf{R}[k], k)$ where

$$par_\mathsf{L}[k] = \max(\{\, i \mid i < k \text{ and } (i,k) \in E_Q \,\} \cup \{0\}),$$
$$par_\mathsf{R}[k] = \min(\{\, i \mid k < i \text{ and } (i,k) \in E_Q \,\} \cup \{n+1\}).$$

If k has no left (resp. right) predecessors, $par_\mathsf{L}[k]$ (resp. $par_\mathsf{R}[k]$) is the dummy value 0 (resp. $n+1$). Let us partition E'_Q into E_L and E_R that consist of leftward edges of the form $(par_\mathsf{R}[k], k)$ and rightward edges of the form $(par_\mathsf{L}[k], k)$, respectively. The edge sets E_L and E_R respectively form forests. Let $\widehat{E}_\mathsf{L}$ and $\widehat{E}_\mathsf{R}$ be the transitive closures of E_L and E_R, respectively.

Lemma 2. *For $(i,j) \in E_Q$, if $i > j$, then $(i,j) \in \widehat{E}_\mathsf{L}$. If $i < j$, then $(i,j) \in \widehat{E}_\mathsf{R}$.*

Therefore, the graph G'_Q admits exactly the same topological orderings as G_Q. Since $|E'_Q| = |E_\mathsf{L}| + |E_\mathsf{R}| \leq 2(n-1)$, the size of E'_Q is less than $2n$.

Algorithm 1 computes the right parent $par_\mathsf{R}[k]$ and the leftmost reachable vertex $reach_\mathsf{L}[k] = \min(\{\, j \mid (j,k) \in \widehat{E}_\mathsf{L} \,\} \cup \{k\})$ for each k. The array par_R can be seen as a tree with the dummy root $n+1$, where we set $l_{n+1} = 1$. That is, we claim that Algorithm 1 constructs the tree $T_\mathsf{L} = ([1, n+1], E^+_\mathsf{L})$ where

$$E^+_\mathsf{L} = E_\mathsf{L} \cup \{\, (n+1, k) \mid k \text{ has no incoming edges in } E_\mathsf{L} \,\}.$$

The function `subtree(`j`)` recursively constructs the subtree rooted at j in the right-to-left depth-first search manner and returns the leftmost descendant minus one. Figure 1 illustrates an example of G'_Q together with T_L and T_R where T_R is defined symmetrically to T_L.

Lemma 3. *Algorithm 1 computes the right parent $par_\mathsf{R}[i]$ and the leftmost reachable vertex $reach_\mathsf{L}[i]$ of all i in $O(n)$ time.*

Computation of $par_\mathsf{L}[k]$ is symmetric to Algorithm 1, so we can construct the graph G'_Q in linear time by the same techniques. If the graph G'_Q contains a cycle, which can be decided in time linear in the graph size, Q admits no consistent permutations. Suppose G'_Q is acyclic. Ono and Nakano [14] have shown that we can enumerate all linear extensions of a given poset with constant delay time. The edge set E'_Q itself is not a poset, but one can recognize its transitive closure in constant time, since $(i,j) \in \widehat{E}_Q$ if and only if $j \in [reach_\mathsf{L}[i], reach_\mathsf{R}[i]]$, where $reach_\mathsf{R}$ is defined and computed in the way exactly symmetric to $reach_\mathsf{L}$.

Theorem 3. *One can enumerate all permutations consistent with q given iRMQAs with constant delay time after processing the input in $O(q+n)$ time.*

Now we turn our attention to finding the smallest consistent permutation. We assume G_Q has no cycle. The left-to-right depth-first search of the tree T_L finds the smallest consistent permutation. This might appear surprising, because not every topological sort on T_L gives a topological sort on G'_Q. However, the numbering based on the "left-to-right" depth-first search on T_L always respects the "rightward" edges of E_R, as long as G'_Q has no cycle.

Algorithm 1: Computing the tree T_{L}

Input: Canonical iRMQA set $Q = \{ (l_k, r_k, k) \mid k \in [1, n] \}$
Output: arrays par_{R} and $reach_{\mathsf{L}}$ such that $par_{\mathsf{R}}[i]$ is the smallest j satisfying $l_j \leq i < j$ and $reach_{\mathsf{L}}[i]$ is the leftmost reachable vertex from i

```
1  begin
2      par_R, reach_L ← arrays of length n;
3      subtree(n + 1);
4      return par_R and reach_L;
5  function subtree(j):
6      i ← j − 1;
7      while i ≥ l_j do
8          par_R[i] ← j;
9          i ← subtree(i);
10     reach_L[j] ← i + 1;
11     return i;
```

Lemma 4. *The preorder numbering P_0 in left-to-right depth-first search of the tree T_{L} gives the smallest permutation consistent with Q, where the dummy vertex $n+1$ is numbered 0.*

Proof. We note the fact that if $i < j$ and j is visited before i, then j is an ancestor of i in T_{L}.

We first show the consistency of P_0, i.e., for any edge (i, j) of G'_Q, it holds $P_0[i] < P_0[j]$. If $(i, j) \in E_{\mathsf{L}}$, obviously $P_0[i] < P_0[j]$. Suppose $(i, j) \in E_{\mathsf{R}}$. Since G'_Q has no cycle, $(j, i) \notin \widehat{E}_{\mathsf{L}}$. Thus, j is not an ancestor of i in T_{L}. Since $i < j$, we visit i before j, i.e., $P_0[i] < P_0[j]$. Therefore, P_0 is consistent with Q.

To establish the minimality of P_0, we show that if a permutation P' is consistent with Q and $P'[1 : i-1] = P_0[1 : i-1]$, then $P'[i] \geq P_0[i]$. The vertices visited before i belong to either

$$J_i = \{ j \in [i+1, n] \mid j \text{ is an ancestor of } i \} \text{ or}$$
$$K_i = \{ j \in [1, i-1] \mid j \text{ is not a descendant of } i \}.$$

That is, $P_0(J_i \cup K_i) = \{1, \ldots, P_0[i] - 1\}$. The assumption $P'[1 : i-1] = P_0[1 : i-1]$ immediately implies $P'(K_i) = P_0(K_i)$ by $K_i \subseteq [1, i-1]$. Thus,

$$P'(J_i) \subseteq [1, n] \setminus P'(K_i) = [1, n] \setminus P_0(K_i) = [P_0[i], n] \cup P_0(J_i).$$

If $P'(J_i) = P_0(J_i)$, then $P'[i] \in [P_0[i], n]$, so $P'[i] \geq P_0[i]$. If $P'(J_i) \neq P_0(J_i)$, there is $j \in J_i$ such that $P'[j] \in [P_0[i], n]$, i.e., $P'[j] \geq P_0[i]$. Since P' is consistent with Q, we have $P'[i] > P'[j] \geq P_0[i]$. □

Corollary 1 (Smallest permutation for iRMQAs). *One can find the smallest permutation of length n consistent with q given iRMQAs in $O(q + n)$ time.*

3.2 Smallest sequence consistent with iRMQAs

In this subsection, we discuss finding the smallest integer sequence consistent with the given iRMQAs. Similarly to the case of permutations, the conditions defined by the iRMQAs can be represented by the graph $G_Q = ([1, n], E_Q)$ defined in Subsection 3.1 with an edge weighting $w \colon E_Q \to \{0, 1\}$ defined by

$$w((i, j)) = \begin{cases} 0 & \text{if } i < j, \\ 1 & \text{if } i > j. \end{cases}$$

Recall that for any sequence A consistent with Q, if $(l, r, k) \in Q$, then $A[k] < A[i]$ for all $i \in [l, k-1]$ and $A[k] \leq A[i]$ for all $i \in [k+1, r]$. Thus, A is consistent with Q if and only if for all $(i, j) \in E_Q$,

$$A[i] + w((i, j)) \leq A[j]. \tag{1}$$

The *length of a path* is the sum of the weights of the constituting edges.

Lemma 5. *Q admits a consistent sequence if and only if G_Q is acyclic.*

Hereafter, we write $w(i,j)$ rather than $w((i,j))$ for readability, and assume that G_Q is acyclic. We define a graph $\widetilde{G}_Q$ by adding a vertex $n+1$ to G_Q and edges $(n+1,i)$ for all $i \in [1,n]$, with $w(n+1,i) = 1$ for all $i \in [1,n]$. We then define an array $\mathit{MaxDist}_{G_Q}$ of length n so that $\mathit{MaxDist}_{G_Q}[i]$ is the maximum length of a path from vertex $n+1$ to i in $\widetilde{G}_Q$.

Lemma 6. *$\mathit{MaxDist}_{G_Q}$ is the smallest integer sequence consistent with Q.*

Similarly to the case of permutations in Subsection 3.1, we use a smaller graph for computing $\mathit{MaxDist}_{G_Q}$.

Lemma 7. *Let T_{L} be the graph defined in Subsection 3.1. Then, $\mathit{MaxDist}_{G_Q}[i]$ coincides with the depth of i in T_{L} for any vertex $i \in [1,n]$.*

Proof. If a longest path p from $n+1$ to i in E_Q consists of leftward edges only, Lemma 2 implies that there is a path p' in E_{L} that is at least as long as p. Therefore, it suffices to show that for any path from $n+1$ to $i \in [1,n]$ in E_Q, there is a path of the same weight consisting of leftward edges only.

Suppose a path p from $n+1$ in E_Q includes a rightward edge. Since the first edge must be leftward, the first rightward edge (j,k) in p must follow a leftward edge $(i,j) \in E_Q$ with $j < i$ and $j < k$. If $i < k$, then $(j,k) \in E_Q$ implies $(j,i) \in E_Q$. We have a cycle of i and j. So, $i > k$. By Lemma 1, E_Q has a leftward edge (i,k). Replacing the sub-path (i,j,k) with (i,k) in p does not change the length. □

Concerning the iRMQA set Q_1 in Example 1, the depth list of the vertices $1,\ldots,n$ of the tree T_{L} is $(2,2,1,3,2,3,3,2,1)$ (see Fig. 1). This is the smallest sequence consistent with Q_1.

Having already discussed linear-time construction of T_{L} (Algorithm 1), we obtain the following theorem.

Theorem 4. *We can compute the smallest sequence of length n consistent with q given iRMQAs in $O(q+n)$ time.*

4 Value-based RMQs

This section is concerned with integer sequences consistent with given value-based RMQs and their answers. A *value-based RMQ with answer (vRMQA)* is a triple (l,r,v) of positive integers such that $l,r \in [1,n]$ with $l \le r$ and $v \in \mathbb{N}_+$ for some fixed n. We formally define our problem as follows.

vRMQA Reversal Problem
Input: a finite set $Q \subseteq \{\, (l,r,v) \in \mathbb{N}_+^3 \mid 1 \le l \le r \le n \,\}$ of vRMQAs;
Output: a positive integer sequence $A[1:n]$ such that $\min A[l:r] = v$ for all $(l,r,v) \in Q$ or a report that no such sequence exists.

Throughout this section, we fix the length n of the underlying sequences and a finite set Q of vRMQAs and let $q = |Q|$.

Example 2. Given $n = 9$ and $Q_2 = \{(1,5,3),(2,3,6),(4,6,3),(6,8,1)\}$, the sequence $A = (3,9,6,8,9,3,5,1,2)$ is one of the solutions. In fact,

$$\min A[1:5] = \min(3,9,6,8,9) = 3\,, \qquad \min A[2:3] = \min(9,6) = 6\,,$$
$$\min A[4:6] = \min(8,9,3) = 3\,, \qquad \min A[6:8] = \min(3,5,1) = 1\,.$$

4.1 Smallest sequence consistent with vRMQAs

The problem of constructing a sequence consistent with a given set of vRMQAs has been previously addressed in competitive programming contests like [19,18]. This subsection reviews the editorial of [19] on this problem.

We call a sequence A *consistent with* a vRMQA set Q if, for every $(l, r, v) \in Q$,

$$\forall i \in [l, r],\ A[i] \geq v, \tag{2}$$

$$\exists i \in [l, r],\ A[i] = v. \tag{3}$$

We refer to the first condition (2) as a *lower-bound condition (LBC)* and denote it by $\mathsf{LB}(l, r, v)$. Similarly, the second condition (3) is referred to as an *existential condition (EXC)* and is denoted by $\mathsf{EX}(l, r, v)$.

We define the *lower-bound array* LB of length n by

$$LB[i] = \max\left(\{1\} \cup \{\, v \mid \exists (l, r, v) \in Q,\ i \in [l, r] \,\}\right), \tag{4}$$

which represents the logical conjunction of the LBCs on each position i.

For the instance of Example 2, we have

$$LB = (3, 6, 6, 3, 3, 3, 1, 1, 1)\,.$$

Proposition 2. *If Q admits consistent integer sequences, LB is the smallest such sequence.*

Theorem 5 ([19]). *In $O(n + q \log n)$ time, one can decide whether given vRMQAs admit a consistent sequence and if so, compute the smallest consistent sequence.*

Proof. The array LB can be constructed in $O(n + q \log n)$ time by employing a segment tree with range updates and point queries[3] as follows. For each vRMQA $(l, r, v) \in Q$ (the order is not of importance), we (lazily) update $LB[i]$ for all $i \in [l, r]$ to be v if v is larger than the current value. Furthermore, the EXCs can also be verified in $O(n + q \log n)$ time using a segment tree, with which we can verify $\min LB[l : r] = v$ for each $(l, r, v) \in Q$. Therefore, the problem can be solved in $O(n + q \log n)$ time in total. □

When the maximum value v of $(l, r, v) \in Q$ is $O(n)$, we can compute LB more efficiently using the union-find structure like Algorithm 2 shown later.

4.2 Permutations consistent with vRMQAs

This subsection discusses counting the permutations and finding the smallest permutation consistent with the given vRMQA set. These problems were previously addressed, respectively, at Nordic Olympiad in Informatics 2018 [21] and Singapore National Olympiad in Informatics 2017 [23]. In addition, a variant problem was addressed at USA Computing Olympiad 2008 [25]. Several unofficial web articles [22,24,26] explain these problems, but they are informal and do not necessarily pursue minimizing the theoretical time complexity. In this subsection, we consolidate these ideas together with our own, and present a formal and comprehensive analysis.

Since we are concerned with permutations, we assume $v \in [1, n]$ for all $(l, r, v) \in Q$. In addition, since there must be a position with value 1 in any permutation, we may assume without loss of generality $(1, n, 1) \in Q$. Furthermore, if there exist two vRMQAs $(l_1, r_1, v), (l_2, r_2, v) \in Q$ with the same value $v \in [1, n]$, since any permutation P over $[1, n]$ has just one position i such that $P[i] = v$, those intervals $[l_1, r_1]$ and $[l_2, r_2]$ must overlap in order to admit a consistent permutation. So, the following two facts hold:

- P satisfies both $\mathsf{LB}(l_1, r_1, v)$ and $\mathsf{LB}(l_2, r_2, v)$ if and only if it satisfies $\mathsf{LB}(\min(l_1, l_2), \max(r_1, r_2), v)$;

[3] For details of segment trees, see, e.g., [2, Section 3.2.5] or [6, Section 10.3].

– P satisfies both $\mathsf{EX}(l_1, r_1, v)$ and $\mathsf{EX}(l_2, r_2, v)$ if and only if it satisfies $\mathsf{EX}(\max(l_1, l_2), \min(r_1, r_2), v)$.

Therefore, the information represented in Q is equivalent to Q' defined as follows.

$$Q' = \{\, (l_v^{\mathsf{LB}}, r_v^{\mathsf{LB}}, v) \mid v \in V_Q \,\} \cup \{\, (l_v^{\mathsf{EX}}, r_v^{\mathsf{EX}}, v) \mid v \in V_Q \,\}, \text{ where}$$
$$l_v^{\mathsf{LB}} = \min\{\, l \in [1,n] \mid (l,r,v) \in Q \,\}, \quad r_v^{\mathsf{LB}} = \max\{\, r \in [1,n] \mid (l,r,v) \in Q \,\},$$
$$l_v^{\mathsf{EX}} = \max\{\, l \in [1,n] \mid (l,r,v) \in Q \,\}, \quad r_v^{\mathsf{EX}} = \min\{\, r \in [1,n] \mid (l,r,v) \in Q \,\},$$

and $V_Q = \{\, v \in [1,n] \mid (l,r,v) \in Q \text{ for some } l, r \,\}$ is the set of values that appear in Q. We say that Q' is the *canonical form* of Q. From now on, we assume Q is given in its canonical form and one can obtain $l_v^{\mathsf{LB}}, r_v^{\mathsf{LB}}, l_v^{\mathsf{EX}}, r_v^{\mathsf{EX}}$ in constant time for each $v \in V_Q$. Just by scanning the vRMQA set Q once, we can store those values in four arrays of size n in $O(q+n)$ time. We note that $1 \in V_Q$, $l_1^{\mathsf{LB}} = 1$, and $r_1^{\mathsf{LB}} = n$ by the assumption.

Let $I_v^{\mathsf{LB}} = [l_v^{\mathsf{LB}}, r_v^{\mathsf{LB}}]$ and $I_v^{\mathsf{EX}} = [l_v^{\mathsf{EX}}, r_v^{\mathsf{EX}}]$. Then, a permutation P is consistent with Q if and only if for all $v \in V_Q$,

$$\forall i \in I_v^{\mathsf{LB}},\ P[i] \geq v, \quad \text{and} \quad \exists i \in I_v^{\mathsf{EX}},\ P[i] = v.$$

As in Eq. 4 in Section 4.1, we define the lower-bound array LB accordingly. Using $I_v^{\mathsf{LB}} = [l_v^{\mathsf{LB}}, r_v^{\mathsf{LB}}]$, the array is expressed as $LB[i] = \max\{\, v \in V_Q \mid i \in I_v^{\mathsf{LB}} \,\}$.

Example 3. For the instance Q_2 in Example 2, we have $V_{Q_2} = \{1, 3, 6\}$ and

$$I_1^{\mathsf{LB}} = [1,9],\ I_3^{\mathsf{LB}} = [1,6],\ I_6^{\mathsf{LB}} = [2,3],\ I_1^{\mathsf{EX}} = [6,8],\ I_3^{\mathsf{EX}} = [4,5],\ I_6^{\mathsf{EX}} = [2,3].$$

Taking advantage of the restriction that $v \in [1,n]$ for any $(l,r,v) \in Q$, Algorithm 2 computes the array LB more efficiently than the method in the proof of Theorem 5. The array LB is initialized with ones. In the **for** loop, decrementing v from n to 2, we set $LB[i] = v$ for all $i \in I_v^{\mathsf{LB}}$ unless $LB[i]$ has already been updated (i.e., $LB[i] \neq 1$). This procedure gives the correct lower-bound array. If we check whether $LB[i] = 1$ for every $i \in I_v^{\mathsf{LB}}$, the computation costs quadratic time, but we can accelerate this by using Theorem 1, following the approach of [26]. We partition the set $[1, n+1]$ into intervals with the invariant property that $LB[j] = 1$ if and only if j is the right end of an interval for any $j \in [1, n+1]$, assuming $LB[n+1] = 1$. This allows us to skip consecutive indices where the values of LB are not 1 and to find the least $j \geq i$ such that $LB[j] = 1$ by *find*(i). This guarantees that LB is updated at most once on each index i and the total number of iterations of the **while** loop is bounded by n. Therefore, Algorithm 2 computes the lower-bound array LB in $O(n \cdot \alpha(n,n))$ time. It will be $O(n)$ time if the machine word size is $\Omega(\log n)$ bits.

Now, we determine all valid positions of values $v \in [1,n]$ in a consistent permutation using LB. Define

$$X_v = \{\, i \in [1,n] \mid LB[i] = v \,\},$$
$$X_{\leq v} = \textstyle\bigcup_{u=1}^{v} X_u\,,$$
$$X_v^{\mathsf{EX}} = X_v \cap I_v^{\mathsf{EX}} \text{ for } v \in V_Q\,.$$

Note that $X_v = \emptyset$ for $v \notin V_Q$ and $X_v^{\mathsf{EX}} \subseteq X_{\leq v}$ for $v \in V_Q$.

Lemma 8. *A permutation P is consistent with Q if and only if*

$$P^{-1}[v] \in \begin{cases} X_v^{\mathsf{EX}} & \text{for all } v \in V_Q, \\ X_{\leq v} & \text{for all } v \notin V_Q. \end{cases}$$

Algorithm 2: Computing the lower bound array

Input: Canonical vRMQA set $Q = \{ (l_v^{\mathsf{LB}}, r_v^{\mathsf{LB}}, v), (l_v^{\mathsf{EX}}, r_v^{\mathsf{EX}}, v) \mid v \in V_Q \}$
Output: The lower-bound array LB

```
1  begin
2      LB ← an array of length n initialized with ones;
3      I ← the discrete partition of [1, n + 1];
4      for v = n, n − 1, ..., 2 do
5          if v ∈ V_Q then
6              i ← I.find(l_v^LB);
7              while i ≤ r_v^LB do
8                  LB[i] ← v;
9                  I.link(i);
10                 i ← I.find(i);
11     return LB;
```

Counting permutations consistent with vRMQAs A strategy for counting the number of permutations consistent with Q is presented in [22]. We determine $P^{-1}[v]$ for $v = 1, \ldots, n$ in this order based on Lemma 8. If $v \in V_Q$, we have $|X_v^{\mathsf{EX}}|$ choices, since none has been chosen from X_v^{EX} earlier. If $v \notin V_Q$, we have $|X_{\leq v}| - (v-1)$ choices, since we have already picked $v-1$ from $X_{\leq v}$. Therefore, the number C_Q of consistent permutations is

$$C_Q = \prod_{v \in [1,n]} \max(0, c_v) \text{ where } c_v = \begin{cases} |X_v^{\mathsf{EX}}| & \text{if } v \in V_Q, \\ |X_{\leq v}| - v + 1 & \text{if } v \notin V_Q. \end{cases} \tag{5}$$

Counting $|X_v^{\mathsf{EX}}|$ and $|X_{\leq v}|$ is easy using LB and I_v^{EX}.

Example 4. For the instance Q_2 in Example 2, we have

$$X_1 = \{7,8,9\}, \quad X_3 = \{1,4,5,6\}, \quad X_6 = \{2,3\},$$
$$X_1^{\mathsf{EX}} = \{7,8\}, \quad X_3^{\mathsf{EX}} = \{4,5\}, \quad X_6^{\mathsf{EX}} = \{2,3\},$$

and

$$C_{Q_2} = 2 \cdot (3-1) \cdot 2 \cdot (7-3) \cdot (7-4) \cdot 2 \cdot (9-6) \cdot (9-7) \cdot (9-8) = 1152\,.$$

Theorem 6. *We can count the number of permutations of length n consistent with q given vRMQAs in $O(q + n \cdot \alpha(n,n))$ time. Moreover, if the machine word size is $\Omega(\log n)$ bits, the factor $\alpha(n,n)$ can be replaced by a constant.*

By expanding the counting arguments above, we can rank all consistent permutations so that, given a rank r in $[1, C_Q]$, we can return the unique consistent permutation with rank r in $O(n)$ time, and vice versa. This immediately yields an enumeration of consistent permutations with an $O(n)$-time delay. We remark that by applying Uno's technique [17], it is also possible to enumerate all consistent permutations with an $O(\log n)$-time delay after $O(n^{2.5})$ time preprocessing.

Smallest permutation consistent with vRMQAs Assuming that Q admits at least one consistent permutation ($C_Q \neq 0$), we compute the lexicographically smallest such permutation. The strategy in [24] greedily assigns each $v \in V_Q$ to the earliest position satisfying the conditions of Lemma 8. Formally, we define a permutation P_0 via its inverse $P_0^{-1}[v]$ for $v = 1, \ldots, n$ in this order by

$$P_0^{-1}[v] = \min Y_v \text{ for } Y_v = \begin{cases} X_v^{\mathsf{EX}} & \text{if } v \in V_Q, \\ X_{\leq v} \setminus P_0^{-1}([1, v-1]) & \text{if } v \notin V_Q. \end{cases} \tag{6}$$

By the counting argument for Theorem 6, Y_v is never empty and $\min Y_v$ differs from any value of $P_0^{-1}[1 : v-1]$. Thus, P_0 is well-defined and consistent with Q.

Algorithm 3: Computing the smallest consistent permutation

Input: Canonical vRMQA set Q and the lower-bound array LB
Output: The smallest consistent permutation P_0

```
1  begin
2      P_0, F ← arrays of length n initialized with zeros;
3      Z_v ← ∅ for all v ∈ V_Q;
4      for i = 1, 2, ..., n do
5          v ← LB[i];
6          if i ∈ I_v^EX and F[v] = 0 then P_0[i] ← v and F[v] ← 1;
7          else add i to Z_v;
8      S ← ∅;
9      for v = 1, 2, ..., n do
10         if v ∈ V_Q then S.insert(i) for all i in Z_v;
11         else P_0[S.extract()] ← v;
12     return P_0;
```

Example 5. Let us compare the smallest consistent permutation P_0 and the lower bound array LB for the instance Q_2 in Example 2:

$$P_0 = (4, 6, 7, 3, 5, 8, 1, 2, 9)\,,$$
$$LB = (3, \underline{6, 6}, \underline{3, 3}, 3, \underline{1, 1}, 1)\,,$$

where the positions in X_v^{EX} for $v \in V_{Q_2}$ are underlined in LB.

Lemma 9. *P_0 is the smallest permutation consistent with Q.*

Proof (Sketch). One can show that if a permutation P is consistent with Q and $P \neq P_0$, then the permutation obtained from P by swapping the values v and u is smaller than P and consistent with Q, where $v \in [1, n]$ is the smallest such that $P^{-1}[v] \neq P_0^{-1}[v]$ and $u = P[P_0^{-1}[v]]$. □

Algorithm 3 computes P_0 based on the definition (Eq. 6). While the underlying greedy logic follows [24], this is slightly modified to enable efficient computation, improving the $O(n \log n)$ time complexity to $O(n \cdot \alpha(n, n))$. The **for** loop of Line 4 computes $P_0^{-1}[v]$ for $v \in V_Q$ straightforwardly, where $F[v]$ indicates whether $P_0^{-1}[v]$ has already been computed. At the same time, we prepare the sets $Z_v = X_v \setminus \{P_0^{-1}[v]\}$ for all $v \in V_Q$. These sets store the positions where values greater than v and not belonging to V_Q can be placed, and are used in the latter half of the algorithm. We determine the positions $P_0^{-1}[v]$ of $v \notin V_Q$ in the **for** loop of Line 9. We perform the following operations on a semi-dynamic set S, which is initially empty, for each $v \in [1, n]$ in ascending order:

1. if $v \in V_Q$, insert every index $i \in Z_v = X_v \setminus \{P_0^{-1}[v]\}$ into S,
2. if $v \notin V_Q$, set $P_0[\min S] = v$ and remove $\min S$ from S.

These operations maintain the loop invariant that after the vth iteration of the second **for** loop, we have $S = X_{\leq v} \setminus P_0^{-1}([1, v])$. Thus, our construction is faithful to the definition of P_0 (Eq. 6). Since we can fix those operations over S before Line 8, we can apply Theorem 2.

Theorem 7. *We can compute the smallest permutation of length n consistent with the q given vRMQAs in $O(q + n \cdot \alpha(n, n))$ time. Moreover, if the machine word size is $\Omega(\log n)$ bits, the factor $\alpha(n, n)$ can be replaced by a constant.*

5 Concluding Remarks

We presented efficient algorithms for reconstructing sequences and permutations consistent with RMQ answers, addressing various natural variations such as computing the lexicographically smallest solution, enumerating all consistent permutations, and counting their total number, for both value-based and index-based settings.

While we showed that counting consistent permutations is tractable for vRMQAs, the complexity of the analogous counting problem for iRMQAs is still open. Counting the number of permutations consistent with a given iRMQA set Q is equivalent to counting that of linear extensions of the poset generated by the directed acyclic graph G'_Q in Section 3. Whilst counting the number of linear extensions of a given poset is #P-complete in general [8], some special cases allow polynomial-time counting (e.g., [13,4]). Since the graph G'_Q is highly restrictive, it might be possible to design a polynomial-time algorithm for counting the consistent permutations.

It is also interesting future work to study the inversion problems of other types of range queries, such as range sum queries and range mode queries.

Acknowledgment The authors are grateful to the anonymous reviewers for their helpful comments. In particular, we thank the reviewer who pointed out the connection to competitive programming, which encouraged us to examine the existing online results in depth and made us aware that some of the results in our submission were not entirely original. This work was supported in part by JSPS KAKENHI Grant Numbers 25K21150, 23H04378 (DK), 23K11325, 24H00697, 24K14827, 25K14981 (RY), and 25K14979 (AS).

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley, Reading, Massachusetts (1974)
2. Alba, D.E., Leal, J.A.R.: Algorithms for Competitive Programming. dnd learning (2023), https://books.google.co.jp/books?id=Uhz50AEACAAJ
3. Alstrup, S., Gavoille, C., Kaplan, H., Rauhe, T.: Nearest common ancestors: A survey and a new algorithm for a distributed environment. Theory Comput. Syst. **37**(3), 441–456 (2004). https://doi.org/10.1007/S00224-004-1155-5
4. Atkinson, M.D.: On computing the number of linear extensions of a tree. Order **7**, 23–25 (1990). https://doi.org/10.1007/BF00383170
5. Bender, M.A., Farach-Colton, M., Pemmasani, G., Skiena, S., Sumazin, P.: Lowest common ancestors in trees and directed acyclic graphs. Journal of Algorithms **57**(2), 75–94 (2005). https://doi.org/10.1016/j.jalgor.2005.08.001
6. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer (2008), https://books.google.de/books?id=tkyG8W2163YC
7. Berkman, O., Vishkin, U.: Recursive star-tree parallel data structure. SIAM Journal on Computing **22**(2), 221–242 (1993). https://doi.org/10.1137/0222017
8. Brightwell, G., Winkler, P.: Counting linear extensions. Order **8**, 225–242 (1991). https://doi.org/10.1007/BF00383444
9. Fischer, J., Heun, V.: Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In: Lewenstein, M., Valiente, G. (eds.) Combinatorial Pattern Matching. pp. 36–48. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). https://doi.org/10.1007/11780441_5
10. Fischer, J., Heun, V.: A new succinct representation of RMQ-information and improvements in the enhanced suffix array. In: Chen, B., Paterson, M., Zhang, G. (eds.) Combinatorics, Algorithms, Probabilistic and Experimental Methodologies. pp. 459–470. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74450-4_41
11. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: DeMillo, R.A. (ed.) Proc. STOC. pp. 135–143. ACM (1984). https://doi.org/10.1145/800057.808675
12. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. Journal of Computer and System Sciences **30**(2), 209–221 (1985). https://doi.org/10.1016/0022-0000(85)90014-5
13. Möhring, R.H.: Computationally tractable classes of ordered sets. In: Rival, I. (ed.) Algorithms and Order, NATO ASI Series C: Mathematical and Physical Sciences, vol. 255, pp. 105–193. Springer, Dordrecht (1989). https://doi.org/10.1007/978-94-009-2639-4_4
14. Ono, A., Nakano, S.i.: Constant time generation of linear extensions. In: Liśkiewicz, M., Reischuk, R. (eds.) Fundamentals of Computation Theory. pp. 445–453. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11537311_39

15. Sadakane, K.: Succinct data structures for flexible text retrieval systems. J. Discrete Algorithms **5**(1), 12–22 (2007). https://doi.org/10.1016/J.JDA.2006.03.011
16. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. J. ACM **22**(2), 215–225 (1975). https://doi.org/10.1145/321879.321884
17. Uno, T.: A fast algorithm for enumerating bipartite perfect matchings. In: Eades, P., Takaoka, T. (eds.) Algorithms and Computation (ISAAC 2001). pp. 367–379. Springer Berlin Heidelberg, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-45678-3_32
18. Inverse range minimum query. https://eolymp.com/en/problems/3080, accessed: 2025-12-01
19. No.1675 Strange Minimum Query. https://yukicoder.me/problems/no/1675, accessed: 2025-11-27
20. No.1675 Strange Minimum Query kaisetsu (editorial). https://yukicoder.me/problems/no/1675/editorial, accessed: 2025-11-27
21. Nordic Olympiad in Informatics. https://nordic.progolymp.se/, accessed: 2025-11-27
22. xiaoliebao1115: P11505 [nordicoi 2018] mysterious array. https://www.luogu.com.cn/article/j9714zmg (2025), accessed: 2025-11-27
23. The Singapore National Olympiad in Informatics. https://noisg.comp.nus.edu.sg/, problem archive available at https://github.com/noisg/sg_noi_archive, Accessed: 2025-11-27
24. https://github.com/noisg/sg_noi_archive/blob/master/2017/solution_writeup/NOI2017_solutions.pdf, accessed: 2025-12-01
25. Haybale guessing. https://www.acmicpc.net/problem/6153, originally USACO 2008 January Contest, Gold Division; Accessed: 2025-11-27
26. [P2898][USACO08JAN] Haybale Guessing Haybale Guessing. https://www.programmersought.com/article/60757109923/, accessed: 2025-12-01