Encoding Nearest Larger Values

Michael Hoffmann[a], John Iacono[b], Patrick K. Nicholson[c], Rajeev Raman[a,]

[a] *University Road, Leicester, United Kingdom*
[b] *New York University, School of Engineering, 6 MetroTech Center, Brooklyn, New York, United States of America*
[c] *Bell Labs, Clyde House, Blanchardstown Business and Technology Park, Dublin, Ireland*

Abstract

In *nearest larger value (NLV)* problems, we are given an array $A[1..n]$ of distinct numbers, and need to preprocess A to answer queries of the following form: given any index $i \in [1, n]$, return a "nearest" index j such that $A[j] > A[i]$. We consider the variant where the values in A are distinct, and we wish to return an index j such that $A[j] > A[i]$ and $|j - i|$ is minimized, the *nondirectional NLV (NNLV)* problem. We consider NNLV in the *encoding* model, where the array A is deleted after preprocessing.

The NNLV encoding problem turns out to have an unexpectedly rich structure: the *effective entropy* (optimal space usage) of the problem depends crucially on details in the definition of the problem. Of particular interest is the tiebreaking rule: if there exist two nearest indices j_1, j_2 such that $A[j_1] > A[i]$ and $A[j_2] > A[i]$ and $|j_1 - i| = |j_2 - i|$, then which index should be returned? For the tiebreaking rule where the rightmost (i.e., largest index) is returned, we encode a path-compressed representation of the Cartesian tree that can answer all NNLV queries in $1.89997n + o(n)$ bits, and can answer queries in $O(1)$ time. An alternative approach, based on *forbidden patterns*, achieves the same space bound and query time, and (for a slightly different tiebreaking rule) achieves $1.81211n + o(n)$ bits. Finally, we develop a fast method of counting distinguish-

[☆] A preliminary version appeared in the Proceedings of the 26th Annual Symposium on Combinatorial Pattern Matching (CPM 2015).

Email addresses: `mh55@leicester.ac.uk` (Michael Hoffmann), `iacono@nyu.edu` (John Iacono), `pat.nicholson@nokia.com` (Patrick K. Nicholson), `r.raman@leicester.ac.uk` (Rajeev Raman)

able configurations for NNLV queries. Using this method, we prove a lower bound of $1.62309n - \Theta(1)$ bits of space for NNLV encodings for the tiebreaking rule where the rightmost index is returned.
Keywords: Data structures, encoding data structures, succinct data structures.

1. Introduction

Nearest Larger Value (NLV) problems have had a long and storied history. Given an array $A[1..n]$ of values, the objective is to preprocess A to answer queries of the general form: given an index i, report the index or indices nearest to i that contain values strictly larger that $A[i]$. If no such index exists, then $A[i]$ is the maximum element in A, and we return -1.

Berkman et al. [1] studied the parallel pre-processing for this problem and noted a number of applications, such as parenthesis matching and triangulating monotone polygons. The connection to string algorithms for both the data structuring and the pre-processing variants of this problem is since well-established.

Since the definition of "nearest" is a bit ambiguous, we propose replacing it by one of the following options in order to fully specify the problem:

- *Unidirectionally nearest*: the solution is the index $j \in [1, i-1]$ such that $A[j] > A[i]$ and $i - j$ is minimized.

- *Bidirectionally nearest*: the solution consists of indices $j_1 \in [1, i-1]$ and $j_2 \in [i+1, n]$ such that $A[j_k] > A[i]$ and $|i - j_k|$ is minimized for $k \in \{1, 2\}$.

- *Nondirectionally nearest*: the solution is the index j such that $A[j] > A[i]$ and $|i - j|$ is minimized. As far as we are aware, this formulation has not been considered before.

Furthermore, the data structuring problem has different characteristics depending on whether we consider the elements of A to be distinct (Berkman et al. considered the undirectional variant when all elements in A are distinct).

We consider the problem in the *encoding* model, where once the data structure to answer queries has been created, the array A is deleted. Since it is not possible to reconstruct A from NLV queries on A, the *effective entropy* of NLV queries [2], the log (base 2) of the number of distinguishable NLV configurations, is very low and an NLV encoding of A can be much smaller than A itself. The encoding variant has several applications in space-efficient data structures for string processing, in situations where the values in A are intrinsically uninteresting. Results on encoding NLV problems include (all of the space bounds below are tight to within lower-order terms):

- The bidirectional NLV when A contains distinct values boils down essentially to encoding a Cartesian tree, through which route $2n + o(n)$-bit and $O(1)$-time data structures exist [3, 4].

- The unidirectional NLV when A contains non-distinct values can be encoded in $2n + o(n)$ bits and queries answered in $O(1)$ time [5, 6]. For the unidirectional NLV the bound is tight even when all values are distinct: we can perturb any instance of the unidirectional problem with non-distinct values in such as way as to preserve the solutions to all queries.[1]

- The bidirectional NLV for the case where elements in A need not be distinct was first studied by Fischer [7]. His data structure occupies $\lg(3 + 2\sqrt{2})n + o(n) \approx 2.544n + o(n)$ bits of space[2], and supports queries in $O(1)$ time.

In this paper, we consider the nondirectionally nearest larger value (NNLV) problem, in the case that all elements in A are distinct. The above results

[1]More details: given any array with non-distinct elements for the unidirectional problem, we first reduce the values of the elements to their ranks (allowing ties). We then tweak the values so that the rightmost of each duplicated value x is $x + \varepsilon$ for some $\varepsilon \in (0, 1)$. We then reduce ε by some positive amount such that it is remains positive, and repeat this step until all elements are distinct.

[2]We use $\lg x$ to denote $\log_2 x$.

already hint at the combinatorial complexity of NLV problems. However, the NNLV problem appears to be even richer, and the space bound appears not only to depend upon whether A is distinct or not, but also upon the specific tie-breaking rule to use if there are two equidistant nearest values to the query index i.

For instance, given a location i where there is a tie, we might always select the larger value to the right of location i to be its nearest larger value. We call this *rule I*. We give an illustration in the middle panel of Figure 1 (on page 8). Alternative tie breaking rules might be: to select the smallest of the two larger values (*rule II*), or to select the larger of the two larger values (*rule III*). Interestingly, it turns out that the tie breaking rule is important for the space bound. That is, if we count the number of distinguishable configurations of the NNLV problem for the various tie breaking rules, then we get significantly different answers. We counted the number of distinguishable configurations, for problem instances of size $n \in [1, 12]$, and got the sequences presented in Table 1.

Unfortunately, none of the above sequences appears in the Online Encyclopedia of Integer Sequences[3]. Consider the sequence generated by some arbitrary tie breaking rule. If z_i is the i-th term in this sequence, then $\lim_{n\to\infty} \lg(z_n)/n$ is the constant factor in the asymptotic space bound required to store all the answers to the NNLV problem subject to that tiebreaking rule.

1.1. Our Contributions

Our main results are as follows. First, we present the following upper bound:

Theorem 1. *Let $A[1..n]$ be an array containing distinct numbers. The array A can be processed to obtain an encoding data structure that occupies $1.89997n + o(n)$ bits of space, that can answer the query $\texttt{NNLV}(A, i)$ in $O(1)$ time for any $i \in [1, n]$. Ties are resolved using rule I. At no point after preprocessing does the data structure require access to the array A.*

[3]https://oeis.org/

Table 1: Number of distinguishable configurations of nearest larger value problems with the three tiebreaking rules discussed.

n	1	2	3	4	5	6	7	8	9	10	11	12
rule I	1	2	5	14	40	116	341	1010	3009	9012	27087	81658
rule II	1	2	5	14	42	126	383	1178	3640	11316	35263	110376
rule III	1	2	5	12	32	88	248	702	1998	5696	16304	46718

Table 2: Summary of results for the nearest larger value problem. The column distinct specifies whether the values are specified to be distinct or not. The column space bound indicates the type of result: "Matching Bounds" means that the bound presented is optimal to within lower order terms.

Distinct	Problem	Bound Type	Space	Query	Reference
Yes	Unidirectional	Matching Bounds	$2n + o(n)$	$O(1)$	Cartesian Tree [5, 6]
	Bidirectional	Matching Bounds	$2n + o(n)$	$O(1)$	Cartesian Tree [3, 4]
	Nondirectional (rule I)	Upper Bound	$1.89997n + o(n)$	$O(1)$	Sections 3 and 6
	Nondirectional (rule III)	Upper Bound	$1.81211n + o(n)$	$O(1)$	Sections 5 and 6
	Nondirectional (rule I)	Lower Bound	$1.62309n - \Theta(1)$	—	Section 8
No	Unidirectional	Matching Bounds	$2n + o(n)$	$O(1)$	Cartesian Tree [5]
	Bidirectional	Matching Bounds	$\lg(3 + 2\sqrt{2})n + o(n)$	$O(1)$	Schröder Trees [7]

As mentioned before, the Cartesian tree (defined later) occupies $2n+o(n)$ bits and can solve NNLV queries. In Section 3 we describe a novel *path-compressed* representation of a binary tree that uses $2n+O(\lg n)$ bits (but supports no operations). To get the improved space bound of Theorem 1 we prove combinatorial properties of the NNLV problem relating to chains of degree one nodes in the Cartesian tree. These properties allow us to compress the Cartesian tree using the representation of Section 3, losing some information, but still retaining the ability to answer NNLV queries. The constant factor (1.89997) comes from a numeric calculation bounding the worst case structure of chains in the Cartesian tree for our compression scheme (Section 4). Later, in Section 6 we show how to support operations on the "lossy" Cartesian tree, thereby proving Theorem 1.

We also present an alternate construction, based on forbidden patterns in binary strings, in Section 5. This construction is simpler than that of Section 3, and also achieves a better upper bound for a different tiebreaking rule.

Theorem 2. *Let $A[1..n]$ be an array containing distinct numbers. The array A can be processed to obtain an encoding data structure that occupies $1.81211n + o(n)$ bits of space, that can answer the query $\texttt{NNLV}(A, i)$ in $O(1)$ time for any $i \in [1, n]$. Ties are resolved using rule III. At no point after preprocessing does the data structure require access to the array A.*

After discussing upper bounds, in Section 7 we discuss methods to efficiently enumerate the number of distinguishable configurations of arrays of size n with respect to NNLV queries (subject to tiebreaking rule I). Using these methods we are able to extend the row for rule I of Table 1 to $n = 700$.

Using this extended table, in Section 8, we prove the following lower bound:

Theorem 3. *Any encoding data structure that can answer the query $\texttt{NNLV}(A, i)$ for any $i \in [1, n]$ (breaking ties according to rule I) must occupy at least $1.62309n - \Theta(1)$ bits, for sufficiently large values of n.*

A summary of the known results for all variants of the NLV problem can be found in Table 2. We leave the variant of the NNLV problem for non-distinct values as future work.

1.2. Other Related Work

Asano et al. [8] studied the time complexity of computing all nearest larger values in an array as well as higher dimensions, and mention applications to communication protocols. Asano and Kirkpatrick [9] considered sequential time-space tradeoffs for computing the nearest larger values of all elements in the array. When A is a random permutation, the expected effective entropy of the bidirectional NLV problem was shown to be $1.74n + o(n)$ bits by Golin et al. [2]. Finally, Jayapaul et al. [6] studied the nearest larger value problem in two dimensional arrays.

2. Cartesian Tree Review

Given a binary tree T, let $d(v)$ denote the degree (i.e., number of children) of node v, and $p(v)$ denote the parent of v. We define the rank $r(v)$ to be the inorder rank of the node v in the binary tree T. Define the *range* of a node v to be the range $[e_1(v), e_2(v)]$, where $e_1(v)$ (resp. $e_2(v)$) is the inorder rank of the leftmost (resp. rightmost) descendant of v.

Suppose we are given an array $A[1..n]$ which stores an n element permutation π, i.e., $A[i] = \pi(i)$. The Cartesian tree of $A[1..n]$ is the n node binary tree T such that the root v of T has rank $r(v) = \arg\max_i A[i]$. If $r(v) > 1$, then the left child of v is the Cartesian tree of $A[1..r(v) - 1]$, otherwise it has no left child. If $r(v) < n$ then the right child of v is the Cartesian tree of $A[r(v)+1..n]$, otherwise it has no right child. Figure 1 (bottom panel) illustrates the Cartesian tree of an example array (top panel).

We require the following technical lemma about Cartesian trees:

Lemma 1. *Consider a node v in a Cartesian tree T having range $[e_1(v), e_2(v)]$. If $e_1(v) - 1 \geq 1$ then $A[e_1(v) - 1] > A[r(v)]$. Similarly, if $e_2(v) + 1 \leq n$ then $A[e_2(v) + 1] > A[r(v)]$.*

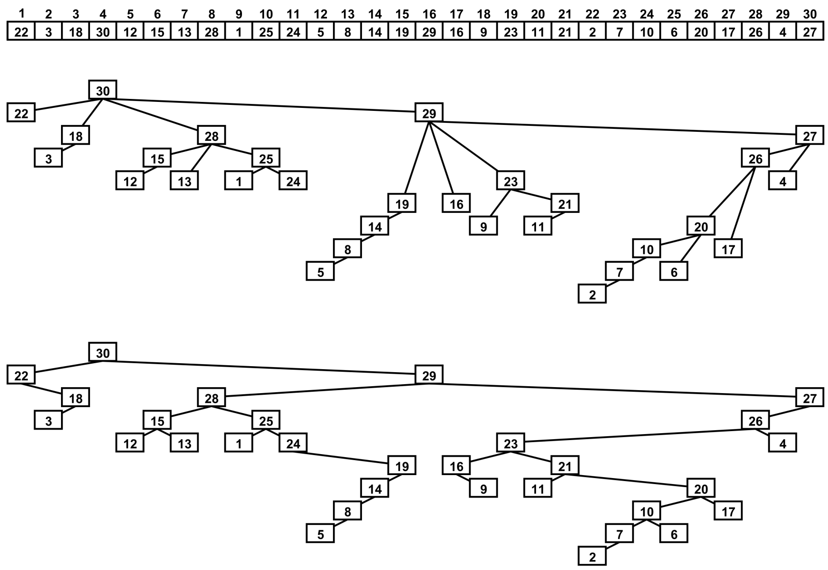

Figure 1: Top: an array containing a permutation of $\{1, \ldots, 30\}$. Middle: The tree structure of the NNLV problem. Here the parent of a node represents its NNLV, breaking ties by selecting the element on the right (rule I). Bottom: The Cartesian tree.

Proof. If v is the root of T, then $e_1(v) - 1 = 0$ and $e_2(v) + 1 = n + 1$, and the lemma holds trivially. Thus, suppose $p(v)$ exists. Since the Cartesian tree is binary, we have two cases: (i) $e_1(p(v)) = e_1(v)$, which implies $r(p(v)) = e_2(v)+1$ and therefore $A[e_2(v) + 1] > A[r(v)]$; or (ii) $e_2(p(v)) = e_2(v)$, which implies $r(p(v)) = e_1(v) - 1$, and therefore $A[e_1(v) - 1] > A[r(v)]$. This proves the lemma in the case where either $e_1(v) - 1 = 0$ or $e_2(v) + 1 = n + 1$. Next, suppose that both $e_1(v) - 1 \geq 1$ and $e_2(v) + 1 \leq n$. Thus, let u be the closest ancestor of v such that $e_1(u) \neq e_1(v)$ and $e_2(u) \neq e_2(v)$. By the definition of u, $u \neq p(v)$. Consider the child of u, denoted w, that contains v in its subtree. There are two cases: (i) $e_2(w) = e_2(v)$, which implies $r(u) = e_2(v) + 1$ and therefore $A[e_1(v) - 1] = A[r(p(v))] > A[r(v)]$ and $A[e_2(v) + 1] = A[r(u)] > A[r(v)]$; or (ii) $e_1(w) = e_1(v)$, which implies $r(u) = e_1(v) - 1$, and therefore $A[e_1(v) - 1] = A[r(u)] > A[r(v)]$ and $A[e_2(v) + 1] = A[r(p(v))] > A[r(v)]$. $\square$

NLV and Cartesian Trees. Lemma 1 is key to using the Cartesian tree for computing NLVs. Let v be a node in a Cartesian tree with range $[e_1(v), e_2(v)]$. Then the following observations are immediate:

1. If $e_1(v) > 1$ then $e_1(v)-1$ is the nearest larger value to the left of $A[r(v)]$. If $e_1(v) = 1$ then there is no larger value to the left of $A[r(v)]$.

2. If $e_2(v) < n$ then $e_2(v)+1$ is the nearest larger value to the right of $A[r(v)]$. If $e_2(v) = n$ then there is no larger value to the right of $A[r(v)]$.

Thus, by comparing $r(v) - e_1(v)$ and $e_2(v) - r(v)$ we can obtain $\texttt{NNLV}(A, r(v))$ according to tie break rule I.

If $e_1(v) > 1$, then let w be the vertex corresponding to $A[e_1(v) - 1]$. If $e_2(v) < n$, then let x be the vertex corresponding to $A[e_2(v) + 1]$. The following observations follow directly from the definition of the Cartesian tree.

1. If $e_1(v) = 1$ and $e_2(v) = n$ then v is the root of the Cartesian tree.

2. If $e_1(v) > 1$ and $e_2(v) = n$ then $w = p(v)$.

3. If $e_1(v) = 1$ and $e_2(v) < n$ then $x = p(v)$.

4. If $e_1(v) > 1$ and $e_2(v) < n$ either w or x is $p(v)$.

Assume that $e_1(v) > 1$ and $e_2(v) < n$. We now argue that either w is a descendant of x, or vice versa. If not, let u be the LCA of w and x. Clearly $A[r(u)]$ is greater than both $A[e_1(v)-1]$ and $A[e_2(v)+1]$, and hence, by Lemma 1, $A[r(u)] > A[r(v)]$. However $e_1(v) - 1 < r(u) < e_2(v) + 1$, i.e. $r(u)$ is within the extent of v, a contradiction. Thus, we can break ties according to rules II or III as well, by seeing if $r(p(v)) = e_1(v) - 1$ or $r(p(v)) = e_2(v) + 1$.

3. A Path Based Tree Representation

Consider an arbitrary rooted binary tree T with n nodes. We next describe a path-based encoding of such a tree that occupies no more than $2n + \Theta(\lg n)$ bits.

We identify all maximal chains $v_1, ..., v_\ell, v_{\ell+1}$ such that:

1. Either v_1 is the root of T, or $d(p(v_1)) = 2$;
2. $d(v_i) = 1$ for $i \in [1, \ell]$, and;
3. $d(v_{\ell+1}) \in \{0, 2\}$.

We refer to $v_{\ell+1}$ as the *terminal* of the chain. Iteratively, we remove each such maximal chain: i.e., the nodes $v_1, ..., v_\ell$ are removed from the tree. If v_1 was the root, then $v_{\ell+1}$ is set to be the new root. Otherwise, $v_{\ell+1}$ is set to be the left (resp. right) child of $p(v_1)$ iff v_1 was the left (resp. right) child of $p(v_1)$. We call the chain *left hanging* if $p(v_1)$ had v_1 as a left child, and *right hanging* otherwise. After removing all such maximal chains, the tree T' that remains is a full binary tree (i.e., it has no nodes of degree one) and has $n' \leq n$ nodes. Suppose that we have removed k nodes, for some $k \in [0, n-1]$, and so $n = n' + k$.

Suppose there are m maximal chains removed during the process just described. We now describe the representation of the original tree T.

- We store the tree T', which is a full binary tree and requires $n' + O(1)$ bits to represent.
- We store a bitvector B of length n'. Bit $B[i] = 1$ iff the node v, corresponding to the i-th node in an inorder traversal of T', is the terminal of a removed chain. This requires $\lceil \lg \binom{n'}{m} \rceil$ bits.
- Suppose we order the subset of nodes that are terminals by their inorder rank in T', and that v is the terminal ordered i-th. We refer to the chain having v as its terminal as $\mathcal{C}_i$, and its length as c_i. We store a bitvector L of length k, which represents the lengths of each removed chain; i.e., the values $c_1, ..., c_m$. Let $p_i = \sum_{j=1}^{i} c_i$ for $i \in [1, m]$. Then $L[p_i] = 1$ for $i \in [1, m]$, and all other entries of L are 0. As L is a bit sequence of length k with m one bits, it can be stored using $\lceil \lg \binom{k}{m} \rceil$ bits.
- For each chain $\mathcal{C}_i = \{v_1, ..., v_{c_i}\}$ having terminal node v_{c_i+1}, we store a bitvector Z_i of length c_i, in which $Z_i[j] = 0$ if v_{j+1} is the left child of v_j,

and $Z_i[j] = 1$ otherwise. Let Z be the concatenation of each Z_i, $i \in [1, m]$ and is of length k. We store Z naively using k bits.

We call the above data structures, bitvectors B, L, Z and the tree T' the *path compressed* representation of T. Note that to decode this and recover the tree T, we require the value of n and n'. These can be stored using an additional $\Theta(\lg n)$ bits. By summing the above space costs, we get the following lemma.

Lemma 2. *The path compressed representation of an arbitrary binary tree T completely describes the combinatorial structure of T, and can be stored using $n' + \lg \binom{n'}{m} + \lg \binom{k}{m} + k + \Theta(\lg n) \leq 2n' + 2k + \Theta(\lg n) = 2n + \Theta(\lg n)$ bits.*

4. Encoding Nearest Larger Values

In this section we show how to use the path compressed tree representation to compress Cartesian trees—losing some information in the process—but still retaining the ability to answer NNLV queries. Our key observation is that chains in the Cartesian tree can be compressed to save space, as illustrated by the following lemma:

Lemma 3. *Consider the set of all possible chains with c_i deleted nodes in a path compressed representation of a Cartesian tree, excluding chains having nodes representing array elements $A[1]$ or $A[n]$. There are exactly $c_i + 1$ combinatorially distinct chains with respect to answering nearest larger value queries, breaking ties according to rule I.*

Proof. Consider a chain with c_i deleted nodes, $\{v_1, ..., v_{c_i}\}$, where v_{c_i+1} is the terminal. Clearly, v_1 represents the maximum element in the chain, and either $r(v_j) = e_1(v_j)$ or $r(v_j) = e_2(v_j)$ for each $j \in [1, c_i]$. This follows because since if v_j is in a chain it is either the left or right endpoint of the range $[e_1(v_j), e_2(v_j)]$. In turn, this implies that the range $[e_1(v_1), e_2(v_1)]$ has a *deleted prefix* and *deleted suffix* which in total contain the inorder ranks of the c_i deleted nodes.

The deleted nodes corresponding to this prefix (resp. suffix) appear contiguously in the array A, and form a decreasing (resp. increasing) run of values in A.

Furthermore, by Lemma 1, and since $1, n \notin [e_1(v_1), e_2(v_1)]$ (by the assertion in the statement of the lemma), we can assert that both $A[e_1(v_1) - 1] > A[e_1(v_1)]$ and $A[e_2(v_1) + 1] > A[e_2(v_1)]$. Thus, for each k such that v_k is in the prefix we have that $A[e_1(v_k) - 1] > A[e_1(v_k)]$, and we can return the nearest larger value of $r(v_k) = e_1(v_k)$ to be $e_1(v_k) - 1$. Similarly, for each k such that v_k is in the suffix we have that $A[e_2(v_k) + 1] > A[e_2(v_k)]$, and return the nearest larger value of $r(v_k) = e_2(v_k)$ to be $e_2(v_k) + 1$.

This implies that, if we know the value c_i, then we additionally need only know how many nodes are in the prefix in order to determine the answer to a nearest larger value query for any index represented by a deleted node. There are at most $c_i + 1$ possible options: $\{0, ..., c_i\}$. Moreover, for an arbitrary index $i \in [1, n] \setminus [e_1(v_1), e_2(v_1)]$ the answer to a nearest larger value query cannot be in $[e_1(v_1), e_2(v_2)]$, since this range is sandwiched between larger values by Lemma 1. Finally, consider indices in the range $[e_1(v_{c_i+1}), e_2(v_{c_i+1})]$. Using the fact that $A[e_1(v_{c_i+1}) - 1]$ and $A[e_2(v_{c_i+1}) + 1]$ by are larger than all elements in $A[e_1(v_{c_i+1}), e_2(v_{c_i+1})]$ by Lemma 1, we can correctly answer queries for a position i in the subtree. First, we find the solution to the NNLV query within the subtree, and denote the index as j. Then, we return the nearest position to i of either j, $e_1(v_{c_i+1}) - 1$, or $e_2(v_{c_i+1}) + 1$, breaking ties according to rule I. $\square$

Recall that recovering a chain of c_i deleted nodes exactly required c_i bits in the path compressed tree representation. In contrast, the previous lemma allows us to get away with $\lg(c_i + 1)$ bits: an exponential improvement. Using the above lemma, we get the following upper bound for the NNLV problem (note that, on its own, it does not allow queries to be performed efficiently).

Lemma 4. *The solutions to all nearest larger value queries can be encoded using no more than* $1.89997 + o(n)$ *bits of space.*

Proof. We store the path compressed version of T, the Cartesian tree of A. However, we replace index Z, by an index Z' consisting of $\lceil \lg \prod_{i=1}^{m} (c_i + 1) \rceil$ bits. Z' represents, for each deleted chain—including those that contain nodes representing $A[1]$ and $A[n]$—the length of its deleted prefix. We explicitly store the answers to nearest larger value queries for $A[1]$ and $A[n]$.

For the remaining $A[i]$, $i \in [2, n-1]$ there are two options:

1. $A[i]$ is represented by a node in a deleted chain from the Cartesian tree T. By Lemma 3 the replacement index Z' is enough information to recover the nearest larger values for all deleted nodes, with the exception of those in chains containing the nodes representing $A[1]$ or $A[n]$. For such a chain $\mathcal{C}_i$, the information recorded in Z' indicates a number $\Delta \in \{0, ..., c_i + 1\}$. Suppose the chain $v_1, ..., v_{\ell+1}$ contains a node u representing $A[1]$. If u is in the deleted prefix, then u represents the largest element in the deleted prefix, so the only information lost by storing Δ is the nearest larger value of $A[1]$. If u is in the deleted suffix, then it represents the smallest element in the deleted suffix, and the nearest larger value can be inferred. The case where a node in the chain represents $A[n]$ is symmetric. Since we store the nearest larger values of $A[1]$ and $A[n]$ explicitly, we can therefore recover the nearest larger value of all deleted nodes.

2. $A[i]$ is represented by a node u in the Cartesian tree T'. In this case, can infer the nearest larger value as follows. Let s_ℓ be the size of the $T(\text{left}(u))$, which is equal to the $T'(\text{left}(u))$ plus the lengths of the chains deleted from $T'(\text{left}(u))$, s_r is defined analogously for $\text{right}(u)$. Then the nearest larger value is either $A[i - s_\ell - 1]$ or $A[i + s_r - 1]$, depending on which is closer. Ties can be broken according to rule I.

The space bound for storing the data structures described is $n' + \lg \binom{n'}{m} + \lg \binom{k}{m} + \lg \prod_{i=1}^{m}(c_i + 1) + O(\lg n)$ bits. This is bounded by $n' + \lg \binom{n'}{m} + \lg \binom{k}{m} + m \lg(\frac{k}{m} + 1) + O(\lg n)$ bits using Jensen's inequality. Numerical methods (see Appendix A) reveal that this expression is upper bounded by $1.91975n + \Theta(\lg n)$ bits. In the sequel we show how to improve this bound. The main idea is to replace the representation of the lengths of the chains, the data structure L, with a slightly more space efficient structure.

Since there are m chains, let σ denote the number of distinct chain sizes. We consider the sequence $\mathcal{R} = \{c_1, c_2, ..., c_m\}$, letting m_j denote the number of occurrences of symbol j in $\mathcal{R}$. Given such a sequence we use $H(\mathcal{R})$ to denote the zeroth-order empirical entropy of the sequence $\mathcal{R}$.[4] Thus, we need only store:

$$\sigma\lceil \lg n\rceil + nH(\mathcal{R}) + O(1) =$$
$$\sigma\lceil \lg n\rceil + \sum_{i=1}^{\sigma}\left(m_i \lg \frac{m}{m_i}\right) + O(1) \text{ bits}$$

in order to reconstruct each $c_1, ..., c_m$.

We can also rewrite the term $\lg \prod_{i=1}^{m}(c_i+1)$ to get

$$\lg \prod_{i=1}^{\sigma}(i+1)^{m_i} = \sum_{i=1}^{\sigma} m_i \lg(i+1)$$

Combining the two sums, this gives us that the total space bound is:

$$n' + \lg \binom{n'}{m} + \sum_{i=1}^{\sigma}\left(m_i \lg \frac{m(i+1)}{m_i}\right) + \sigma\lceil \lg n\rceil + O(\lg n) <$$
$$n' + \lg \binom{n'}{m} + \sum_{i=1}^{\sigma}\left(m_i \lg \frac{m(i+1)}{m_i}\right) + O(\sqrt{n}\lg n)$$

bits, since the number of distinct chain lengths can be at most $\Theta(\sqrt{n})$. We can then rewrite the equation, recalling $n' = n - k$, $k = \sum_{i=1}^{\sigma}(im_i)$, and letting $y_i = m_i/n$, and $Y = k/n$, and $y = \sum_{i=1}^{\sigma}(y_i)$, to get:

$$n - k + \lg \binom{n-k}{m} + \sum_{i=1}^{\sigma}\left(m_i \lg \frac{m(i+1)}{m_i}\right) + O(\sqrt{n}\lg n) <$$
$$n\left((1-Y)\left(1 + H\left(\frac{y}{1-Y}\right)\right) + \sum_{i=1}^{\sigma}\left(y_i \lg \frac{y(i+1)}{y_i}\right) + o(1)\right)$$

By numerical methods (see Appendix A), we find that this expression is upper bounded by $1.89997n + o(n)$; a slight improvement. $\square$

5. Forbidden Patterns

In this section we describe a simpler approach to upper-bounding the number of distinguishable configurations of the `NNLV` problem, based on forbidden

[4]Overloading notation, if x is a probability instead of a sequence of symbols, we use $H(x)$ to denote the standard binary entropy of x: $x\lg(\frac{1}{x}) + (1-x)\lg(\frac{1}{1-x})$.

patterns. As described in the previous section, the aim is, given an input array A, to come up with a new array A_0, such that using the Cartesian tree T_0 of A_0 to compute NNLVs using the above approach will give the same answer as for A. Our goal is to ensure that this new Cartesian tree T_0 will be more compressible. The general approach is to consider the encodings of the modified Cartesian trees as strings over an alphabet, then argue that certain substrings are forbidden in these encodings, and count the number of strings that exclude these substrings to upper-bound the number of modified Cartesian trees.

5.1. Forbidding Zig-Zags

Say that a degree 1 node is an *turn* if it is the right child of its parent, and it has a left child, or it is the left child of its parent and it has a right child (we take the root as being the left child of an imaginary super-root). It is a *non-turn* otherwise. Consider the encoding of a node of a binary tree where a turn is encoded as $b = 01$, a non-turn is encoded using $c = 10$ and degree 2 nodes and leaves are encoded $d = 11$ and $a = 00$ respectively. For any binary tree T, let $\mathcal{E}(T)$ be the sequence of symbols that give the encoding of the nodes of the T, visiting the nodes of T in depth-first order. Overloading notation, for any array A containing distinct items, we use $\mathcal{E}(A)$ to denote $\mathcal{E}(T)$ where T is the Cartesian tree of A. We now claim:

Lemma 5. *Given any array A of size n, if $\mathcal{E}(A)$ has a subsequence of the form bc^kb for some $k \geq 0$, then there is an array A_0 such that in $\mathcal{E}(A_0)$, the above subsequence is replaced by cbc^k, such that all* NNLV *queries on A and A_0 return the same answer, except possibly for* NNLV(1) *and* NNLV(n).

Proof. We first consider the case where the indices in A representing the given subsequence are $1 > i_1, \ldots, i_{k+2} > n$. Assume without loss of generality that the node corresponding to i_1 has only a right child and let i_0 be the parent of i_1 (i_0 always exists, since i_1 cannot be the root: the root cannot be a degree 1 node unless it is at position 1 or n). We make the following observations (see Figure 2):

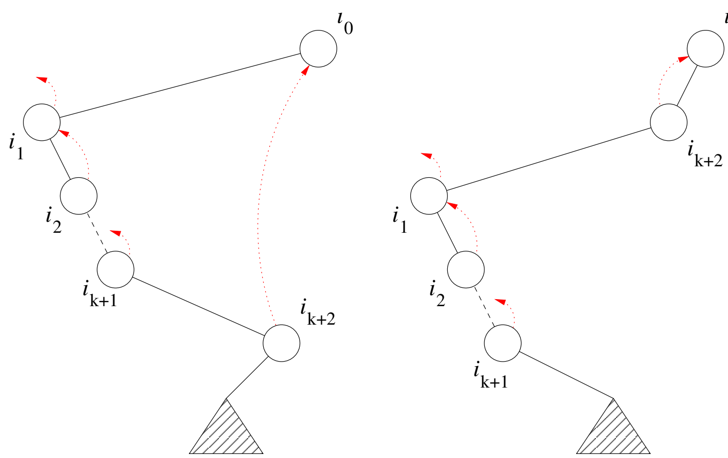

Figure 2: Diagrammatic representation of Lemma 5.

1. $i_1, \ldots, i_{k+1}$ have only a right child (since $i_2, \ldots, i_{k+1}$ are non-turn nodes).
2. i_{k+2} has only a left child.
3. $i_{j+1} = i_j + 1$ for $j = 1, \ldots, k$.
4. $i_0 = i_{k+1} + 1$.
5. $i_k < i_{k+1} - 1$.

Clearly, $A[i_j] > A[i_{j+1}]$ for $j = 0, \ldots, k$. Thus, we have:

6. $\texttt{NNLV}(A, i_j) = i_{j-1}$ for $j = 2, \ldots, k+1$
7. $\texttt{NNLV}(A, i_1) = i_1 - 1$, since $i_0 - i_1 \geq 3$.
8. $\texttt{NNLV}(A, i_{k+2}) = i_0$, since $i_{k+2} - i_{k+1} > 1$.

Thus, we can create a new array A_0 such that $A[i_1] < A_0[i_{k+2}] < A[i_0]$ and $A[j] = A_0[j]$ for all $j \neq i_{k+2}$. It is easy to verify that all NNLV answers in A_0 are the same as in A.

Now suppose that i_1 is the root of the Cartesian tree. For it to be a turn, it must be the case that i_1 only has a right child (recall that the root is the left child of its imaginary parent). Then $i_1 = 1$, and furthermore, $i_{k+2} = n$. In

A_0, the only NNLV answers that change will be for 1 and n. A similar argument shows that if i_1 is not the root of the Cartesian tree and it is the left (right) child of its parent i_0 then $i_1 = 1$ ($i_1 = n$) is the only possibility not yet considered; in this case $\texttt{NNLV}(1)$ ($\texttt{NNLV}(n)$) is the only answer that changes. □

5.2. Forbidding a Turn Before a Leaf

Lemma 6. *Given an array A of size n, if $\mathcal{E}(A)$ has a subsequence of the form ba, then there is an array A_0 such that in $\mathcal{E}(A_0)$, the above subsequence is replaced by ca, such that all NNLV queries, except possibly on positions 1 and n on A_0 return a correct answer for A, using Rule III for tiebreaking (break ties to larger).*

Proof. First consider the case that the nodes labelled b and a correspond to the indices $1 < i$ and $i+1 < n$. In this case, we have the following observations.

- The parent of i must be $i + 2$.
- $i+1$ has two equidistant larger values, i and $i+2$, and $\texttt{NNLV}(A, i+1) = i+2$ since $A[i + 2] > A[i]$.
- Observe that $i - 1$ must be an ancestor of i meaning $A[i - 1] > A[i]$, and hence $\texttt{NNLV}(A, i) = i - 1$.

In A_0, we exchange the values $A[i]$ and $A[i+1]$. Now $\texttt{NNLV}(A_0, i+1) = i+2$ as before. $A_0[i]$ now has two equidistant larger values, $i - 1$ and $i + 1$, since $A_0[i - 1] = A[i - 1] > A[i] = A_0[i + 1]$, so $\texttt{NNLV}(A_0, i) = i - 1$ as before.

The case that b is $i + 1 < n$ and a is $i > 1$ is symmetric. Finally, it can be verified that if i or $i + 1$ is one of the boundary elements $A[1]$ or $A[n]$, the only possibilites where $\texttt{NNLV}(i, A_0) \neq \texttt{NNLV}(i, A)$ are the cases $i = 1$ or $i = n$. □

5.3. Counting Strings with Forbidden Sub-Patterns

Overview. We apply the transformations on A repeatedly until the patterns ba and bc^*b do not exist. We upper-bound the number of distinct modified Cartesian trees by the number of distinct strings over the alphabet $\{a, b, c, d\}$

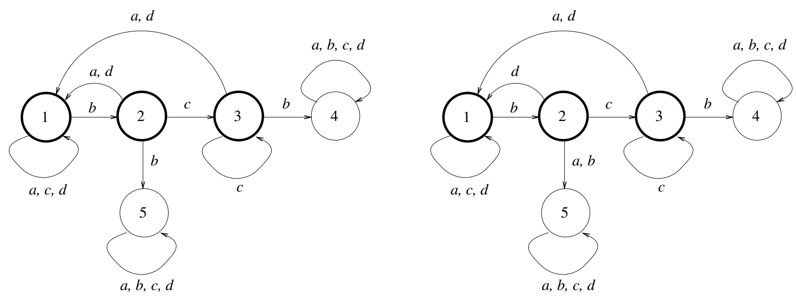

Figure 3: Automata for all strings over $\{a, b, c, d\}$ excluding bc^*b (left), and both bc^*b and ba (right). The initial state is 1 and the final states are 1, 2, and 3 in each case.

that exclude just the pattern bc^*b and that exclude both bc^*b and ba. To count the number of strings with forbidden sub-patterns, we use the *transfer matrix* approach [12]. In this approach, we first create a DFA with states $s_1, \ldots, s_k$ that accepts exactly those strings which do not have any forbidden substrings (assume wlog that s_1 is the start state). We then create the transfer matrix M, which is a $k \times k$ matrix where the (i, j)-th entry is the number of distinct symbols that label a transition from s_i to s_j. It is not hard to see that the (i, j)-th entry of $\sum_{i=0}^{\infty}(Mz)^i = (I - Mz)^{-1}$, where z is a formal variable, is the generating function for the number of distinct strings that lead from s_i to s_j. Summing up the $(1, j)$-th entries of $(I - Mz)^{-1}$ for all final states j gives the required generating function.

In our case, the generating functions will be *rational* functions of the form $P(z)/Q(z)$ where P and Q are polynomials. To obtain asymptotic upper bounds on the coefficient of z^n we use the following:

Theorem 4 (Rational Expansion Theorem[12]). *If $R(z) = P(z)/Q(z)$ is the generating function for the sequence $\langle r_n \rangle$, where $Q(z) = (1 - \rho_1 z)(1 - \rho_2 z) \ldots (1 - \rho_\ell z)$, and the numbers $(\rho_1, \ldots, \rho_\ell)$ are distinct, and if $P(z)$ is a polynomial of degree less than ℓ, then $r_n = \sum_{i=1}^{\ell} a_i \rho_i^n$ for constants $a_1, \ldots, a_\ell$.*

Results. Let f_n (g_n) for $n \geq 1$ denote the number of distinct strings of length n over the alphabet $\{a, b, c, d\}$ that exclude just the pattern bc^*b (exclude both bc^*b and ba, respectively). The automata that accept all strings over $\{a, b, c, d\}$ excluding bc^*b, and both bc^*b and ba are shown in Figure 3. The corresponding transfer matrices are below:

$$M_1 = \begin{pmatrix} 3 & 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 1 \\ 2 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 3 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 2 \\ 2 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{pmatrix}$$

We obtain $(I - M_1 z)^{-1}$ as:

$$\begin{pmatrix} \frac{1-z}{z^2-4z+1} & \frac{-(z-1)z}{z^2-4z+1} & \frac{z^2}{z^2-4z+1} & \frac{z^3}{-4z^3+17z^2-8z+1} & \frac{(z-1)z^2}{(4z-1)(z^2-4z+1)} \\ \frac{2z}{z^2-4z+1} & \frac{3z^2-4z+1}{z^2-4z+1} & \frac{z-3z^2}{z^2-4z+1} & \frac{z^2(3z-1)}{(4z-1)(z^2-4z+1)} & \frac{z(3z^2-4z+1)}{-4z^3+17z^2-8z+1} \\ \frac{2z}{z^2-4z+1} & \frac{2z^2}{z^2-4z+1} & \frac{-2z^2-3z+1}{z^2-4z+1} & \frac{z(2z^2+3z-1)}{(4z-1)(z^2-4z+1)} & \frac{2z^3}{-4z^3+17z^2-8z+1} \\ 0 & 0 & 0 & \frac{1}{1-4z} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{1-4z} \end{pmatrix}$$

Adding together the entries in positions $(1, 1), (1, 2)$, and $(1, 3)$ we get that $F(z) = \frac{1-z-(z-1)z+z^2}{z^2-4z+1} = \frac{1}{z^2-4z+1}$ is the generating function for $\langle f_n \rangle$. The roots of $z^2 - 4z + 1$ are $2 + \sqrt{3}$ and $2 - \sqrt{3}$ giving $\rho_1 = 1/(2 + \sqrt{3}) < 0.27695$ and $\rho_2 = 1/(2 - \sqrt{3}) < 3.73206$. From this we use Theorem 4 to conclude that $\lg f_n = n \lg 3.73206 + o(n) = 1.89997n + o(n)$.

We remark that the constant achieved via the above calculation is strikingly similar to that of Lemma 4. We have been unable to determine whether the bounds achieved by the two approaches are indeed equal, or just matching up to five decimal places.

Similarly $(I - M_2 z)^{-1}$ equals:

$$\begin{pmatrix} \frac{z-1}{z^3-2z^2+4z-1} & \frac{(z-1)z}{z^3-2z^2+4z-1} & \frac{z^2}{-z^3+2z^2-4z+1} & \frac{z^3}{4z^4-9z^3+18z^2-8z+1} & \frac{-2(z-1)z^2}{(4z-1)(z^3-2z^2+4z-1)} \\ \frac{-z(z+1)}{z^3-2z^2+4z-1} & \frac{-3z^2+4z-1}{z^3-2z^2+4z-1} & \frac{z(3z-1)}{z^3-2z^2+4z-1} & \frac{z^2-3z^3}{4z^4-9z^3+18z^2-8z+1} & \frac{2z(3z^2-4z+1)}{(4z-1)(z^3-2z^2+4z-1)} \\ \frac{-2z}{z^3-2z^2+4z-1} & \frac{-2z^2}{z^3-2z^2+4z-1} & \frac{z^2+3z-1}{z^3-2z^2+4z-1} & \frac{-z(z^2+3z-1)}{(4z-1)(z^3-2z^2+4z-1)} & \frac{4z^3}{4z^4-9z^3+18z^2-8z+1} \\ 0 & 0 & 0 & \frac{1}{1-4z} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{1-4z} \end{pmatrix}$$

From which we get that $G(z) = \frac{z-1+(z-1)z+z^2}{z^3-2z^2+4z-1} = \frac{-1}{z^3-2z^2+4z-1}$. Solving for $z^3 - 2z^2 + 4z - 1 = 0$ we get that $\rho_1 < 3.51155$, and that ρ_2 and ρ_3, both of which are complex, have magnitude < 0.53365. From this we use Theorem 4 to conclude that $\lg g_n = n \lg 3.51155 + o(n) = 1.81211n + o(n)$.

6. Data Structures

To accomplish the goal of supporting operations, we use a technical modification of the mini-micro tree decomposition presented by Farzan and Munro [10] which can be stated as follows:

Lemma 7 (Theorem 1 [10]). *For any parameter $k > 1$, a tree with n nodes can be decomposed into $\Theta(\frac{n}{k})$ subtrees of size at most $2k$, which are pairwise disjoint aside from their roots. With the exception of edges branching from the root of a subtree, there is at most one edge from a non-root node in a subtree to a node outside the subtree.*

We also make use of the succinct binary tree data structure of Davoodi et al. [4] that supports the following operations in $O(1)$ time, and represents an arbitrary binary tree using $2n + o(n)$ bits:

1. `select_inorder`(T, i): return the node u in T having inorder number i.
2. `subtree_size(T,u)`: Return the size of the subtree rooted at node u in T.
3. `parent`(T, u): Return the parent node of u in T.

Algorithm 1 Computing `NNLV`(A, i).

if $i = 1$ or $i = n$ **then**
  return explicitly stored answer for $A[1]$ or $A[n]$.
else
  $\ell \leftarrow$ `subtree_size(left(select_inorder(`i`)))`
  $r \leftarrow$ `subtree_size(right(select_inorder(`i`)))`
  if $\ell < r$ and $i - \ell - 1 \geq 1$ **then**
    return $i - \ell - 1$
  else if $i + r + 1 \leq n$ **then**
    return $i + r + 1$
  else
    return -1 ($A[i]$ is the maximum, and has no NNLV)
  end if
end if

4. `left`(T, u) (resp. `right`(T, u)): return node u's left (resp. right) child in T.

We are now ready to prove Theorem 2. We note that the same machinery can be applied to prove Theorem 1, almost verbatim, except we replace the forbidden pattern representation with that of Lemma 4.

Proof. Given the array A, we obtain an array A_0 as described in Sections 5.1 and 5.2. We create the Cartesian tree T_0 of A_0, and seek to represent T_0 in $1.81211n + o(n)$ bits so that NNLV queries can be answered in $O(1)$ time.

We now present a straightforward modification of the succinct binary tree representation of Davoodi et al. [4]. The representation of Davoodi et al. applies Lemma 7 to decompose the tree into $O(n/\lg n)$ micro-trees of with at most $\lceil \frac{\lg n}{k} \rceil$ nodes, for some constant $k \geq 8$. Apart from the space needed to represent the micro-trees, the space usage of their representation is $o(n)$ bits.

Our objective is to replace the encoding of the micro-trees with one based on Sections 5.1 and 5.2. For each micro-tree μ of ν nodes, we create $\mathcal{E}(\mu)$ as a string of length ν over the alphabet $\{a, b, c, d\}$ as above. Observe that patterns that are forbidden in $\mathcal{E}(T_0)$ are also (essentially) forbidden in $\mathcal{E}(\mu)$. The only exception is that a degree 1 or 2 node v in μ may have its children in another micro-tree. Then v is a leaf of μ, and if its parent is a degree 1 turn node, then a forbidden pattern may appear in $\mathcal{E}(\mu)$. However, this can happen only for nodes which have their children in another micro-tree, and there is at most one such node by Lemma 7. The information needed to store the modifications to $\mathcal{E}(\mu)$ so that $\mathcal{E}(\mu)$ now has no forbidden patterns takes at most $O(\lg \nu) = O(\lg \lg n)$ bits, which is negligible summed over all micro-trees since the number of micro-trees is $O(n/\lg n)$.

The representation of $\mathcal{E}(\mu)$ is as a pointer into a table that stores all possible trees whose encodings have no forbidden patterns; this pointer clearly takes $1.81211 \cdot \nu + O(1)$ bits. The representations of all micro-trees take $1.81211 \cdot n + O(n/\lg n)$ bits. Since the encoding of each micro-tree takes at most $(\lg n)/4 + O(1)$ bits, operations on nodes inside a micro-tree can be done in $O(1)$ time by table-lookup, as in [4]. Finally, the algorithm to answer the NNLV query is presented in Algorithm 1, and uses only operations supported by the representation of Davoodi et al. [4]. □

7. Exact Enumeration of Distinguishable Configurations

In this section we count the distinguishable configurations with respect to `NNLV` queries. We obtain recursive formulae for the precise numbers of distinguishable configurations. The values calculated by these formulae will also improve the lower bound as indicated in Theorem 3.

In total we define six sequences. Of these, Γ is the sequence that counts the number of distinguishable configurations of the `NNLV` problem; the others are auxiliary sequences. We will use the notation of a *chain of nearest neighbours* for a list of numbers $L_1, \dots, L_n$ where the nearest neighbour of L_i is L_{i+1}. Further, we use the term *distinguishable configurations of* $A[l \dots r]$ for some l, r when configurations can be distinguished by asking the queries `NNLV`(i) for $i = l, \dots, r$.

1. $\mathcal{A} = \{\alpha_n\}_{n\in\mathbb{N}}$. Let $A[0\ldots n+1]$ be an array of $n+2$ numbers such that $A[0]$ and $A[n+1]$ are greater than $A[i]$ for all $1 \le i \le n$. Then α_n is the number of distinguishable configurations of $A[1\ldots n]$. Figure 7 (first row) denotes the conditions for $\mathcal{A}$ graphically.

2. $\mathcal{B} = \{\beta_n\}_{n\in\mathbb{N}}$. Let $A[0\ldots n]$ be an array of $n+1$ numbers such that $A[0] > A[i]$ for all $1 \le i \le n$. Then β_n is the number of distinguishable configurations of $A[1\ldots n]$. Figure 7 (second row) denotes the conditions for $\mathcal{B}$ graphically.

3. $\mathcal{B}^{\text{rev}} = \{\beta_n^{\text{rev}}\}_{n\in\mathbb{N}}$. Let $A[1\ldots n+1]$ be an array of $n+1$ numbers such that $A[n+1] > A[i]$ for all $1 \le i \le n$. Then β_n^{rev} is the number of distinguishable configurations of $A[1\ldots n]$.

4. $\Gamma = \{\gamma_n\}_{n\in\mathbb{N}}$. Let $A[1\ldots n]$ be an array of n numbers. Then β_n is the number of distinguishable configurations of $A[1\ldots n]$. Figure 7 (third row) denotes the conditions for Γ graphically.

5. $\Delta = \{\delta_{n,m}\}_{n,m\in\mathbb{N}}$. Let $A[0\ldots n+m+1]$ be an array of $n+m+2$ numbers such that $A[0] > A[i]$ for all $1 \le i \le n$; $A[n+m+1] > A[i]$ for all $1 \le i \le n+m$; $A[n] > A[i]$ for all $n+1 \le i \le n+m$; and there exists a chain of nearest neighbours from $A[n]$ to $A[0]$ without including $A[n+m+1]$. Then $\delta_{n,m}$ is the number of distinguishable configurations of $A[1\ldots n]$. Figure 7 (fourth row) denotes the conditions for Δ graphically.

6. $\Delta^{\text{rev}} = \{\delta_{n,m}^{\text{rev}}\}_{n,m\in\mathbb{N}}$. Let $A[-m\ldots n+1]$ be an array of $n+m+2$ numbers such that $A[n+1] > A[i]$ for all $1 \le i \le n$; $A[-m] > A[i]$ for all $-m+1 \le i \le n$; $A[1] > A[i]$ for all $-m+1 \le i \le 0$; and there exists a chain of nearest neighbours from $A[1]$ to $A[-m]$ without including $A[n+1]$. Then $\delta_{n,m}^{\text{rev}}$ is the number of distinguishable configurations of $A[1\ldots n]$.

All sequences share that they count the number of distinguishable configurations of $A[1\ldots n]$. In the special case of $n=0$ the array $A[1\ldots n]$ is of length

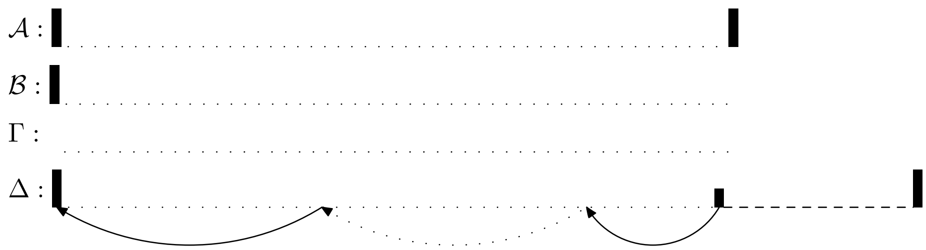

Figure 4: Overview of conditions for each sequence

zero. No queries can be made that distinguishes between any two configurations. Hence the number of distinguishable configurations is 1 and in particular $\alpha_0 = \beta_0 = \beta_0^{\text{rev}} = \delta_{0,m} = \delta_{0,m}^{\text{rev}} = 1$ for $m \in \mathbb{N}$.

We first prove some auxiliary lemmas.

Lemma 8. *Let $A[0 \ldots n+1]$ be an array of $n+2$ numbers such that $A[0] > A[i]$ for all $1 \leq i < (n+1)/2$ and $A[n+1] > A[j]$ for all $1 \leq j \leq n$. Then α_n is the number of distinguishable configurations of $A[1 \ldots n]$.*

Proof. Any array of $n+2$ numbers that satisfy the condition for sequence $\mathcal{A}$ also satisfies the condition of this lemma. Hence there are at least α_n distinguishable configurations of $A[1 \ldots n]$ where A satisfies the condition of this lemma. Lets assume there exist an array $A[0 \ldots n+1]$ of $n+2$ numbers that satisfies the condition of this lemma and its configuration of $A[1 \ldots n]$ is distinguishable from all conditions of $\bar{A}[1 \ldots n]$ within an array $\bar{A}$ that satisfy the condition of sequence $\mathcal{A}$. We now consider the array $A'[0 \ldots n+1]$ defined by $A'[i] = A[i]$ for all $1 \leq i \leq n+1$ and $A'[0] = A[n+1]+1$. So A' satisfies the condition of sequence $\mathcal{A}$ and A' configuration of $A'[1 \ldots n]$ must distinguishable A's configuration of $A[1 \ldots n]$. As both arrays are the same apart from $A'[0] > A[0]$ there must exists an $1 \leq j \leq n$ which nearest neighbour in A' is $A[0]$ and in A it is not $A[0]$. So $A[0] < A[j] < A'[0]$. Since A satisfies the condition of this lemma $j \geq (n+1)/2$. This is contradiction as any such position is not closer to $A'[0]$ than to $A'[n+1]$, in $A'[j]$ nearest neighbour cannot be $A'[0]$. Hence by weakening the condition of sequence $\mathcal{A}$ to the condition of this lemma does not

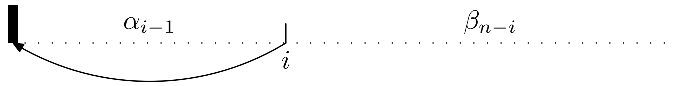

Figure 5: β_n sequence step

create further distinguishable configuration on $A[1 \dots n]$. $\square$

A similar approach will give following lemma.

Lemma 9. *Let $A[0 \dots n+1]$ be an array of $n+2$ numbers such that $A[0] > A[i]$ for all $1 \leq i \leq n$ and $A[n+1] > A[j]$ for all $(n+1)/2 \leq j \leq n$. Then α_n is the number of distinguishable configurations of $A[1 \dots n]$.*

Now we give the formulae for each sequence.

Lemma 10. $\beta_n \quad = \sum_{i=1}^{n} \alpha_{i-1} \beta n - i$

Proof. Let $A[0 \dots n]$ be an array satisfying the condition of sequence $\mathcal{B}$. Further let $A[i]$ be the highest number in $A[1 \dots n]$. Then $A[0]$ has no nearest neighbour in A. The nearest neighbour of $A[i]$ is $A[0]$. The nearest neighbours of $A[1]$ to $A[i-1]$ must lie within the subarray $A[0 \dots i]$ which also satisfies the condition sequence $\mathcal{A}$. Similarly the the nearest neighbours of $A[i+1]$ to $A[n]$ must lie with the subarray $A[i \dots n]$, which satisfies the condition of sequence $\mathcal{B}$. Hence, the number of distinguishable configuration of $A[1 \dots n]$ is $\alpha_{i-1}\beta_{n-i}$.

For any $1 \leq i \leq n$, $A[i]$'s nearest neighbour is $A[0]$ and there does not exists a number $A[j]$ with $j > i$ such that $A[j]$'s nearest neighbour is $A[0]$. Hence all configurations for one value of i are distinguishable from configurations of different value of i.

Summing up all distinguishable configuration for all possible values of i gives the above formula of β_n.

$\square$

Following the structure of the proof of Lemma 10 one can show that $\beta_n^{\text{rev}} = \sum_{i=1}^{n} \beta_{i-1}^{\text{rev}} \alpha n - i = \sum_{i=1}^{n} \alpha_{i-1} \beta_{n-i}^{\text{rev}}$. Since $\beta_0^{\text{rev}} = 1 = \mathcal{B}_0^{\text{rev}}$ we have the following lemma.

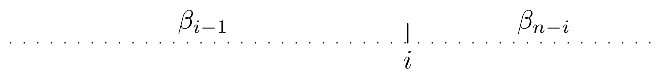

Figure 6: γ_n sequence step

Lemma 11. *$\beta_n^{rev} = \beta_n$ for all $n \in \mathbb{N}$.*

Lemma 12. $\gamma_n \quad = \sum_{i=1}^{n} \beta_{i-1}\beta_{n-i}$

Proof. Let $A[1 \dots n]$ be an array of n numbers. Further let $A[i]$ be the highest number in A. Then $A[i]$ has no nearest neighbour in A; the nearest neighbours of $A[1]$ to $A[i-1]$ lie in the subarray $A[1 \dots i-1]$ which satisfy the condition of B'; the nearest neighbours of $A[i+1]$ to $A[n]$ lie in the subarray $A[1 \dots n]$ which satisfies the condition of B. See figure 7.

Hence the number of distinguishable configurations of $A[1 \dots n]$ is $\beta_{i-1}\beta_{n-i}$. For any $1 \leq i \leq n$, $A[0]$ is the only number that has no nearest neighbour in A. Hence all configurations for one value of i are distinguishable from configurations of different value of i. Summing up all distinguishable configuration for all possible values of i gives the above formula of γ_n. $\square$

Lemma 13. $\delta_{n,m} \quad = \sum_{i=1}^{\min(m,n)} \alpha_{i-1}\delta_{(n-i,i+m)}$

Proof. As given by the conditions of the sequence Δ there exists a chain of nearest neighbours from $A[n]$ to $A[0]$. So let i be the distance of the $A[n]$ to its nearest neighbour. Then there exists a chain of nearest neighbours from $A[n-i]$ to $A[0]$, $A[n-i]$ is greater than any number between $A[n-i+1]$ to $A[n+m]$, and also $A[n+m+1]$ remain higher than all number in $A[1 \dots n]$. Hence there are $\delta_{n-i,m+i}$ distinguishable configuration for $A[1]$ to $A[n-i]$. The subarray $A[n-i+1 \dots n$ satisfy the condition of $\mathcal{A}$. So every i there are $\alpha_{i-1}\delta_{n-i,i+m}$ distinguishable configurations. Building the sum of over all possible values of i give the formula of $\delta_{n,m}$. $\square$

Following the same structure as used in the proof of Lemma 13 one can show that $\delta_{n,m}^{\text{rev}} = \sum_{i=1}^{\min(m-1,n)} \alpha_{i-1}\delta_{(n-i,i+m)}^{\text{rev}}$. Since $\delta_{0,m}^{\text{rev}} = 1 = \Delta_{0,m}^{\text{rev}}$ for all $m \in \mathbb{N}$ we have the following lemma.

Lemma 14. $\delta^{rev}_{n,m} = \delta^{rev}_{n,m-1}$ *for all* $n, m \in \mathbb{N}$.

Lemma 15. *Let* $r = \left\lfloor \frac{2n+1}{3} + 1 \right\rfloor$ *and* $l = \left\lfloor \frac{r}{2} \right\rfloor$ *then*

$$\begin{aligned}
\alpha_n = &\sum_{k=l}^{r-1} \alpha_{k-1}\alpha_{n-k} + \\
&\sum_{i=1}^{l-1}\sum_{j=r}^{n} \alpha_{i-1}\alpha_{j-i-1}\alpha_{n-j} + \\
&\sum_{i=1}^{l-1}\sum_{j=r}^{n} \sum_{k=2i+1}^{i+\left\lfloor \frac{j-i+1}{2} \right\rfloor - 1} \sum_{p=2k-i+1}^{\min(j,2k)} \alpha_{i-1}\alpha_{n-j}\alpha_{k-i-1}\alpha_{p-k-1}\delta_{j-p,p} + \\
&\sum_{i=1}^{l-1}\sum_{j=r}^{n} \sum_{k=i+\left\lfloor \frac{j-i+1}{2} \right\rfloor}^{2j-1-n} \sum_{p=\max(i,2k-n)}^{2k-j} \alpha_{i-1}\alpha_{n-j}\alpha_{j-k-1}\alpha_{k-p-1}\delta_{p-i,n-p} + \\
&\sum_{j=r}^{n} \alpha_{n-j}\delta_{j-1,1}
\end{aligned}$$

Proof. The setting of α_n splits into five disjoint cases, see Figure 7. Each of them corresponds to a line in the formula of α_n. We use i for the rightmost position that has $A[0]$ as its nearest neighbour and j for the leftmost position that has $A[n+1]$ as its nearest neighbour. As we break ties to the right, j must always exists but i must not. Further we split $A[1 \ldots n]$ in three roughly evenly sized parts: First third $A[1 \ldots l-1]$, the middle third $A[l \ldots r-1]$ and the last third $A[r \ldots n]$ with $r = \left\lfloor \frac{2n+1}{3} + 1 \right\rfloor$ and $l = \left\lfloor \frac{r}{2} \right\rfloor$.

We first identify three cases that cover all configurations and do not share any configurations. We later split case 2 into three separate sub-case and reach our five cases as given in the formula.

- Case 1: Either i or j lies in middle third.
- Case 2: The position i lies in the first third and j lies in the last third.
- Case 3: The position j in the last third and i does not exists.

From the definitions the cases have disjoint configurations. To show all configuration are covered by the three cases one has to show that there are no configurations with i in last third or j in the first third. As the nearest neighbour of $A[i]$ is $A[0]$ it must be strictly closer to $A[0]$ than to $A[n+1]$ as both of them are higher than $A[i]$ and we break ties to the right. As $r = \lfloor \frac{2n+1}{3} + 1 \rfloor \geq \frac{2n+1+1}{3} > \frac{n+1}{2} \geq \lfloor n+1 \rfloor 2$ any position in the last third cannot have $A[0]$ as its nearest neighbour. Similarly for the first third, as $l - 1 = \lfloor r \rfloor 2 - 1 < \frac{2n+1+3}{6} = \frac{n}{3} + \frac{1}{2} \leq \frac{n+1}{2}$ a number in the first third cannot have $A[n+1]$ as its nearest neighbour.

We will now go through each case and justify the corresponding part in the formula of α_n.

Case 1. We first assume $A[i]$ lies in the middle third. Note that this does not mean $A[i]$ is highest number among $A[1 \dots n]$, but it is higher than any $A[1]$ to $A[i-1]$. Also it is higher than any $A[i+1]$ to $A[i+i]$. If one shows that $i+i$ is at least $\lfloor (n+1+i)/2 \rfloor - 1$ by Lemma 8 the number of distinguishable configuration will be $\alpha_{i-1}\alpha_{n-i}$. So

$$i + i \geq \lfloor (n+1+i)/2 \rfloor - 1$$

$$\Leftarrow 3i \geq n$$

$$\Leftarrow 3\left\lfloor \frac{r}{2} \right\rfloor \geq \frac{3r-3}{2} \geq \frac{3(\lfloor \frac{2n+1}{3} + 1 \rfloor) - 3}{2} \leq \frac{2n+1+3-3}{2} > n + \frac{1}{2}$$

If we assume $A[j]$ lies in the middle third, by a similar argument and by the Lemma 8 the number of distinguishable configuration is $\alpha_{j-1}\alpha_{n-j}$.

So the number of all distinguishable configurations covered by case 1 is $\sum_{k=l}^{r-1} \alpha_{k-1}\alpha_{n-k}$.

Case 2. So $A[i]$ lies in the first third and $A[j]$ lies in the last third. $A[i]$ or $A[j]$ must the be the highest value among $A[1 \dots n]$. However the other does not have to be the second highest. We first assume that $A[i]$ and $A[j]$ are the highest and second highest values (case 2a). This gives $\sum_{i=1}^{l-1} \sum_{j=r}^{n} \alpha_{i-1}\alpha_{j-i-1}\alpha_{n-j}$ distinguishable configurations.

In case 2b, we count the additional configuration to the case 2a, when $A[j]$ is highest and $A[i]$ is not the second highest value in among $A[1 \dots n]$. In order to have a configuration that has not yet been counted in case 2a there must exists an $A[k] > A[i]$ with $i < k < j$ such that $A[k]$ nearest neighbour is not $A[i]$ but it would be so if $A[i]$ would have been the second highest. The number $A[k]$ must lie further away from $A[i]$ then $A[i]$ is from $A[0]$ as $A[i]$'s nearest neighbour is $A[0]$. Also $A[k]$ must be closer to $A[i]$ then to $A[j]$. Hence the range for k is from $2i+1$ to $i+\left\lfloor \frac{j-i+1}{2} \right\rfloor - 1$. In a single configuration there might multiple numbers that satisfy the condition of $A[k]$. To avoid double counting of configuration we assume that $A[k]$ is the furthest left of such numbers and hence count the configuration in between $A[i]$ and $A[k]$ by α_{k-i-1}. We now consider $A[p]$ the nearest neighbour of $A[k]$. The number $A[k]$ must be closer to $A[i]$ than to $A[p]$ as it other wise never has $A[i]$ as its nearest neighbour independent of the value of $A[i]$. Hence $2k - i + 1 \geq p$. As $A[p]$ is $A[k]$'s nearest neighbour, $A[p]$ must be less or the same distance away from $A[k]$ than $A[k]$ is from $A[0]$, as otherwise $A[0]$ is $A[k]$'s nearest neighbour. Also it must not lie to the right of $A[j]$. Hence $p \leq \min(j, 2k)$. From $A[p]$ onwards there must be a chain of nearest neighbours to $A[j]$. Hence the number of configuration for case 2b are

$$\sum_{i=1}^{l-1} \sum_{j=r}^{n} \sum_{k=2i+1}^{i+\left\lfloor \frac{j-i+1}{2} \right\rfloor - 1} \sum_{p=2k-i+1}^{\min(j,2k)} \alpha_{i-1}\alpha_{n-j}\alpha_{k-i-1}\alpha_{p-k-1}\delta_{j-p,p}$$

The case 2c is equivalent to case 2b with $A[i]$ being the highest number and $A[j]$ not being the second highest number in $A[1 \dots n]$. The reasoning concerning the range for k and p are similar and lead to $k \geq i + \left\lfloor \frac{j-i+1}{2} \right\rfloor$, $j - k \geq n - j + 1$, $k - p < n + 1 - k$, $p \geq i$ and $k - p > j - k$ Hence all configuration of case 2c are

$$\sum_{i=1}^{l-1} \sum_{j=r}^{n} \sum_{k=i+\left\lfloor \frac{j-i+1}{2} \right\rfloor}^{2j-1-n} \sum_{p=\max(i,2k-n)}^{2k-j} \alpha_{i-1}\alpha_{n-j}\alpha_{j-k-1}\alpha_{k-p-1}\delta_{p-i,n-p}$$

Case 3. There is no number that has $A[0]$ as it's nearest neighbour. So there is a chain of nearest neighbours from $A[1]$ to $A[j]$. The number of configuration for $A[1 \dots j]$ is $\delta_{n-j,1}$ and for $A[j+1 \dots n$ is α_{n-j}. The total number of configuration of case 3 is

$$\sum_{j=r}^{n} \alpha_{n-j}\delta_{j-1,1}$$

By adding up the formulae for each case we obtain the formula for α_n.

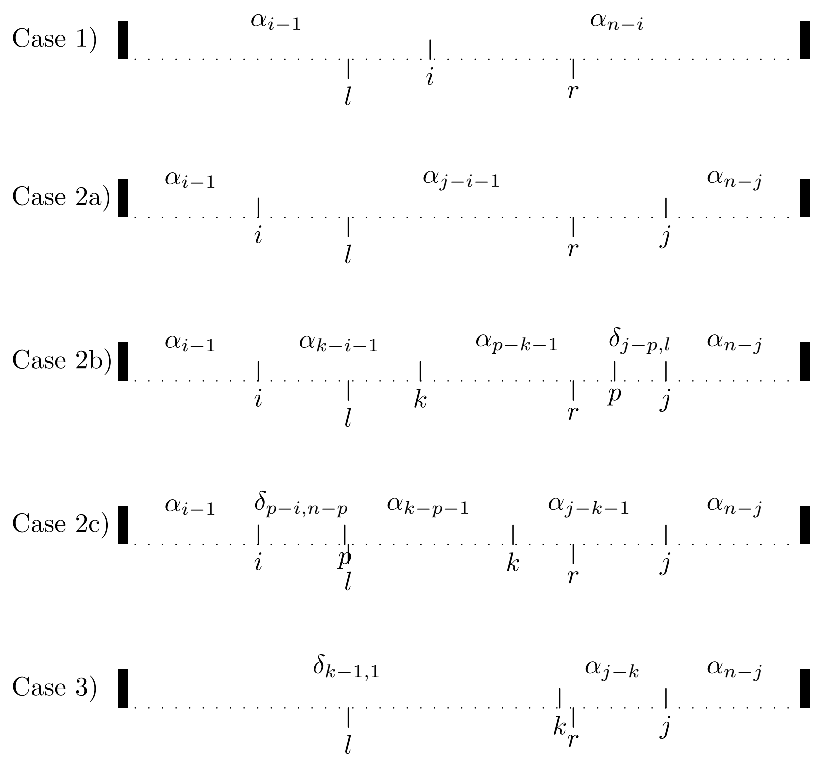

Figure 7: α_n sequence step

□

We were unable to give a closed-form solution for γ_n. We have, however, computed them up to $n = 700$. Figure 7 shows the ratios of γ_n/γ_{n-1} and α_n/α_{n-1}, and 2^k_n where k_n is lower bound of bytes obtained by using the formula, as given in Section 8, with the values of γ_n and α_n. Up to the computed values the ratios for Γ and $\mathcal{A}$ are monotonic increasing.

8. Lower Bound

The main idea of the lower bound is to show that for a given n, there are many configurations of A that can be distinguished by NNLV queries. To do this, we reuse the sequence definitions for both α_n and γ_n from the previous section: α_n can be considered to be a *restricted input* to the NNLV problem

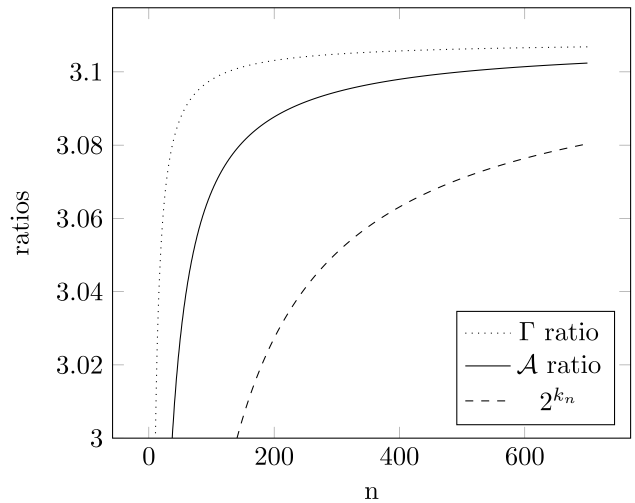

Figure 8: The ratios of consecutive values for sequence Γ and $\mathcal{A}$, 2 to the power of the lower bound of bytes.

that is padded with two additional entries $A[0] = \infty$ and $A[n+1] = \infty$.

For an n element array, we use α_n to denote the number of different configurations for the NNLV problem on restricted inputs, and γ_n to denote the number of solutions to NNLV, both subject to tie breaking rule I. We present both sequences in Table 3 for some values of n and up to $n = 700$.

Next we discuss how to the values in Table 3 to derive a lower bound. Consider an array of length n, for n sufficiently large. Without loss of generality, we assume that a parameter $\ell \geq 1$ divides $n-2$ and that $\frac{n-2}{\ell}$ is odd. Let D_i denote the i-th odd block, and E_i denote the i-th even block. Locations $A[1]$ and $A[n]$ are assigned values $n-1$ and n, respectively. Odd block D_i is assigned values $[(i-1)\ell+1, i\ell]$, and can be arranged in one of α_ℓ configurations, to form an instance of a restricted input. Suppose there are λ odd blocks. Even block E_i will be assigned values from $[(\lambda+i-1)\ell+1, (\lambda+i)\ell]$, and arranged in one of the γ_ℓ configurations of the NNLV problem.

Our claim is that each even (resp. odd) block can be assigned any of the γ_ℓ (resp. α_ℓ) possible configurations, without interference from other blocks. To see this, consider that for each even block we have assigned values so that—with

n	γ_n	α_n
0	1	1
1	1	1
2	2	2
3	5	4
4	14	9
5	40	22
6	116	55
7	341	142
8	1010	378
9	3009	1015
10	9012	2768
50	2.60634×10^{23}	1.76356×10^{22}
200	1.23839×10^{97}	2.13372×10^{95}
400	3.14284×10^{195}	2.71590×10^{193}
700	1.49191×10^{343}	7.37685×10^{340}

Table 3: The calculated values of γ_n and α_n for some selected values of n.

the exception of the maximum element—the nearest larger value to all elements must be within the same block. This follows since the adjacent odd blocks contain strictly smaller values than those in any even block. Moreover, for odd blocks, the values immediately to the left and right of the block are strictly larger than any values in the block. Thus, we can force the global solution to the NNLV problem on the entire array into at least $(\gamma_\ell \alpha_\ell)^{\frac{n-2}{2\ell}}$ distinct structures. This implies that $\lg \gamma_n$ is at least $\frac{(n-2)}{2\ell} \lg(\gamma_\ell \alpha_\ell)$: selecting $\ell = 700$ yields the lower bound of Theorem 3.

9. Conclusions

We have introduced the encoding NNLV problem, and have noted its combinatorial richness. Using a novel path-compressed representation of Cartesian trees, we gave a space-efficient NNLV encoding that supports queries in $O(1)$ time. Determining the effective entropy of NNLV, and to consider the other NNLV variants (such as for arrays of non-distinct values), is an open problem. Finding ways to apply NNLV encodings to compressed suffix trees, as Fischer [7] did for his bidirectional NLV encoding, would also be interesting.

We conclude with a final remark about the sub-optimality of our approaches for representing an NNLV tree. To show that these data structures are sub-optimal, consider the following two example arrays:

$$[10, 12, 8, 9, 1, 7, 3, 4, 2, 6, 5, 13, 11] \text{ and } [10, 12, 7, 9, 1, 6, 3, 4, 2, 8, 5, 13, 11]$$

Both of these arrays have *different* cartesian trees which contain no degree one nodes, however they both have the same NNLV tree under the three tie-breaking rules. This indicates that any strategy which focuses only on degree one nodes in the Cartesian tree is unlikely to achieve optimal bounds, whatever they may be.

References

[1] O. Berkman, B. Schieber, U. Vishkin, Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values, J. Algorithms 14 (3)

(1993) 344–370. doi:10.1006/jagm.1993.1018.
URL http://dx.doi.org/10.1006/jagm.1993.1018

[2] M. J. Golin, J. Iacono, D. Krizanc, R. Raman, S. R. Satti, S. M. Shende, Encoding 2d range maximum queries, Theor. Comput. Sci. 609 (2016) 316–327. doi:10.1016/j.tcs.2015.10.012.
URL http://dx.doi.org/10.1016/j.tcs.2015.10.012

[3] J. Fischer, V. Heun, Space-efficient preprocessing schemes for range minimum queries on static arrays, SIAM Journal on Computing 40 (2) (2011) 465–492.

[4] P. Davoodi, G. Navarro, R. Raman, S. Rao, Encoding range minima and top-2 queries, Phil. Trans. R. Soc. A 372 (2016) (2014) 1471–2962.
URL https://lra.le.ac.uk/handle/2381/28856

[5] J. Fischer, V. Mäkinen, G. Navarro, Faster entropy-bounded compressed suffix trees, Theor. Comput. Sci. 410 (51) (2009) 5354–5364.

[6] V. Jayapaul, S. Jo, R. Raman, V. Raman, S. R. Satti, Space efficient data structures for nearest larger neighbor, J. Discrete Algorithms 36 (2016) 63–75. doi:10.1016/j.jda.2016.01.001.
URL http://dx.doi.org/10.1016/j.jda.2016.01.001

[7] J. Fischer, Combined data structure for previous- and next-smaller-values, Theor. Comput. Sci. 412 (22) (2011) 2451–2456. doi:10.1016/j.tcs.2011.01.036.
URL http://dx.doi.org/10.1016/j.tcs.2011.01.036

[8] T. Asano, S. Bereg, D. G. Kirkpatrick, Finding nearest larger neighbors, in: S. Albers, H. Alt, S. Näher (Eds.), Efficient Algorithms, Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday, Vol. 5760 of Lecture Notes in Computer Science, Springer, 2009, pp. 249–260. doi:10.1007/978-3-642-03456-5_17.
URL http://dx.doi.org/10.1007/978-3-642-03456-5_17

[9] T. Asano, D. G. Kirkpatrick, Time-space tradeoffs for all-nearest-larger-neighbors problems, in: F. Dehne, R. Solis-Oba, J. Sack (Eds.), Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings, Vol. 8037 of Lecture Notes in Computer Science, Springer, 2013, pp. 61–72. doi:10.1007/978-3-642-40104-6_6.
URL http://dx.doi.org/10.1007/978-3-642-40104-6_6

[10] A. Farzan, J. I. Munro, A uniform paradigm to succinctly encode various families of trees, Algorithmica 68 (1) (2014) 16–40. doi:10.1007/s00453-012-9664-0.
URL http://dx.doi.org/10.1007/s00453-012-9664-0

[11] R. Raman, V. Raman, S. S. Rao, Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets, ACM Transactions on Algorithms 3 (4).

[12] P. Flajolet, R. Sedgewick, Analytic Combinatorics, 1st Edition, Cambridge University Press, New York, NY, USA, 2009.

Appendix A. Mathematica Code

In this section we present Mathematica code for numerical maximization that leads to the bounds presented in Lemma 4. The following snippet produces first bound discussed in the Lemma:

```
h[x_] :=  x*Log2[1/x] + (1-x) * Log2[1/(1-x)]
f[m_, k_]:= 1-k + (1-k)*h[m/(1-k)] + k*h[m/k] + m*Log2[k/m+1]
NMaximize[ { Re[f[m, k]],
             k > $MachineEpsilon, m > $MachineEpsilon,
             m < k, k < 1 },
           {k, m}, {MaxIterations->100000, Method->{"RandomSearch"}}]
```

The second bound is slightly more involved, and involves optimizing over a vector to determine the values of y_i that maximize the entropy of the chains. Through experimentation, we determined that $y_i = 0$ for all $i > 15$ (hence setting $\sigma = 15$ below).

```
Sigma = 15;
g[m_,k_]:=
  1-(Sum[m[[i]]*i, {i,1,k}])  +
  (1-(Sum[m[[i]]*i, {i,1,k}])) *
  h[(Sum[m[[i]], {i,1,k}]) / (1-(Sum[m[[i]]*i, {i,1,k}]))] +
  Sum[m[[j]]*Log2[(Sum[m[[i]], {i,1,k}])/m[[j]]], {j,1,k}]  +
        Sum[m[[j]]*Log2[j+1], {j,1,k}];
yArr = Array[Unique[y],{Sigma}]
NMaximize[{ Re[g[yArr,Sigma]],
            And@@Table[yArr[[i]]>=$MachineEpsilon, {i,Sigma} ] &&
            (Sum[yArr[[i]], {i,1,Sigma}]) <
            1 - (Sum[yArr[[i]] * i, {i,1,Sigma}])},
            yArr, {MaxIterations->100000, Method->{"RandomSearch"}}]
```