Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 331 (2026)

Sorting with $o(N \log N)$ Tape Steps and Boolean Gates

Yann Tal*

Abstract

We give a deterministic algorithm that stably sorts n binary keys of arbitrary lengths on a fixed multitape Turing machine, carrying along any associated data. It also sorts signed numbers in numerical order given their finite expansions in any fixed base. If N is the total encoded input length in bits, including the associated data, the algorithm uses

$$O\left(\frac{N \log n}{(\log\log\log n)^{1/2-o(1)}}\right) \quad \text{time and} \quad O(N) \quad \text{space.}$$

The worst-case running time is therefore $o(N \log N)$. A deterministic algorithm that sorts by comparing keys and copies each output bit from the corresponding bit of its input item uses $\Omega(N \log N)$ tape steps in the worst case.

Applying the reduction of Harvey and van der Hoeven to OpenAI's integer multiplication algorithm yields an improved time bound for matrix transposition on a multitape Turing machine. We adapt the external-memory sorting algorithm of Li, Li and Liu, using transposition to group keys by their next digit on tape. At each step we distribute the items across rows so that most fit in columns reserved for their current digit. Within each row we split the items into pieces so that most pieces can be copied in order. Transposing the layout then brings together items with the same digit.

The same rule for routing pieces within a row yields a circuit construction. Polynomial multiplication over $\mathbb{F}_2$ gives small circuits for its cyclic shifts. The resulting circuits sort binary keys of arbitrary lengths with their associated data using

$$O\left(\frac{N \log N}{(\log\log\log\log N)^{1/2-o(1)}}\right)$$

bounded-fan-in Boolean gates. In particular, these circuits sort n integers of $\Theta(\log n)$ bits each using $o(n \log^2 n)$ gates.

*yanntal@mail.tau.ac.il

ISSN 1433-8092

Contents

1 Introduction **3**
- 1.1 Earlier Algorithms and Main Results . . . 3
- 1.2 Overview of the Sorting Algorithms . . . 6
- 1.3 Organization of the Paper . . . 7

2 Preliminaries **8**
- 2.1 The Tape Model . . . 8
- 2.2 The Cost of Copying Input Bits . . . 8
- 2.3 Interchanging Two Address Fields . . . 9
- 2.4 Boolean Circuits . . . 9

3 Local Permutations of Binary Strings **10**
- 3.1 A Balanced Word for All Relabelings . . . 10
- 3.2 Routing the Pieces of a Binary String . . . 11

4 Radix Sorting **13**
- 4.1 Distributing One Digit . . . 13
- 4.2 Gathering Items by Digit . . . 15
- 4.3 Proof of Theorem 1.2 . . . 15

5 Permutation Routing and General Inputs **16**
- 5.1 Removing the Width Restrictions . . . 17
- 5.2 Proof of Theorem 1.1 . . . 18

6 Sorting with $o(N \log N)$ Boolean Gates **19**
- 6.1 Routing the Pieces of an Item . . . 20
- 6.2 Routing by Destination Digits . . . 20
- 6.3 Computing Stable Destinations . . . 22
- 6.4 Proof of Theorem 1.3 . . . 24

7 Open Problems **25**

AI Use Disclosure **25**

Bibliography **26**

A Wide Binary Strings **29**

B Implementation on Two Writable Tapes **32**
- B.1 Copying and Merging Streams . . . 32
- B.2 Address-Field Interchange . . . 33
- B.3 The Sorting Passes . . . 34

C The Movement Bound for Bit Copies **35**

D Circuits for Delimited Strings **35**

E Lower Bounds for Restricted Circuit Constructions **39**

F Reductions Between Shifting, Routing and Sorting **41**
- F.1 Cyclic Shifts and Stable Sorting in Boolean Circuits . . . 41
- F.2 Permutation Routing and Stable Sorting on Tapes . . . 42

1 Introduction

1.1 Earlier Algorithms and Main Results

How many tape steps or Boolean gates suffice to sort N bits of input? In the classical comparison model studied by Ford and Johnson [FJ59], sorting n distinct keys involves at least $\lceil \log(n!) \rceil$ comparisons in the worst case, since a binary comparison tree must distinguish all $n!$ input orders [CLRS22, proof of Theorem 8.1].

On tapes, the machine must also move the keys and their associated data into sorted order. Suppose a sorter determines the order by comparing keys and each output bit comes, through a chain of copies, from the corresponding bit of the same input item. We call such a machine a *whole-key comparison sorter*. The bits of an item may move separately. Even with this allowance, moving the items can cost $\Omega(N \log N)$ steps in the worst case on inputs of N bits (Section 2.2).

For string inputs, we fix a finite alphabet and an order on its symbols. We sort the keys lexicographically, carrying their associated data. For variable-length inputs, a delimiter follows each key and each associated string. The sort is *stable*, meaning that items with equal keys appear in their original relative order. An order-preserving encoding also lets us sort signed finite expansions in any fixed base on tapes.

In a *comparison network*, the pairs of positions to be compared are fixed in advance, and each comparison places the two items in key order. Both a tape machine and a Boolean circuit may combine bits from different items while rearranging them. Our guiding question is whether this freedom can improve asymptotically on whole-key comparison sorting on tapes and on the usual implementation of comparison networks by Boolean gates. We answer that question in the affirmative. For N input bits, our multitape algorithm uses $o(N \log N)$ steps and our bounded-fan-in Boolean circuits use $o(N \log N)$ gates.

We write $[a] = \{0, \ldots, a-1\}$ for a positive integer a. Throughout, log denotes the base-two logarithm and $\log^{(j)} x$ its j-fold iterate. For example, $\log^{(3)} x = \log(\log(\log x))$. In asymptotic bounds and width hypotheses, a logarithm of a size parameter x means $\log_2 \max\{2, x\}$ at each iteration. In identities and integer bit counts, logarithms have their usual values. An exponent $1/2 + o(1)$ in an upper bound means that the same machine or circuit family satisfies the bound with exponent $1/2 + \varepsilon$ for every fixed $0 < \varepsilon < 1/2$. For a denominator with exponent $1/2 - o(1)$ the corresponding exponent is $1/2 - \varepsilon$. The time or gate-count constant may depend on ε. The linear tape-space bound has a single constant.

Integer sorting on a word RAM. A word RAM can read or write an entire word at a computed address in one operation. Radix sort uses this access to distribute $O(\log n)$-bit keys by their digits in $O(n)$ word operations [CLRS22, Lemma 8.4]. For arbitrary single-word keys, Fredman and Willard's fusion trees sort in $O\left(n \log n / \log^{(2)} n\right)$ deterministic worst-case time on a word RAM with multiplication and a fixed number of precomputed constants whose values depend on the word width [FW93, pp. 425, 427]. Linear-space algorithms for the same problem use $O\left(n \log^{(2)} n\right)$ word operations deterministically [Han04] or $O\left(n\sqrt{\log^{(2)} n}\right)$ in expectation [HT02, Theorem 1]. For word widths of at least $(\log n)^{2+\varepsilon}$ bits and fixed $\varepsilon > 0$, Andersson, Hagerup, Nilsson and Raman [AHNR98, p. 74] gave a randomized algorithm using $O(n)$ expected word operations.

A word RAM can write a list of n integers of $\lceil \log n \rceil$ bits each in $O(n)$ word operations. Writing the same list to a fresh tape costs $\Omega(n \log n)$ tape steps.

Sorting on tapes. Hopcroft, Paul and Valiant [HPV75, Section 3] studied the time to sort $N/\log N$ integers of $\log N$ bits each. They showed that a nondeterministic sorting algorithm running in $O(N \log N/\log^* N)$ time on $k \geqslant 2$ tapes would suffice to separate one-tape and k-tape nondeterministic computation for the time bounds in their theorem. Here $\log^* N$ counts the successive binary logarithms needed to reach a value at most one.

For deterministic sorting, Schnorr [Sch78, Program p_1, p. 139] described a mergesort for an encoded list of integers using a linear-time three-tape merge. Each of the $O(\log n)$ merge levels costs $O(N)$ steps. Penttonen and Katajainen [PK85, pp. 621–622] stated the resulting $O(N \log n)$ bound for at least three tapes and asked whether it could be improved or proved optimal. Their Theorem 3.1 gives a linear-time algorithm on a pointer machine. Floyd and Smith [FS74] gave a linear-time two-tape merge for fixed-width items. For n integers of w bits each, Salamon and Wehar [SW23, Section 8.3, Lemma 20] give the bound $O(nw \log n)$ tape steps.

Lower bounds and data movement. Paul [Pau79, Section 3] reduced matrix transposition to sorting by labeling each entry with its position in the transposed matrix. His $\Omega(N \log N)$ lower bound for a binary tape with a fixed number of heads assumes a restriction on how information about a matrix entry can accumulate in the tape blocks visited during a time interval. He measures the information in each block separately using conditional Kolmogorov complexity, and bounds the increase in their sum over the interval. As Paul observes, several blocks together can determine an entry even when each block alone reveals little about it [Pau79, pp. 328, 333].

Dietzfelbinger, Maass and Schnitger [DMS91, Theorem 5.1] showed that, for infinitely many total input lengths N, sorting logarithmic-width strings can cost $\Omega\left(N^{3/2}/\sqrt{\log N}\right)$ steps on a machine with one work tape and a two-way read-only input tape.

From transposition to sorting. Let $\mathcal{V}$ be the bit volume of an $r \times s$ matrix with $r, s \geqslant 2$ and entries of equal width, and let $M(\mathcal{V})$ be the time to multiply two $\mathcal{V}$-bit integers on a fixed multitape Turing machine. Harvey and van der Hoeven [HvdH25, Corollary 6.2 of the full version] transpose the matrix in $O\left(M(\mathcal{V}) + \mathcal{V} \log^{(2)} \max\{r, s\}\right)$ tape steps. OpenAI first reduced the exponent of $\log \mathcal{V}$ in the multiplication bound below one [Ope26a, Theorem 1], then obtained $M(\mathcal{V}) = O\left(\mathcal{V}(\log \mathcal{V})^{1/2+o(1)}\right)$ on a single machine [Ope26c, Section 1.1 and Theorem 1.1]. For a $d \times d$ binary matrix the reduction therefore gives $O\left(d^2(\log d)^{1/2+o(1)}\right)$ steps.

When $d = 2^u$, transposition exchanges the u-bit row and column fields of the address. More generally, an *address-field interchange* exchanges two fields of the same width, leaving the other address bits fixed. For every fixed $\varepsilon > 0$, two u-bit fields in an array of $\mathcal{V}$ bits can be exchanged in $O_\varepsilon\left(\mathcal{V} u^{1/2+\varepsilon}\right)$ steps [Ope26c, Theorem 2.1]. We obtain this bound on one machine for all ε (Section 2.3).

Boolean sorting circuits. The sorting network of Ajtai, Komlós and Szemerédi [AKS83] uses $O(n \log n)$ comparators. Comparing two w-bit items and exchanging them according to the outcome costs $O(w)$ Boolean gates, giving a circuit of size $O(nw \log n)$. When the keys are short, the dependence on $\log n$ can be improved. Asharov, Lin and Shi [ALS22] gave circuits that carry associated data and whose size depends on the key width. Lin and Shi [LS22] obtained depth $O(\log n + \log w)$ for short keys with w bits of associated data per item. For n integers of k bits each, Koucký and Král [KK21, Theorem 1] gave a circuit of size $O(nk^2)$. At $k = \Theta(\log n)$ this gives the same $O(n \log^2 n)$ bound as the sorting network.

For keys and associated data each of $\log n$ bits (n a power of two), Asharov, Lin and Shi [ALS22, Theorem 1.4 of the full version] obtained a conditional $\Omega(N \log N)$ gate bound from the undirected

multiple-unicast conjecture, where N counts input bits. Li, Li and Liu subsequently disproved the underlying network coding conjecture [LLL26, Theorem 3.2]. More recently, Koucký, Mertz and Rajgopal [KMR26, abstract] described $\Omega(N \log N)$ as the conjectured lower bound on the circuit size of integer sorting. They study error reduction for randomized sorting circuits, in particular for n integers from $[n^3]$.

Main results.

Theorem 1.1 (Sorting in sub-comparison time)**.** *Let $(k_1, x_1), \ldots, (k_n, x_n)$ be $n \geqslant 1$ pairs of strings of total encoded length N, including delimiters. There exists a deterministic Turing machine with a fixed number of tapes that sorts every such input lexicographically by key using*

$$O\left(\frac{N \log n}{(\log^{(3)} n)^{1/2-o(1)}}\right) \quad \textit{time and} \quad O(N) \quad \textit{space.}$$

In particular, the worst-case running time as a function of N is

$$O\left(\frac{N \log N}{(\log^{(3)} N)^{1/2-o(1)}}\right) = o(N \log N).$$

Pairs with equal keys retain their input order.

The proof appears in Section 5.2. Appendix B shows that two writable tapes suffice.

The same tape bounds hold for signed finite expansions in any fixed base. We encode magnitudes in order of integral digit count and then in digit order, reversing this order for negative values. The encoding in Section 5.2 uses linear time and preserves the original spelling as associated data.

In the fixed-width results, widths are measured in bits. Equal-width binary keys written most significant bit first compare as unsigned integers.

Theorem 1.2 (Fixed-width sorting)**.** *Let $n \geqslant 1$ and let $X_1, \ldots, X_n$ be items of common width $\beta = \Omega(\log n)$ such that the keys occupy the same contiguous field of $O(\log n)$ bits in every item. Let $V = n\beta$ be the input volume. There exists a Turing machine with two writable tapes that outputs the items intact in stable nondecreasing key order on every such input, using*

$$O\left(\frac{V \log n}{(\log^{(3)} n)^{1/2-o(1)}}\right) \quad \textit{time and} \quad O(V) \quad \textit{space.}$$

The proof appears in Section 4.3, with the two-tape implementation in Appendix B.

For any fixed set of $n \geqslant 2$ distinct keys of width $\lceil \log n \rceil$, the minimum width for n distinct values, the input length is $N = \Theta(n \log n)$. On every ordering of these keys, Theorem 1.2 uses $o(N \log N)$ steps. By the movement bound in Section 2.2, every deterministic whole-key comparison sorter uses $\Omega(N \log N)$ steps on some ordering of the same keys.

For the logarithmic-width integers of Hopcroft, Paul and Valiant, $\log^* N = o((\log^{(3)} N)^{1/4})$. Thus Theorem 1.2 meets their sorting hypothesis already for two tapes [HPV75, Theorem 2].

Wider items allow larger radix digits. For every fixed $a > 0$, when the common item width satisfies $\beta \geqslant n^a$, the denominator in Theorem 1.2 improves to $(\log^{(2)} n)^{1/2-o(1)}$, even for keys occupying the entire item. When $\beta \geqslant 2^{n^a}$, the time per input bit is $O_a\left((\log n)^{1/2+o(1)}\right)$. Appendix A gives the dependence on β.

Theorem 1.3 (Sorting by small Boolean circuits). *Let $n \geqslant 2$ and $\beta \geqslant 1$ be integers, and let $V = n\beta$. For each contiguous key field of ℓ bits, where $0 \leqslant \ell \leqslant \beta$, there exists a bounded-fan-in Boolean circuit that sorts every array of n items of width β by this field using*

$$O\left(\frac{V \log n}{(\log^{(4)} n)^{1/2-o(1)}}\right)$$

gates. For each encoded length N, there exists a bounded-fan-in Boolean circuit that sorts every valid delimited stream of that length by key using

$$O\left(\frac{N \log N}{(\log^{(4)} N)^{1/2-o(1)}}\right)$$

gates. Both circuit families preserve the input order of items with equal keys and can be constructed in polynomial time.

The proof appears in Section 6.4. Appendix D reduces delimited streams to fixed-width sorting, giving the second bound. We also use this bound when a fixed-width item is too narrow to hold its position index.

For n integers from $[n^3]$, the problem considered by Koucký, Mertz and Rajgopal [KMR26, Theorem 4.1], the input length is $N = \Theta(n \log n)$. The first bound gives $o(n \log^2 n) = o(N \log N)$ gates, even when values repeat. Suppose each key carries independent data, with $\lceil \log n \rceil$ bits in each field. A *binary selector* copies one of two data bits according to a control bit. If these controls depend only on the keys, a sorting circuit that moves data only through such selectors uses $\Omega(N \log N)$ of them (Appendix E).

Relation to external-memory sorting. A related question arises in external memory, where the cost is the number of block transfers. Aggarwal and Vitter [AV88, Theorem 3.1] gave matching transfer bounds for comparison sorting. Farhadi, Hajiaghayi, Larsen and Shi [FHLS23] obtained a lower bound for integer sorting conditional on the undirected multiple-unicast conjecture. Li, Li and Liu [LLL26, Theorem 3.1 and Corollary E.1] used coded address-field interchange to improve on the comparison bound for sorting logarithmic-width integers in certain external-memory regimes. They use Leighton's Columnsort [Lei85] to sort local subarrays and partition larger inputs by successive groups of high-order key bits [LLL26, Proposition 7.1 and Section 8.2].

After sorting each subarray by the current digit, Li, Li and Liu assemble each bucket by fetching that digit's interval from each subarray in turn [LLL26, proof of Lemma 8.1]. Their I/O bound charges $O(1)$ additional transfers per interval for indexing and partial blocks, regardless of the distance between intervals. A column that fits in memory incurs no further transfers during the local sort between its read and write passes [LLL26, proof of Lemma 7.2]. On tapes, the time for a digit partition includes both the local computation and the travel between intervals.

1.2 Overview of the Sorting Algorithms

Binary radix sorting groups items by a t-bit digit in t passes over the full items. To do better, we arrange the items in a table and reserve the same columns for a given digit in every row. If each row places its items in the columns for their digits, one transposition gathers all 2^t groups. A row may have too many items of a digit to do this for all of them. We rotate the input blocks before distributing their entries over the rows, choosing the rotations so that only an $O(2^{-t})$ fraction overflow. These items can be routed separately.

The remaining task is to permute the items within a row. We split each item into indexed pieces. If item x is assigned slot $\pi(x)$, its pieces may occupy that slot in any order, since their indices let us read the next digit before reassembly. We use this freedom to move the pieces at a common position in all the items by a single cyclic shift. A piece with offset c first moves from x to $a = x + c$, modulo the row length. At a we give it offset $a - \pi(x)$. A shift back by its new offset now sends each piece to its own destination.

For fixed a, the change of offsets $c \mapsto a - \pi(a - c)$ is a permutation. On tape we choose the list of offsets so that it shares a long subsequence with every such relabeling. We can copy the matched pieces in order between the two shifts. The $O(1/t)$ fraction left unmatched needs $O(t)$ ordinary radix passes, whose total cost is linear in the volume of the row. Fast address-field interchange implements the parallel shifts and the final gathering of the digit columns. In the circuit, each piece position has one offset coordinate for each possible π. Fixed wiring changes all coordinates at once, and the shifts use the coordinate indexed by π. To rotate a column of q pieces by s, we view the bits at each common bit position as coefficients of a polynomial over $\mathbb{F}_2$. Multiplying by z^s modulo $z^q - 1$ rotates all q coefficients, so the polynomial multiplication bound applies directly.

On tape we partition by digits of the keys, with input positions appended to break ties. The circuit first solves *permutation routing*. Each item has a distinct prescribed destination, so the size of every group formed by destination digits is known in advance. To find the destinations for a stable digit sort, recursion on shorter keys orders each block locally. The run of digit c in a block starts after all smaller digits and all occurrences of c in earlier blocks. Prefix sums of the local digit counts therefore give the global destinations of the runs. Processing the key digits from least to most significant gives each item its *stable position*, its final slot with ties broken by input order. One last routing sends the full items to these slots.

For n fixed-width items of total volume V with logarithmic-width keys, one digit level costs $O(Vt^{1/2+o(1)})$, including the work of arranging the rows. Over $O(\log n/t)$ levels this gives $O(V \log n/t^{1/2-o(1)})$ tape steps for sorting or gates for routing. The recursive computation of destinations fits the same gate bound. The circuit uses a smaller digit width because its piece positions encode one coordinate for every local permutation.

Remark. Since equal keys occur consecutively in sorted order, comparing adjacent keys decides element distinctness and a longest run of equal keys identifies a mode. Thus the improvement over tape mergesort extends to both problems, even on two writable tapes.[1]

1.3 Organization of the Paper

Section 2 specifies the tape model, formalizes the copying rule by tracking the origin of each output bit and states the address-field interchange used by the algorithm. It also introduces the circuit model and the bounds for routing marked subsets and for cyclic shifts. Section 3 constructs a balanced word that shares a long subsequence with each relabeling. We use these subsequences to route pieces of binary strings and show how to read further key digits while their pieces are out of order.

Section 4 gives the radix step. It chooses the block rotations, gathers the items by digit and accounts for the exceptional items and the terminal mergesorts in the proof of Theorem 1.2. Section 5 reduces permutation routing to sorting and extends the fixed-width algorithm to all key and item widths. In Section 5.2 we sort variable-length items in bins and merge the bins to prove

[1] Ben-Amram, Berkman and Petersen [BABP03] study element distinctness on one tape. Dobkin and Munro [DM80] study mode finding in the comparison model.

Theorem 1.1. Section 6 constructs a circuit for routing to prescribed destinations and uses it recursively to compute stable positions. The fixed-width construction and the reduction for delimited inputs together prove Theorem 1.3. Section 7 asks whether a fixed number of tapes suffices for faster sorting of logarithmic-width items and faster routing of wide strings.

Appendix A routes wide strings through a recursive permutation network, restores the order of their pieces and uses this routing procedure to obtain the width-dependent sorting bound. Appendix B implements the interchange and the sorting passes with two writable tapes. It also gives the linear-time merge of variable-length items used when the sorted bins are combined. Appendix C derives the movement lower bound of Section 2.2 by counting the moves at each bit offset. Appendix D reduces delimited-stream sorting to fixed-width sorting circuits by grouping short keys in batches and ordering the resulting groups together with the long keys. Appendix E gives the selector lower bound and bounds the number of lanes needed by the two-shift construction. Appendix F relates cyclic shifts to stable sorting in Boolean circuits. On tapes it shows that a faster router for logarithmic-width items would give a faster stable sorter, and relates the two open questions.

2 Preliminaries

2.1 The Tape Model

We store arrays in lexicographic order, with the last index varying fastest. The machine is deterministic and has a fixed finite tape alphabet and a fixed number of one-dimensional tapes, each with one head. A step reads and writes at most one symbol per tape and moves each head by at most one cell. In the two-tape implementation both tapes are writable. The first initially holds the input and the second is blank.

For fixed-width inputs, n denotes the number of items, β their common width in bits, and ℓ the width of the common key field. The input volume is $V = n\beta$ bits. The dimensions and the location of the key field are given in binary without leading zeros. The algorithm of Theorem 1.2 carries each item alongside its *augmented key*, a copy of the key followed by the $\lceil \log n \rceil$-bit zero-based input position. This breaks ties in input order and adds $O(\log n)$ bits per item.

For variable-length inputs, a delimiter follows each key and each associated string. Either may be empty. The input length includes all symbols and delimiters. A fixed-width binary encoding changes this length by a constant factor, so we use N in either unit. Tape time includes preprocessing, head movements and output, and space is measured across all tapes. A tape bound expressed solely in terms of N is a worst-case bound over encodings of length at most N.

2.2 The Cost of Copying Input Bits

Definition 2.1 (Source labels)**.** Bit j of input occurrence i has *source label* (i, j). Local copies inherit their source label, and bits computed by arithmetic or comparison have no label. The copying rule says that bit j of occurrence i still has label (i, j) in the output.

The *rank* of a key in a list is the number of distinct smaller key values in that list. The sorter may use ranks and indices to direct its copies.

Hopcroft, Paul and Valiant [HPV75, p. 62] applied the rearrangement bound of Stoß [Sto73] to sorting integers moved as units. The counting argument extends to the copying rule of Definition 2.1, which allows the bits of a key to move separately. Under this rule, for every fixed set of $n \geqslant 2$ distinct w-bit keys, some input order costs $\Omega(wn \log n)$ steps (Appendix C).

2.3 Interchanging Two Address Fields

Let P, G, B, u be positive integers. In a binary array indexed in tape order by $[P] \times [2^u] \times [G] \times [2^u] \times [B]$, the *address-field interchange* moves each bit by exchanging the two u-bit fields of its address,

$$(p, a, g, b, z) \longmapsto (p, b, g, a, z). \tag{1}$$

The bit volume is $\mathcal{V} = PGB2^{2u}$. For each fixed $\alpha > 1/2$, OpenAI [Ope26c, Theorem 2.1] gives a multitape machine that performs (1) in $O_\alpha(\mathcal{V}u^\alpha)$ steps and visits $O_\alpha(\mathcal{V})$ cells. These bounds are uniform in the prefix, gap and suffix ranges P, G, B, and include the removal of padding.

Appendix B.2 performs the interchange on a single two-tape machine in $O\left(\mathcal{V}u^{1/2+o(1)}\right)$ time and $O(\mathcal{V})$ space. At width u it selects an interchange program from an effective sequence with exponents decreasing to $1/2$. Processing separately the slices obtained by fixing the high bits of both fields keeps the space constant independent of the selected program.

2.4 Boolean Circuits

A circuit is a finite acyclic network over a fixed complete Boolean basis of fan-in at most two. Its size is the number of gates, including those that compute data-dependent routing decisions. A fixed permutation of wires and arbitrary fan-out are free. Using another fixed complete basis of bounded fan-in changes the size by at most a constant factor. A bound of two on fan-out can also be imposed by inserting binary copy trees, at a cost linear in the number of gates and outputs. For a fixed input length, the circuit has a prescribed number of input and output bits. For fixed-width sorting, the parameters are the number n of items, their width β, and the location and width ℓ of the key field, as in Section 2.1. The circuit has $V = n\beta$ input bits. For delimited inputs, the encoded stream length N determines the circuit. The input and output use the encoding of Section 2.1, with one delimiter after each key and one after its associated string. A polynomial-time construction gives a deterministic algorithm that writes the circuit description in time polynomial in the input length given in unary and, for fixed-width inputs, its dimensions.

We use a few elementary circuit operations. Comparison, addition and a controlled exchange on b-bit words each cost $O(b)$ gates. For an associative operation, an upward reduction in a binary tree followed by a downward pass computes all prefix values on s inputs in $O(s)$ applications of the operation. A segment-start flag gives the corresponding scan within each segment. These scans include prefix sums and propagation of the most recent marked word. An ordinary sorting network sorts s words of width b in $O(sb \log s)$ gates [AKS83].

Routing marked subsets. A superconcentrator on s inputs and s outputs connects any two marked subsets of the same cardinality by vertex-disjoint paths. The bijection between the subsets can be chosen by the routing circuit. Pippenger's self-routing superconcentrator [Pip96] has bounded degree and $O(s)$ edges. Given the two membership masks, its settings can be computed in $O(s \log s)$ gates [KK21, Theorem 17 and Corollary 18]. With the settings fixed, passing words of b bits along the paths costs $O(sb)$ gates. Thus one connection, including its settings, costs

$$O\big(s(b + \log s)\big). \tag{2}$$

The paths can also be traversed in reverse with the same settings. In particular, if j inputs are marked, choosing the first j outputs packs the selected words in some order. Both the network and its routing circuit can be constructed uniformly [HR24, Corollary 19].

Cyclic shifts. Let q be a power of two. A circuit receives q words of b bits and a binary offset $s \in [q]$. The bits at a common position in the q words form a *bit plane*. Its q bits are the coefficients of a polynomial over $\mathbb{F}_2$. A cyclic shift by s is multiplication by z^s modulo $z^q - 1$. The sole nonzero coefficient of z^s is at position s. The coefficient vector is the one-hot encoding of s and can be computed in $O(q)$ gates by successively extending binary prefixes. OpenAI's bound for division-free polynomial multiplication over a field [Ope26b, Corollary 8.2] applies over $\mathbb{F}_2$, where addition and multiplication are XOR and AND. Folding the product coefficients modulo $z^q - 1$ costs $O(q)$ gates per plane. For each fixed $0 < \varepsilon < 1/2$, the shift therefore has size

$$O_\varepsilon\left(qb(\log q)^{1/2+\varepsilon}\right). \tag{3}$$

3 Local Permutations of Binary Strings

3.1 A Balanced Word for All Relabelings

The routing procedure assigns an offset to each piece position. In each intermediate row a permutation of the items induces a relabeling of these offsets. A long common subsequence between the original and relabeled words allows most pieces to be copied in order.

Definition 3.1 (Balanced words)**.** Let q, M be positive integers. A word $W : [M] \to [q]$ is *balanced* if every letter of $[q]$ occurs M/q times. For a permutation γ of $[q]$, the *relabeling* γW is the word with $(\gamma W)(y) = \gamma(W(y))$ at each position $y \in [M]$. For a finite set X, we write $\mathrm{Sym}(X)$ for its permutation group.

A *subsequence matching* pairs equal letters in two words, in increasing order of position in both. For words U, V of the same positive length, let $d(U, V)$ be the smallest fraction of positions left unmatched in either word. A longest common subsequence attains this minimum.

Lemma 3.2. *Let $q \geqslant 2$ be an integer and let $0 < \varepsilon \leqslant 1/2$. There exists a balanced word W of length M over $[q]$ satisfying*

$$d(W, \gamma W) \leqslant \varepsilon \quad (\gamma \in \mathrm{Sym}([q])), \qquad \log M = O\left(q^2(\log(q/\varepsilon))^2\right).$$

Moreover, each of the M/q successive blocks of q letters in W contains every letter once. A fixed multitape Turing machine constructs W from q and $\lceil 1/\varepsilon \rceil$ in $(qM)^{O(1)}$ time and space. With a permutation $\gamma \in \mathrm{Sym}([q])$ as an additional input, the same machine finds a matching of length at least $(1-\varepsilon)M$ between W and γW within these bounds.

Proof. The initial word lists the q letters once. From the current word W we form W' by concatenating one section for each transposition s, in a fixed order. The section for s alternates between W and sW for $4q\lceil 1/\varepsilon \rceil$ copies. Relabeling and concatenation preserve the property that each successive block of q letters contains every letter once, so W' is balanced.

Let $D(W)$ be the maximum of $d(W, gW)$ over transpositions g. To match W' with gW', we compare the corresponding sections. Outside the g-section, each pair of corresponding copies has unmatched fraction at most $D(W)$. For copies sW and gsW this follows by relabeling both by s^{-1}, since $s^{-1}gs$ is a transposition. In the g-section we pair the alternating copies one position apart. Every paired copy then agrees in full. One copy in each word remains unmatched, giving

an unmatched fraction of at most $\varepsilon/(4q)$ in this section. The sections have equal length and their matchings concatenate in order. Hence

$$D(W') \leqslant D(W) - \frac{D(W) - \varepsilon/(4q)}{\binom{q}{2}}.$$

By $1-x \leqslant e^{-x}$, a round of $\binom{q}{2}$ iterations reduces any positive excess of $D(W)$ over $\varepsilon/(4q)$ by at least half. Initially $D(W) \leqslant 2/q$, as we may delete the two letters moved by a transposition. After $2 + \lceil \log(1/\varepsilon) \rceil$ rounds we therefore have $D(W) \leqslant \varepsilon/(4q) + (2/q)(\varepsilon/4) < \varepsilon/q$. Every permutation is a product of at most $q-1$ transpositions. Composing matchings on the common positions of the middle word gives the triangle inequality for d. Since simultaneous relabeling preserves d, we obtain $d(W, \gamma W) \leqslant (q-1)D(W) \leqslant \varepsilon$ for every $\gamma \in \mathrm{Sym}([q])$.

Each iteration multiplies the length by $4q\lceil 1/\varepsilon \rceil \binom{q}{2}$. There are $O(q^2 \log(1/\varepsilon))$ iterations, giving $\log M = O\left(q^2 \log(q/\varepsilon) \log(1/\varepsilon)\right)$ as claimed. The machine computes the round count by rounding its integer input $\lceil 1/\varepsilon \rceil$ up to a power of two. The construction uses sequential relabeling and concatenation. For a given relabeling, the ordinary longest-common-subsequence dynamic program and traceback find the matching. With sequential scans for table access, this computation runs in time polynomial in qM. □

For fixed q, the existence of balanced words over $[q]$ arbitrarily close to all their relabelings also follows from Best and Peres [BP25, Theorem 1]. For any ordering $g_1, \ldots, g_{q!}$ of $\mathrm{Sym}([q])$, the substitution sends each $a \in \mathrm{Sym}([q])$ to $(ag_1, \ldots, ag_{q!})$. Each image contains every permutation once, so the substitution is primitive and of constant length. It also commutes with left multiplication. Its iterates from a and γa therefore have common subsequences covering a fraction tending to one by their theorem. Projecting each group letter g to $g(0)$ gives balanced words related by γ and preserves those matches. The bound on $\log M$ in Lemma 3.2 lets us use the construction when q grows with the input.

3.2 Routing the Pieces of a Binary String

Let $q \geqslant 2$ be a power of two. Row indices and offsets are computed modulo q. Let W be the word of Lemma 3.2 for $\varepsilon = 1/\max(2, \log q)$, and let M be its length.

For a positive integer b, the array has coordinates $(x, y, \nu, z) \in [q] \times [M] \times [q] \times [b]$. In row x, the qb consecutive bits at position y form a *lane*, with ν selecting one of its q parts and z a bit within that part. The *color* of position y is $W(y)$, the offset by which a piece at that position shifts across rows.

Thus the row width is qMb bits and the array volume is $\mathcal{V} = q^2Mb$ bits.

Lemma 3.3. *Let the array of bits have coordinates $(x, y, \nu, z) \in [q] \times [M] \times [q] \times [b]$ as above. There exists a multitape Turing machine that, given the array and any permutation π of $[q]$, realizes a bijection of bit positions of the form*

$$(x, y, \nu, z) \longmapsto (\pi(x), \lambda_x(y), \nu, z), \tag{4}$$

where each λ_x is a permutation of $[M]$. A source lane position is matched *if the subsequence matching in its intermediate row pairs it with a target position, and* unmatched *otherwise. This designation and the permutations λ_x depend only on W and π. Within each input row and color, λ_x preserves the relative order of the matched positions and that of the unmatched positions. There is an absolute constant D such that, for row widths $qMb \geqslant (qM)^D$, the machine runs in $O\left(\mathcal{V}(\log q)^{1/2+o(1)}\right)$ time and uses $O(\mathcal{V})$ space.*

Proof. The first stage shifts the row index by the color of the lane, $x \leftarrow x + W(y)$. Interchanging x with v by (1) gives the layout (v, y, x, z), in which each fixed (v, y) has a contiguous fiber of q cells of width b. A rotation by $W(y)$ cells splits this fiber into two consecutive segments and copies them in the opposite order. A second interchange restores the field order. The rotations scan linear volume. The inverse shift uses subtraction in the same layout.

In the intermediate row $a = x + W(y)$, a lane of color $c = W(y)$ comes from row $a - c$ and has destination $\pi(a - c)$. Since the inverse shift subtracts the target color from a, that color must be

$$\phi_a(c) = a - \pi(a - c). \tag{5}$$

This is a permutation of the colors. It suffices to permute lanes within row a by a bijection f_a satisfying $W(f_a(y)) = \phi_a(W(y))$.

By Lemma 3.2, we can compute a subsequence matching of $\phi_a W$ and W of length at least $(1 - \varepsilon)M$. The matching sends each matched source position to its paired target position. Since W is balanced (Definition 3.1), every color occurs M/q times in both W and $\phi_a W$. The unmatched occurrences of each color can therefore be paired in order. A fixed traceback rule makes both pairings depend only on W and π.

To implement f_a on tapes, a sequential mask scan separates the matched and unmatched source lanes, preserving the order of the matched stream. A stable binary split places the lanes with a zero in the current bit before those with a one, preserving the order within each group. Applying these splits from the least to the most significant destination-color bit groups the unmatched lanes by color. For the vacant target positions, we compute the masks of the stable radix passes that would group these positions by color. Applying their inverses in reverse order to the grouped source stream places the unmatched lanes in the order of the vacant positions. Each binary split or inverse split uses a constant number of scans (Appendix B.1). A final scan interleaves the matched and unmatched lanes according to the target mask. Over all rows, the full-row scans cost $O(\mathcal{V})$ and the radix passes cost $O(\varepsilon \mathcal{V} \log q) = O(\mathcal{V})$, since they act only on the unmatched lanes.

Reversing the first shift sends the lane to row $a - W(f_a(y)) = \pi(a - W(y)) = \pi(x)$. All three stages are bijections and preserve v, z at their endpoints, giving (4). The lanes of one input row and color pass through the same intermediate row, where each pairing preserves their relative order within its matched or unmatched subsequence.

The masks, radix schedules and rotation offsets can be constructed in time polynomial in qM. Reusable patterns of marks on tape identify the segment boundaries, with each scan charged to the bits copied. Across all fibers the number of lengths and resets is polynomial in qM, and each uses arithmetic polynomial in $\log(qMb)$. Their total cost is at most $(qM)^C (\log(qMb))^C$ steps for a fixed C. For $D \geqslant 2C$ the width hypothesis bounds the first factor by $(qMb)^{1/2}$, and the second is $O\left((qMb)^{1/2}\right)$. Hence this cost is $O(qMb)$. Increasing D if necessary also bounds the polynomial cost of constructing and storing the masks and schedules by the row width qMb. The time bound follows from Section 2.3, and the row workspaces are reused successively. □

During sorting an item and its augmented key form the *augmented item*. We pad it at the end and divide it into M equal data pieces. We prefix each piece by its *original lane number* in $\lceil \log M \rceil$ bits and pad the resulting lane width to a multiple of q. This number travels with the piece, while its color $W(y)$ depends on its current position. The external header, which holds the flags and current radix digit, follows the same permutation π as its item. Scans of the q headers suffice in time polynomial in q and the digit width. Between local permutations each encoded item moves as a whole.

If the original data-piece width is at least t, a t-bit digit meets at most two pieces. From its logical offset we compute their original lane numbers, then recover the digit in order by scanning the lanes once. The offset calculations and label comparisons cost at most $(qM)^{O(1)}(\log \beta)^{O(1)}$ steps per item. At the widths used in Section 4.3, these calculations and the scan cost $O(\beta)$ time. After a bucket's last digit, or just before its terminal mergesort, stable binary radix passes on the lane numbers restore the original piece order in $O(\beta \log M)$ time per item. The machine writes the restored augmented item at the front of its slot and pads the rest, so that completed buckets retain their width.

4 Radix Sorting

For the fixed-width items of Theorem 1.2, we read the augmented keys from the most significant end in digits of t bits. A *bucket* consists of the items with a common prefix of processed digits. As in the high-order recursion of Li, Li and Liu [LLL26, Section 8.2], one step partitions each bucket by the next digit, placing its children consecutively in digit order.

4.1 Distributing One Digit

For radix $K = 2^t$ with $t \geqslant 2$, let $L = K^2$. We consider a bucket of $m \geqslant L^2$ items in arbitrary order. Let $R = 2^\nu$ be the least power of two with $LR \geqslant m$. Then $R \geqslant L$ and $m \leqslant LR < 2m$. Padding with dummies gives L consecutive blocks of R items each. The digit stream determines one cyclic shift per block, which the machine applies to the full items. Row $r \in [R]$ then consists of the item at position r in each shifted block.

Definition 4.1 (Excess items)**.** For a digit $c \in [K]$, let m_c be the number of items with digit c in the bucket and $\mu_c = m_c/R$ their mean count per row. Each row reserves s_c slots of K item positions for c, where

$$s_c = \lceil 2\mu_c/K \rceil + 1. \tag{6}$$

The capacity for c is therefore Ks_c items. For a choice of block shifts, let $X(r, c)$ count the genuine items with digit c in row r. Their *excess* in this row is $(X(r, c) - Ks_c)_+$, where $(x)_+ = \max\{x, 0\}$. The *total excess* E is the sum over all rows and digits.

Binary radix sorting a copy of the m digits gives the counts m_c in $O(mt^2)$ time, using amortized binary increments within each run. With the width choice of Section 4.3, digit extraction and counting together cost $O(m\beta)$ time. Since $\sum_c \mu_c \leqslant L$, the capacities use $\sum_c s_c \leqslant 4K$ slots. The values s_c can be computed from m_c, R, K by integer arithmetic, comparing products to find quotients of size $O(K)$.

Lemma 4.2. *Let $K = 2^t$ with $t \geqslant 2$ and let a bucket of $m \geqslant K^4$ items be arranged in blocks as above. There exists a deterministic multitape Turing machine that chooses one cyclic shift per block so that the total excess is $E = O(m/K)$. The machine computes the shifts from the digit streams in $O(mKt^{O(1)})$ steps and uses $O(m\beta t^{1/2+o(1)})$ further steps to rotate the blocks and write the items in contiguous rows.*

Proof. Let d_c be the least power of two at least $2L/(\mu_c + K)$. Since $\mu_c \leqslant L$ and $K \leqslant L$,

$$1 \leqslant d_c \leqslant 4K, \qquad 2L \leqslant d_c(\mu_c + K) \leqslant 4L. \tag{7}$$

Comparisons of integer products also determine these weights from m_c, R, K. We use the weighted sum

$$\Phi = \sum_{r,c} d_c (X(r, c) - \mu_c)^2.$$

Under independent uniform shifts, $X(r,c)$ is a sum of independent zero-one variables with mean μ_c and variance at most μ_c. Hence $\mathbb{E}[\Phi] \leqslant R \sum_c d_c \mu_c$.

We fix one block shift at a time. Let $S(r,c)$ count the contributions of the fixed blocks, and $B(r,c)$ be the unshifted indicator for the current one. Dummies contribute zero for every digit class $c \in [K]$. Under a shift $\ell \in [R]$, the current block contributes $B(r+\ell, c)$, with row indices computed modulo R. Up to the common factor two, the only term of the conditional expected potential depending on ℓ is

$$\sum_{r,c} d_c S(r,c) B(r+\ell, c). \tag{8}$$

Indeed the sum over r of each shifted indicator and its square is fixed. The expected contribution of the remaining blocks to $X(r,c)$ depends only on c. Its cross term with the shifted indicator is consequently independent of ℓ. Choosing a shift for which the sum in (8) is at most its mean preserves the conditional expectation bound.

For $0 \leqslant j \leqslant \nu$, the residue histograms are

$$S_j(u,c) = \sum_{r \equiv u \pmod{2^j}} S(r,c), \qquad B_j(u,c) = \sum_{r \equiv u \pmod{2^j}} B(r,c).$$

If ℓ is uniform among shifts congruent to e modulo 2^j, the mean of (8) is

$$\frac{2^j}{R} \sum_{u \in [2^j],\, c \in [K]} d_c S_j(u,c) B_j(u+e,c). \tag{9}$$

Here $u+e$ is read modulo 2^j. For $j = 1, \ldots, \nu$, we retain the extension of the current residue with the smaller score. The two scores average to the preceding conditional mean, so the final shift satisfies the desired inequality.

The tables are stored in (u,c) order. Expanding each position of the current block into its K indicator entries gives B_ν. For each table, adding the two contiguous halves entrywise produces the next coarser level. We keep the levels in successive tape segments and read them back from coarse to fine. At level j, each residue has $R/2^j$ positions and $S(r,c) \leqslant L$, so a count has $O(t+\nu-j)$ bits. For either candidate e, a cyclic shift by e groups of K entries aligns the two tables. A scan multiplies corresponding counts, reading the K power-of-two weights in order at each residue. Pairwise summation halves the number of terms in each round while adding at most one bit to their width.

For a fixed constant a, the total time for one block is bounded by

$$O\left(K \sum_{j=0}^{\nu} 2^j (t+\nu-j)^a\right) = O\left(RKt^a\right). \tag{10}$$

Computing the table dimensions at each level also fits this bound, since $\nu^{O(1)} = O(2^\nu)$. A scan adds the rotated block to S_ν for the next block. The tables occupy $O\left(K \sum_{j=0}^{\nu} 2^j (t+\nu-j)\right) = O(RKt)$ bits, reused across the L blocks. Since $LR < 2m$, (10) gives the claimed time for computing the shifts.

With these choices $\Phi \leqslant R \sum_c d_c \mu_c$. For real x and $T > \mu$, the inequality $(x-T)_+ \leqslant (x-\mu)^2/(4(T-\mu))$ follows by completing a square. Equations (6) and (7) give

$$E = \sum_{r,c} (X(r,c) - Ks_c)_+ \leqslant \frac{\Phi}{8L} \leqslant \frac{R}{8L} \sum_c d_c \mu_c \leqslant \frac{RK}{2} = O(m/K). \tag{11}$$

Finally, the machine rotates each block by splitting it at the chosen offset and concatenating the two segments in the opposite order. The rotations cost $O(m\beta)$ time. With positions within a block written as $uL + s$, the divisibility $L \mid R$ gives the tape layout (a, u, s), where the block index a and the offset s both lie in $[L]$. Interchanging a and s by (1) gives (s, u, a). Each consecutive group of L entries contains one item from each block, all at the same position $uL + s$ after shifting. The field width is $2t$ and the bit volume is $O(m\beta)$. □

4.2 Gathering Items by Digit

Each row has $4K$ slots, each holding K complete items. The slots for classes $0, 1, \ldots, K-1$ occur in successive intervals, with s_c consecutive slots for class c. The surplus slots contain dummies. All R rows of the bucket use the same slot intervals.

Padding a row with dummies gives $q = 8L = 8K^2$ item positions. For each class c, its first $\min(X(r, c), Ks_c)$ genuine items occupy successive reserved positions in occurrence order, all within the first $4L$ positions. The excess items go to consecutive positions at the start of the last L, in occurrence order. The dummies fill the vacancies in order. Since the original row has at most L genuine items, these assignments define a permutation. Its table is computable in time polynomial in q from the row's digits, dummy flags and slot counts.

Each of the q items in this padded row occupies one row of the array in Lemma 3.3. We route the encoded items and their headers as in Section 3.2, then copy the first $4L$ positions, including dummies, to the main stream and only genuine excess items to the overflow stream. The main stream has $4RL = O(m)$ item positions.

To gather the slots by class, we number the rows in tape order by $aR/(4K) + u$, where $a \in [4K]$ and $u \in [R/(4K)]$. This is possible since K and R are powers of two and $R \geqslant K^2 \geqslant 4K$. Writing s for the slot number and z for a bit position within the slot, the main stream has layout (a, u, s, z). Interchanging a and s gives (s, u, a, z). Since every row reserves the same slot interval for each digit, all slots of a class are now contiguous. The two interchanged fields have width $t + 2$, so (1) costs $O\left(m\beta t^{1/2+o(1)}\right)$ time.

A scan removes the dummies and retains all class boundaries, including those of empty classes. On the compact overflow stream, t stable binary splits by the current digit cost $O(m\beta t/K) = O(m\beta)$ by (11). We append each overflow class to the corresponding main class and discard empty children. The resulting children occur in ascending digit order. Within a child the remaining digits determine the order, since the augmented keys are distinct.

4.3 Proof of Theorem 1.2

Armed with the balanced word of Lemma 3.2, the routing procedure of Lemma 3.3, and the excess bound of Lemma 4.2, we are now ready to prove Theorem 1.2.

Proof. For items of width $\Theta(\log n)$, the M lanes use at least $qM \geqslant q^2$ bits per item. Thus $q = 8 \cdot 2^{2t}$ already gives $t = O\left(\log^{(2)} n\right)$ in linear space. To account for the time to compute the matchings, for all sufficiently large n we set

$$t = \lfloor \tfrac{1}{16} \log^{(3)} n \rfloor, \qquad K = 2^t, \qquad q = 8K^2. \tag{12}$$

Integer bit lengths computed from the input dimensions change t by at most $O(1)$. Below a fixed threshold the machine uses ordinary mergesort.

Here $q = O\left((\log^{(2)} n)^{1/8}\right)$ and $\log q = 2t + 3$. Applying Lemma 3.2 with tolerance $1/\log q$ gives

$$\log M = O\left(q^2(\log q)^2\right) = o(\log^{(2)} n), \qquad (qM)^{O(1)} = (\log n)^{o(1)}. \tag{13}$$

With the key and input position appended, the item width is at least $\log n$ and is $O(\beta)$ under the hypotheses of Theorem 1.2. By (13), this width exceeds $(qM)^D$ for all sufficiently large n, as needed in Lemma 3.3. The word and the instructions for routing one padded row are computable in $(qM)^{O(1)} = O(\beta)$ time. The balancing cost $Kt^{O(1)}$ per item is also $O(\beta)$.

Each of the M pieces has an $O(\log M)$-bit label and fewer than q padding bits. Since $\log M \geqslant q$ for the word constructed in Lemma 3.2, the total overhead, including the t-bit header, is $O(M \log M)$ bits per item. The encoded width is therefore $O(\beta)$ and each data piece contains at least t bits for large n. Digit extraction costs $O(\beta)$ per item, since $(qM)^{O(1)} = (\log n)^{o(1)}$ and every fixed power of $\log \beta$ is $O(\sqrt{\beta})$. Counts local to a bucket of m items use $O(\log m)$ bits, and polynomial time in this width once per bucket is $O(m)$.

The radix step of Sections 4.1 and 4.2 processes successive digits, padding the last digit with trailing zeroes if necessary. A bucket of $m < L^2$ items has $\log m < 4t$. After its pieces are restored to their original order, ordinary tape mergesort orders the bucket by its remaining key bits in $O(m\beta t)$ time. Its slots pass through later levels in order. All other items regain their original piece order after the last digit. Thus each item undergoes one restoration and at most one terminal mergesort. The augmented keys break ties by input position, and a final scan writes the original items from their slots.

Let $0 < \varepsilon < 1/2$. There are $O((\log n)/t)$ levels, and the active parents at each level contain at most n items in total. The routing procedure of Lemma 3.3 costs $O_\varepsilon\left(t^{1/2+\varepsilon}\right)$ time per bit, as do the two global interchanges within a parent. Copying the completed buckets costs $O(V)$ per level. The total time over all levels is $O_\varepsilon\left(V \log n / t^{1/2-\varepsilon}\right)$. The terminal sorts cost $O(Vt)$ and restoring the piece order costs $O(V \log M)$. Both are $o(V \log n / t^{1/2-\varepsilon})$, since $t = \Theta\left(\log^{(3)} n\right)$ and $\log M = o(\log^{(2)} n)$. The same machine satisfies this estimate for every fixed ε, giving the running time

$$O\left(\frac{V \log n}{(\log^{(3)} n)^{1/2-o(1)}}\right).$$

The current level and its output occupy $O(V)$ bits. On a fixed number of tapes we keep the outer heads at the parent boundaries and reuse a work area for each parent in turn. Its padded items, the two interchanges and the main and overflow streams occupy $O(m\beta)$ bits. The histograms use $O(RKt) = O(m\beta)$ bits, and the routing instructions use $(\log n)^{o(1)}$ bits. Each rewind costs at most the scan of the same region. This gives $O(V)$ space. Appendix B implements the passes on two writable tapes within the same bounds. □

5 Permutation Routing and General Inputs

In *permutation routing*, the input gives strings $X_0, \ldots, X_{n-1}$ and their destinations $\pi(0), \ldots, \pi(n-1)$ in source order, where π permutes $[n]$. The output strings satisfy $Y_{\pi(i)} = X_i$ for all $i \in [n]$.

Schnorr's Program p_2 reduces permutation routing to sorting [Sch78, p. 139]. For $n \geqslant 2$ strings of width $\beta \geqslant 1$, the $\lceil \log n \rceil$-bit destination of each string serves as its key. Sorting by these keys with Theorem 1.2 and discarding them in a final scan leaves the strings intact in destination order. For the total input length $N = \Theta(n(\beta + \lceil \log n \rceil))$, the algorithm uses $O\left(N \log n / (\log^{(3)} n)^{1/2-o(1)}\right)$ time and $O(N)$ space. If $\beta = \Omega(\log n)$, the strings alone occupy

$\Theta(N)$ bits. For logarithmic-width items, Proposition F.2 gives a converse by computing each item's position in sorted order, breaking ties by input order.

5.1 Removing the Width Restrictions

Corollary 5.1. *Let $X_1, \ldots, X_n$ be items of common width $\beta = \Omega(\log n)$, where $n \geqslant 1$. Suppose their keys occupy the same contiguous field of ℓ bits, with $0 \leqslant \ell \leqslant \beta$. There exists a Turing machine with two writable tapes that stably sorts every such input within the time and space bounds of Theorem 1.2.*

Proof. For $n = 1$ a scan suffices. Let $h = \lceil \log n \rceil$ and $n' = 2^h$. For $\ell \leqslant h$, Theorem 1.2 applies directly. For $\ell > h$ we retain the original items in input order and compute an h-bit rank for each key. The first n rows of a table contain the input keys, and copies of the first key fill the remaining $n' - n$ rows. Each row consists of h-bit chunks in order from most to least significant, with zeros on the right of the last chunk if needed. Appending zero chunks until the number of columns is a power of two gives a table of $O(n\ell)$ bits.

We rank successive pairs of blocks by doubling their length, as in the naming method of Karp, Miller and Rosenberg [KMR72, Section II, Algorithm 1]. Initially a chunk is its own label. In each round we pair adjacent labels in each row. Within each column of pairs, we attach to each pair its position in the physical column. A lexicographic sort by the two labels with Theorem 1.2 brings equal pairs together. A scan assigns consecutive ranks starting at zero to the distinct pairs, and a sort by the attached positions restores the column order. These ranks order the concatenated blocks within each column. This suffices for the next round, since corresponding halves of two blocks always come from the same column. In the final column the labels are the ranks of the full keys.

For the column sorts, let D' be the number of pairs in a row. If $D' \geqslant n'$, an address has the form (i, g, j) with $i, j \in [n']$ and (g, j) the pair position. The interchange $(i, g, j) \mapsto (j, g, i)$ makes each column contiguous. If $1 < D' < n'$, the row index has the form (a, b) with $a \in [D']$, and the interchange $(a, b, j) \mapsto (j, b, a)$ gives physical row order (b, a) within each column. If $D' = 1$, the column is already contiguous. In all three cases the positions within a column fit in h bits. Reversing the interchange after the two sorts restores row-major order for the next round.

There are $O(\ell/h)$ initial chunks per row, and each round halves the number of labels. Over all rounds we process $O(n\ell/h)$ pairs of $O(h)$ bits. For every fixed $0 < \varepsilon < 1/2$, Theorem 1.2 bounds the two sorts in each column by $O_\varepsilon\left(h/(\log^{(3)} n)^{1/2-\varepsilon}\right)$ steps per bit. The interchanged fields have at most h bits, so each interchange costs $O_\varepsilon\left(h^{1/2+\varepsilon}\right)$ steps per bit by Section 2.3, which fits the same bound since $\log^{(3)} n \leqslant h$. Thus rank computation uses $O_\varepsilon\left(n\ell h/(\log^{(3)} n)^{1/2-\varepsilon}\right)$ time, with $O(n\ell)$ space in addition to the original items. The first n ranks correspond to those items in input order. We attach each rank to its item and sort once more by Theorem 1.2. The added rank has width $h = O(\beta)$, and stability of the last sort preserves the input order at each rank. Appendix B.3 implements the column operations on two writable tapes. □

For smaller item widths we use binary radix sorting or combine sorted batches. In the remaining reductions we use the eventual monotonicity of $\log x/(\log^{(3)} x)^{1/2-\varepsilon}$ for fixed $0 < \varepsilon < 1/2$. Thus the time bound per bit for an n-item sort also applies to each list of $O(n)$ items.

Corollary 5.2. *Let $n \geqslant 2$ and let β, ℓ be integers such that $\beta \geqslant 1$ and $0 \leqslant \ell \leqslant \beta$. Suppose the input consists of n items of width β whose keys occupy the same contiguous ℓ-bit field, and let $V = n\beta$. There exists a Turing machine with a fixed number of tapes that stably sorts every such input in $O\left(V \log n/(\log^{(3)} n)^{1/2-o(1)}\right)$ time and $O(V)$ space, uniformly in β and ℓ.*

Proof. Here $h = \lceil \log n \rceil$. Empty keys need only a scan. For $\beta \geqslant h/4$, Corollary 5.1 applies. For $\beta \leqslant 2 \log h$, ordinary stable least-significant-bit radix sorting uses at most β linear-volume passes. Its time $O(V\beta)$ fits the bound. It remains to consider $2 \log h < \beta < h/4$.

We divide the input into consecutive batches of $r = 2^{2\beta}$ items, padding the last batch to r with flagged dummies as needed. Since $r < 2^{h/2} \leqslant \sqrt{2n}$, the padded batches contain $O(n)$ items altogether. We sort each batch stably by key using Theorem 1.2 and discard the dummies. The item width with the padding flag is $\Theta(\beta) = \Theta(\log r)$. The total batch volume is $O(V)$, and $r \leqslant n$, so these sorts fit the stated time and space bounds.

A sorted batch consists of at most $2^\ell \leqslant 2^\beta$ runs of equal keys. Let s be the total number of runs, counting a new run at every batch boundary. Then

$$\sqrt{n/2} \leqslant n/r \leqslant s \leqslant \lceil n/r \rceil 2^\beta \leqslant 2n\,2^{-\beta}.$$

Thus $\log s = \Theta(\log n)$ and $sh = O(V)$, since $2^\beta > h^2$. As the machine writes the batches consecutively, it marks the start of each nonempty batch and each change of key within it. A binary counter counts these marks in amortized constant time per increment.

With $p = \lceil n/s \rceil$, a second scan divides each run into consecutive pieces of p items, padding its last piece as needed. The number s' of pieces satisfies $s \leqslant s' \leqslant s + n/p \leqslant 2s$. The header of a piece contains its key and the number of unpadded items, each in $O(h)$ bits. Padding the key field to h bits makes the physical width $\Omega(\log s')$, while the key width is $O(\log s')$. Since $p \geqslant 2^{\beta-1} > h^2/2$ and $s'p = O(n)$, the total volume is $O(V)$. Binary division computes p in $h^{O(1)}$ steps. Initializing a counter for each run or piece costs $O(sh)$ time in total. We copy the items and decrement the counters in time linear in the volume scanned.

We sort the pieces by key alone using Theorem 1.2. Since $\log s' = \Theta(\log n)$ and $s' \leqslant 2n$, the time and space fit the claimed bounds. A final scan outputs the unpadded prefix of each piece. Equal-key pieces remain in their input order, both within a run and across consecutive batches. The items within each piece also remain in order. □

5.2 Proof of Theorem 1.1

Corollary 5.1 allows keys of arbitrary width, and Corollary 5.2 removes the lower bound on the common item width. With these extensions of Theorem 1.2, we can now prove Theorem 1.1.

Proof. For $n = 1$ a scan suffices. An initial scan determines N and n, from which we compute $T = \lceil N/n \rceil$. Bin $j \geqslant 0$ has capacity $H_j = 2^j T$. If key i has length ℓ_i, its pair belongs to the first bin with $1 + \ell_i \leqslant H_j$. Since $1 + \ell_i \leqslant N \leqslant nT$, there are at most $1 + \lceil \log n \rceil$ bins. Summing H_0 over the pairs in bin zero gives at most $nT = O(N)$. In every other bin $H_j < 2(1 + \ell_i)$ for each assigned pair. Hence the sum of the assigned capacities over all pairs is $O(N)$.

A stable binary radix sort distributes the pairs by bin index. The tag for pair i uses $O\left(\log^{(2)} \ell_i\right)$ bits. Sequential length counts give the tags in time polynomial in $\log \ell_i$ and $\log T$ per pair, hence in $O(\sum_i (\ell_i + T)) = O(N)$ time overall. A scan attaches each tag in least-significant-bit-first order. Each pass removes the low bit from a working copy, treating an exhausted copy as zero. The original tags mark the bins after $O\left(\log^{(2)} n\right)$ passes of $O(N)$ time each.

We number the input symbols from one in increasing order and encode each number in a fixed width of b bits, most significant bit first. In bin j, a zero block terminates each key and trailing zeros fill the remaining bits up to width bH_j. At the first differing symbol, the codes compare in the order of the symbols. If one key is a proper prefix of another, its zero block precedes the next symbol of the longer key. For each nonempty bin let n_j be its number of pairs and N_j the total bit

length of their original encodings together with one key of bH_j bits per pair. Since b is constant, the capacity bound gives $\sum_j N_j = O(N)$. A scan measures each bin's original volume and pair count. Computing all N_j from these measurements costs $(\log N)^{O(1)} = O(N)$ further steps.

For each nonempty bin, the capacity of a piece is $p_j = \lceil N_j/n_j \rceil \geqslant bH_j$. We split the original encoding of each pair into consecutive p_j-bit pieces, padding the last piece as needed. Every piece carries its pair's comparison key and a header giving the number of valid bits in the piece. The number s_j of pieces satisfies

$$n_j \leqslant s_j \leqslant n_j + N_j/p_j \leqslant 2n_j.$$

The pieces in this bin have a common width of $O(p_j)$ bits, including the comparison key and the $O(\log p_j)$ header. Writing the comparison key and initializing the length counter cost $O(p_j)$ per piece, with the two-tape copies described in Appendix B.3. Since $n_j p_j = O(N_j)$, the expanded bin has volume $O(N_j)$ and is constructed in the same time.

For $s_j \geqslant 2$, Corollary 5.2 sorts the pieces by their comparison keys. A singleton stream needs only a scan. Stability preserves the input order of all pieces with a given key. In particular, the pieces of each pair remain consecutive and in their original order. Concatenating the valid prefixes recovers the pairs with their original delimiters. For fixed $0 < \varepsilon < 1/2$, the inequalities $s_j \leqslant 2n$ and $\sum_j N_j = O(N)$ bound the total sorting time by $O_\varepsilon\left(N \log n/(\log^{(3)} n)^{1/2-\varepsilon}\right)$. Processing the bins successively uses $O(N)$ space.

The linear-time merge of Appendix B.1 combines the sorted bins in $O\left(\log^{(2)} n\right)$ balanced rounds. Each round costs $O(N)$ time, including copying the associated data. The bin distribution and all merges fit the sorting bound since $(\log^{(2)} n)(\log^{(3)} n)^{1/2-\varepsilon} = o(\log n)$. Equal keys belong to the same bin, where their input order was preserved. This proves stability and the time bound for every fixed $0 < \varepsilon < 1/2$ on the same machine. The bound in terms of N follows from $n \leqslant N$ and monotonicity. □

For signed finite expansions in a fixed radix, we first remove leading zeros from the integral part and trailing zeros from the fractional part. All spellings of zero receive the same key. For a nonzero magnitude with u remaining integral digits, its binary code starts with 1^u0. The integral and fractional digits follow in order, each digit d encoded as $d + 1$ in a common fixed width. An all-zero block terminates the code. The initial run orders magnitudes by integral length, and for equal integral lengths the digit blocks order them numerically. If one digit string is a proper prefix of another, the longer fractional part contains a further nonzero digit, so the shorter magnitude is smaller. The magnitude codes are prefix-free, so complementing every bit reverses their order. Three equal-width tags, in increasing order, mark negative values, zero and positive values. The tag is followed by the complemented code for a negative value, by the ordinary code for a positive value, and by no bits for zero. A constant number of scans per numeral writes the key and carries the original pair as associated data, preserving its spelling in total length $O(N)$.

6 Sorting with $o(N \log N)$ Boolean Gates

For n integers of $\Theta(\log n)$ bits each, the comparison networks and integer-sorting circuits discussed in Section 1 give a bound of $O(n \log^2 n)$ gates. Our circuit uses $o(n \log^2 n)$ gates. We first solve the permutation-routing problem of Section 5. Let $k = \lceil \log n \rceil$ be the index width, denoted by h in Section 5.1. When $n = 2^k$, every k-bit destination occurs, so each prefix determines a bucket of known size and position. Computing distinct destinations from keys that may repeat then gives a sorter for items of width at least k.

For delimited streams, N fixes the circuit before the key and data boundaries are known. Appendix D groups the pieces into tables whose dimensions depend only on N, sorts them with Proposition 6.4, and reassembles the pairs in stable key order. Lemma D.1 gives the resulting bound. Encoding items of width less than k as delimited pairs also gives the remaining fixed-width case.

Throughout the construction we fix $0 < \varepsilon < 1/2$ and write $G(s) = (\log^{(4)} s)^{1/2-\varepsilon}$.

6.1 Routing the Pieces of an Item

In the proof of Lemma 3.3, a lane shifts to an intermediate row, changes color there according to (5), and shifts back by its new color. For circuits we can implement the color change by fixed wiring if each lane has one coordinate for every possible row permutation.

Lemma 6.1. *Let $q \geqslant 2$ be a power of two, let $X = [q]$ with arithmetic modulo q, and let $Y = X^{\mathrm{Sym}(X)}$ have cardinality $H = q^{q!}$. Suppose each row $x \in X$ contains one b-bit lane for each $y \in Y$, where $b \geqslant 1$. There exists a Boolean circuit that, given the rows and the table of a permutation π of X, sends each lane (x, y) to $(\pi(x), \lambda_{\pi,x}(y))$, preserving the bit order within the lane. For every x and π, the map $\lambda_{\pi,x}$ is a permutation of Y. The circuit has size*

$$O_\varepsilon\big(qHb(\log q)^{1/2+\varepsilon}\big) + (qH)^{O(1)}.$$

Proof. A lane index is a tuple $y = (y_\tau)_{\tau \in \mathrm{Sym}(X)}$ with entries in X. For the input permutation π, the color of lane y is y_π. In each intermediate row a the fixed wiring is the permutation g_a of Y given by

$$(g_a y)_\tau = a - \tau(a - y_\tau) \qquad (\tau \in \mathrm{Sym}(X)).$$

Each coordinate map is a permutation. Its π-coordinate satisfies $(g_a y)_\pi = \phi_a(y_\pi)$, so the two shifts and g_a meet the routing condition in the proof of Lemma 3.3. Every lane of x thus reaches row $\pi(x)$, and the three bijections permute its H lanes.

For each fixed lane the first and last operations are programmable cyclic shifts on q entries. Equation (3) gives their total size. To determine their offsets, the circuit compares the input table with all $q!$ constant permutation tables. For each y the comparison bits select y_π among the constants y_τ. This costs $(qH)^{O(1)}$ gates independently of b. □

6.2 Routing by Destination Digits

The capacities in Definition 4.1 depend on the digit counts of the current bucket. Within a bucket containing all destinations with a given prefix, every next digit has the same multiplicity, so we can fix the capacity at its mean and measure the excess by the deviations from that mean.

Lemma 6.2. *Let $t \geqslant 1$, $K = 2^t$, $q = K^3$, $R \geqslant q$ and $m = Rq$. Suppose m items form R blocks of q items and each digit $c \in [K]$ occurs m/K times. There exist a fixed wiring from the blocks to R rows of q items and a Boolean circuit of size $mq^{O(1)}$ that computes one cyclic shift per block from the digits. Let $X(r, c)$ count the occurrences of c in row r after applying the shifts and the wiring, in that order. Then*

$$\sum_{r,c} (X(r, c) - q/K)^2 \leqslant 9m.$$

Applying the shifts to items of width β' uses $O_\varepsilon\big(m\beta'(\log q)^{1/2+\varepsilon}\big)$ additional gates.

Proof. There is a q-regular bipartite multigraph on R vertices per side whose biadjacency matrix A has second singular value at most $2\sqrt{q-1}$ [MSS18, Theorem 1.1]. Cohen [Coh16] constructs such a graph in deterministic polynomial time. The entry A_{rg} counts edges from input block g to row r. Each block rotates its items onto its incident edges in a fixed order, and each row receives its items in a fixed edge order.

For independent uniform block shifts, let p_{gc} be the fraction of block g having digit c, and let $p_c = (p_{gc})_{g\in[R]}$. The mean row counts for c form the vector Ap_c, and the entries of p_c have mean $1/K$. With $\mathbf{1}$ denoting the all-ones vector in $\mathbb{R}^R$, this gives

$$\sum_c \|Ap_c - (q/K)\mathbf{1}\|_2^2 \leqslant 4q \sum_c \|p_c - \mathbf{1}/K\|_2^2 \leqslant 4q \sum_{g,c} p_{gc}^2 \leqslant 4m.$$

For fixed r, g, the counts contributed by block g to row r are nonnegative and sum to the edge multiplicity A_{rg}. The sum of their squared values is at most A_{rg}^2. Independence between distinct blocks and the singular-value bound give

$$\sum_{r,c} \mathrm{Var}(X(r,c)) \leqslant \sum_{r,g} A_{rg}^2 \leqslant q^2 + 4q(R-1) \leqslant 5m.$$

Adding the squared bias proves the claimed bound in expectation.

It suffices that the shifts of any two blocks meeting in one row are independent. A block shares a row with at most $q(q-1)$ other blocks. The circuit generator can therefore assign a label $z_g \in \mathbb{F}_{q^2}$ to each block greedily so that blocks meeting in a row have distinct labels. In a fixed binary basis of $\mathbb{F}_{q^2}$, the first $\log q$ coordinates of $uz_g + v$ encode the shift of block g for a uniform seed $(u, v) \in \mathbb{F}_{q^2}^2$. These shifts are uniform, and distinct labels give independent shifts by the bijectivity of two-point affine interpolation.

The circuit evaluates the squared deviation from the given digits for all q^4 seeds, with fixed wiring for each candidate. For one seed, local counts and squares cost $mq^{O(1)}$ gates. A balanced addition tree sums the RK values of $O(\log q)$ bits in $O(RK \log q)$ gates, since the width grows by one bit per level. Choosing a least score costs $O(q^4 \log m)$ further gates. All offsets of the selected seed are then available within the stated control size. Only these shifts act on the full items, at the cost given by (3). □

Proposition 6.3. *Let $n = 2^k$ with $k \geqslant 1$, let $\beta \geqslant k$ be the common item width, and let $V = n\beta$. There exists a Boolean circuit that, given n items and a permutation of $[n]$ listed in source order, sends each item to its destination with its bits in their original order. Its size is*

$$O_\varepsilon\left(\frac{Vk}{(\log^{(4)} n)^{1/2-\varepsilon}}\right).$$

Proof. The k-bit destinations occupy $nk \leqslant V$ bits in addition to the items. For sufficiently large n let

$$t = \left\lfloor \tfrac{1}{32} \log^{(4)} n \right\rfloor, \qquad K = 2^t, \qquad q = K^3, \qquad H = q^{q!}.$$

For bounded t a comparison network suffices. Since $q \log q = o(\log^{(2)} k)$, we have $\log H = q! \log q = (\log k)^{o(1)}$ and $(qH)^{O(1)} = k^{o(1)}$. We append the destination to the item and use the lane labels of Section 3.2. Padding to H equal pieces and labeling each by its original lane number adds at most $H(1 + \lceil \log H \rceil)$ bits, so the encoded items still have width $O(\beta)$. For each bit of the current digit, its original lane and its offset within that lane are fixed. Testing the labels and selecting the bit at that offset costs $O(tH \log H) = O(k)$ gates for the entire digit, in any lane order.

At every stage the items in a bucket have a common prefix of their destinations. All its suffixes occur once, so for a bucket of size $m \geqslant q^2$ the next t bits divide it into K children of size m/K. Their boundaries are fixed. By Lemma 6.2 we can distribute the items into $R = m/q$ rows. As in Section 4.2, in each row the first $\min(X(r,c), q/K)$ items of digit c occupy successive slots reserved for c. Matching the excess items with the remaining slots in order completes the local permutation. An item is misplaced if its slot is reserved for a different digit. The permutation table, the flags for these items and the permuted digits cost $q^{O(1)}$ gates per row. Lemma 6.1 applies the permutation to the full items. A fixed wiring gathers all slots reserved for a digit consecutively. The number E of misplaced items satisfies

$$E = \tfrac{1}{2} \sum_{r,c} |X(r,c) - q/K| \leqslant \tfrac{1}{2}\sqrt{RK \cdot 9m} = \frac{3m}{2K}.$$

Let $s = 2m/K$. We compute E from the flags and use the superconcentrator of (2) to pack the misplaced items into the first E positions of an s-slot buffer. The same paths also carry the digit reserved at each vacated slot. For each digit c, the number of buffered items of digit c equals the number of vacated slots reserved for c, since both original totals were m/K and all items retained in their slots have the reserved digit. A separate s-slot superconcentrator for c routes these marked sources to these marked targets. Masking the other outputs and forming their bitwise OR over c yields one item at each active target. The reverse packing paths return the items to their vacated slots. Since $\log s \leqslant \log m \leqslant k \leqslant \beta$, the two full-size passages use $O(m\beta)$ gates. The K smaller superconcentrators also use $O(m\beta)$ gates in total, since $Ks = 2m$. Thus every child receives the intended items, with the current order of each item's lanes preserved during the correction.

The expander edges and the fixed permutation gathering the slots cost only wires in the circuit. The tape layout in Sections 4.1 and 4.2 instead groups items by interchanging address fields, whose movement cost enters the time bound.

The same operation recurs in the fixed child intervals. When $m < q^2$, fewer than $6t$ destination bits remain. A comparison network sorts this last suffix in $O(m\beta t)$ gates after its bits are read from the labeled lanes. Finally each item sorts its H lanes by their distinct lane numbers. Removing these numbers, the padding and the destinations gives the original items at their indicated positions. This last sorting costs $O(V \log H)$ gates.

Each full digit level has size $O_\varepsilon(Vt^{1/2+\varepsilon})$, including the header circuits because $\beta \geqslant k$ and $(qH)^{O(1)} = k^{o(1)}$. There are at most k/t levels. The terminal buckets and the final restoration together cost $O(V \log H) = o(Vk/t^{1/2-\varepsilon})$. The claimed bound follows from $t = \Theta(\log^{(4)} n)$. □

6.3 Computing Stable Destinations

The routing circuit of Proposition 6.3 accepts distinct destinations in source order. To determine them from keys with repetitions, we compute the *stable position* of each item, its zero-based position in sorted order when equal keys are ordered by input position. This position distinguishes different occurrences of the same key, even though they have the same rank as defined in Section 2.2.

Proposition 6.4. *Let $n \geqslant 2$, let $k = \lceil \log n \rceil$, and let $\beta \geqslant k$ be the common item width. Let $V = n\beta$. For each contiguous key field of ℓ bits, where $0 \leqslant \ell \leqslant \beta$, there exists a Boolean circuit that stably sorts every array of n items by that field. The family can be constructed in polynomial time and, for every fixed $0 < \varepsilon < 1/2$, has size*

$$O_\varepsilon\left(\frac{V \log n}{(\log^{(4)} n)^{1/2-\varepsilon}}\right).$$

Proof. With Proposition 6.3 available, it suffices first to compute stable positions using only the keys. Suppose $n = 2^k$ and the keys have k bits. Let $T(k)$ denote the size of our circuit returning their stable positions in source order. We use four stable digit passes from least to most significant, with $d = \lceil k/4 \rceil$ bits per digit and leading zero padding. Let $b = 2^d$. For all sufficiently large k, we have $b \mid n$ and $d + 1 < k$. The items in each pass carry their original indices as well as their full keys, so their width is $O(k)$.

In the current order the items form n/b consecutive blocks of size b. Each block receives one marker for every $c \in [b]$. The marker for c has auxiliary key $2c$, and each item with digit c has auxiliary key $2c + 1$. The real items appear in their current relative order. A recursive call on these $2b = 2^{d+1}$ auxiliary keys of $d + 1$ bits gives a local destination for every marker and every real item. In the resulting order, the marker for c precedes the entire run of its real items.

Let p_{gc} be the local destination of marker c in block g and let $p_{g,b} = 2b$. In this block $p_{gc} - c$ real items have digit smaller than c. Hence $h_{gc} = p_{g,c+1} - p_{gc} - 1$ is the frequency of digit c. The first global position of run (g, c) in a stable digit sort is

$$G_{gc} = \sum_{c'<c} \sum_{g'} h_{g'c'} + \sum_{g'<g} h_{g'c}.$$

The recursive outputs give all marker positions on fixed wires. A prefix scan of the n frequencies, with digit first and block second, computes all G_{gc} using $O(nk)$ gates.

A first application of Proposition 6.3 on $2n$ slots realizes all local permutations together, using block g and the local destination as the two fields of each destination. Marker (g, c) carries the offset $G_{gc} - p_{gc} - 1$. A scan in local destination order propagates this value to the real items following the marker. An item now in local slot j receives global destination $G_{gc} + j - p_{gc} - 1$. Within each run these are consecutive positions in stable order, and the frequencies partition $[n]$ into the corresponding intervals. Giving marker (g, c) destination $n + gb + c$ completes a permutation of $[2n]$. A second application of Proposition 6.3 leaves the real items in stable digit order in the first n slots. All temporary fields have $O(k)$ bits.

After the four passes, routing the sorted slot numbers back by the original indices gives the stable positions in source order. For bounded k a comparison network with index tie breaking suffices. We obtain

$$T(k) \leqslant 4\,2^{k-d} T(d + 1) + O_\varepsilon\left(\frac{2^k k^2}{G(2^k)}\right).$$

After division by $2^k k^2 / G(2^k)$, the coefficient of the recursive term is

$$8\left(\frac{d+1}{k}\right)^2 \frac{G(2^k)}{G(2^{d+1})} \longrightarrow \frac{1}{2}.$$

Induction gives $T(k) = O_\varepsilon(2^k k^2 / G(2^k))$. One final routing of the original β-bit items by these positions sorts them stably. A constant number of stable passes on fields of at most k bits handles $O(k)$-bit keys. For general n, padding at the end with maximal-key dummies to the next power of two places all dummies last by stability.

For a key of $\ell > k$ bits we apply the naming construction in the proof of Corollary 5.1 to $L = \lceil \ell/k \rceil$ successive chunks. Its column rearrangements are fixed wiring. Each column sort has n keys of at most $2k$ bits. Adjacent comparisons and a prefix scan assign the names, and routing by the attached positions returns them to their input order. The proof of Corollary 5.1 shows that the final names preserve order and ties. There are $O(L)$ columns over all rounds, each costing $O_\varepsilon(nk^2 / G(2^k))$ gates. Since $Lk \leqslant \ell + k \leqslant 2\beta$, their cost and the final stable sort of the β-bit items

fit the stated bound. An empty key field leaves every item in input order. The construction of the circuit descriptions is given below. □

For items of width $\beta \geqslant k$ with $O(k)$-bit keys, larger item widths allow us to route by larger digits. Let $B = \min\{n, \beta\}$. In the proof of Proposition 6.3, let $t = \lfloor (\log^{(3)} B)/32 \rfloor$ for sufficiently large B. Now $q = 2^{3t}$ satisfies $q \log q = o(\log^{(2)} B)$, so $\log H = q! \log q = (\log B)^{o(1)}$ and $(qH)^{O(1)} = B^{o(1)}$. Thus the controls and piece labels fit the allowance $O(\beta)$ per item. Also $B \leqslant n$ implies $\log H = k^{o(1)}$, so restoring piece order at the end still fits the gate bound for routing. The stable-position circuit on k-bit keys keeps its original size. Since $k \leqslant B \leqslant \beta$, $\log^{(3)} k = \log^{(4)} n + O(1)$ and $x/(\log^{(3)} x)^{1/2-\varepsilon}$ is eventually increasing, its size is $O_\varepsilon(Vk/(\log^{(3)} B)^{1/2-\varepsilon})$. This is also the total size. In particular, for $\beta \geqslant n^\delta$ with fixed $\delta > 0$, the denominator becomes $(\log^{(3)} n)^{1/2-\varepsilon}$.

Constructing the circuits. The graph construction in Lemma 6.2 runs in polynomial time. The generator assigns labels to the input blocks greedily, giving distinct labels to blocks that meet in a row. The small finite fields and seeds can be enumerated in polynomial time. The sorting networks and self-routing superconcentrators have logspace-uniform constructions [HR24, Lemma 8 and Corollary 19].

At the widths used in (3), the generator can find a smallest circuit for shifting a single bit plane. For width q it enumerates Boolean circuits in increasing size, up to the $O(q \log q)$ gates of a barrel shifter with one conditional shift by a power of two for each offset bit, and tests every candidate on its $q2^q$ possible inputs. Encoding the gates and output wires gives total search time $2^{O(q(\log q)^2)}$. Even at the larger choice of t above, $q \leqslant (\log^{(2)} n)^{3/32}$, so the search time is $n^{o(1)}$. The first correct circuit satisfies the size bound of (3) for every fixed ε, and parallel copies implement the shifts on all bit planes. All lane indices, local permutations and wirings can be enumerated within the polynomial bounds on their sizes. Integer bit lengths determine t to within an additive constant, which suffices in the size estimates. Thus a single polynomial-time generator gives the circuit family for every fixed ε.

For $O(k)$-bit keys the circuits may also be arranged in depth $O(k^3)$. The seed tests and local settings are computed in parallel. Within a row, balanced comparisons and selection among the permutation tables have depth polynomial in q and $\log H$, hence $k^{o(1)}$ for both choices of t. Summing the scores occupies $O(k)$ levels with $O(k)$-bit additions. The winning-seed comparisons and the superconcentrator routes of [HR24, Corollary 19] also fit depth $O(k^2)$ per digit. A full permutation has depth $O(k^3)$. Let $\Delta(k)$ be the depth of the circuit computing stable positions for 2^k keys of k bits. The recursive calls within each digit pass run in parallel, so $\Delta(k) \leqslant 4\Delta(\lceil k/4 \rceil + 1) + O(k^3) = O(k^3)$.

6.4 Proof of Theorem 1.3

With the fixed-width sorter of Proposition 6.4 and the delimited-stream bound of Lemma D.1 in hand, we can now prove Theorem 1.3.

Proof. The proposition gives the first bound for $\beta \geqslant \lceil \log n \rceil$, and the lemma gives the second. It remains to cover $1 \leqslant \beta < \lceil \log n \rceil$. For the binary instance of the lemma, fixed wiring writes each key and its entire original item, each followed by a delimiter. A tag bit distinguishes delimiters from data bits. All encoded items have equal length, so fixed output wires recover the sorted items. The encoding has length $\Theta(V)$, and $\log V = \Theta(\log n)$ and $\log^{(4)} V = \Theta(\log^{(4)} n)$. The second bound thus gives the first throughout the remaining range. The generators of the two constructions and the fixed encoding give the stated uniformity. □

7 Open Problems

In sorting, the machine determines the output order from the keys. In routing, each item comes with its destination. We ask about both tasks on a deterministic machine with a fixed number of tapes and working space linear in the input length.

Question 7.1 (Sorting logarithmic-width items)**.** *Can we stably sort* n *items of* $\Theta(\log n)$ *bits each in* $O\left(n(\log n)^{2-\delta}\right)$ *tape steps for some fixed* $0 < \delta < 1$*?*

By Proposition F.2, the same question can be asked for routing logarithmic-width items to distinct destinations given in source order. At this width, tape mergesort uses $O\left(n(\log n)^2\right)$ steps. Theorem 1.2 saves a factor of $(\log^{(3)} n)^{1/2-o(1)}$ over this bound. By Corollary F.3, a positive answer would extend to the arbitrary-length keys and associated data of Theorem 1.1, with any fixed saving $0 < \gamma \leqslant \delta$ such that $\gamma < 1/2$.

In the next question Q is a power of two. The destinations form a permutation of $[Q]$ listed in source order, and the time bound includes computing the moves.

Question 7.2 (Routing to given destinations)**.** *Each of* Q *strings of* B *bits is given with its position in the output. Are there constants* $\alpha > 0$ *and* $0 < \delta < 1$ *such that we can write the strings in that order in* $O\left((\log Q)^{1-\delta}\right)$ *tape steps per bit whenever* $B \geqslant Q^{\alpha}$*?*

For $B \geqslant 2^{Q^{\alpha}}$ with fixed $\alpha > 0$, Theorem A.1 gives $(\log Q)^{1/2+o(1)}$ steps per bit.

An affirmative answer to Question 7.1 would imply one to Question 7.2 for every $B \geqslant \lceil \log Q \rceil$ by Corollary F.3. In the other direction, a positive answer to Question 7.2 would improve the bound for logarithmic-width items by replacing the power of $\log^{(3)} n$ in its denominator with a positive power of $\log^{(2)} n$, an intermediate step toward Question 7.1. A radix level processing t key bits routes rows of $8 \cdot 2^{2t}$ items. With a polynomial width threshold we could set $t = \lfloor c \log^{(2)} n \rfloor$ for a sufficiently small constant $c > 0$. At this choice of t, the items meet the width threshold and we could compute the shifts and row permutations in $O(\log n)$ steps per item. For any fixed $0 < \varepsilon < 1/2$, digit partitioning would use $O_{\varepsilon}\left(t^{1-\delta} + t^{1/2+\varepsilon}\right)$ steps per bit, including the address-field interchanges. Summing over $O(\log n/t)$ levels gives a saving of $(\log^{(2)} n)^{\min\{\delta, 1/2-\varepsilon\}}$, with terminal mergesorts charged once per item as in Section 4.3.

AI Use Disclosure

The author used OpenAI's ChatGPT (GPT-6) to develop the tape and circuit constructions, including balanced-word routing and deterministic block shifts, and to refine the proofs and exposition.

Bibliography

[AHNR98] Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman. Sorting in linear time? *Journal of Computer and System Sciences*, 57(1):74–93, 1998. doi:10.1006/jcss.1998.1580. p. 3.

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983. doi:10.1007/BF02579338. pp. 4 and 9.

[ALS22] Gilad Asharov, Wei-Kai Lin, and Elaine Shi. Sorting short keys in circuits of size $o(n \log n)$. *SIAM Journal on Computing*, 51(3):424–466, 2022. doi:10.1137/20M1380983. Full version available as arXiv:2010.09884. pp. 4a and 4b.

[AV88] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535. p. 6.

[BABP03] Amir M. Ben-Amram, Omer Berkman, and Holger Petersen. Element distinctness on one-tape Turing machines: A complete solution. *Acta Informatica*, 40(2):81–94, 2003. doi:10.1007/s00236-003-0125-8. p. 7.

[Ben64] V. E. Beneš. Optimal rearrangeable multistage connecting networks. *The Bell System Technical Journal*, 43(4):1641–1656, 1964. doi:10.1002/j.1538-7305.1964.tb04103.x. p. 29.

[BM91] Robert S. Boyer and J Strother Moore. MJRTY—a fast majority vote algorithm. In Robert S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 105–117. Kluwer Academic Publishers, 1991. doi:10.1007/978-94-011-3488-0_5. p. 31.

[BP25] Andrew Best and Yuval Peres. Edit distance in substitution systems. arXiv:2501.00440v3, 2025. p. 11.

[Cha07] A. V. Chashkin. On the complexity of a cyclic shift of a set of real numbers. *Journal of Applied and Industrial Mathematics*, 1(2):175–177, 2007. doi:10.1134/S199047890702007X. p. 39.

[CLRS22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, fourth edition, 2022. ISBN 978-0-262-04630-5. pp. 3a and 3b.

[Coh16] Michael B. Cohen. Ramanujan graphs in polynomial time. In *57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 276–281, 2016. doi:10.1109/FOCS.2016.37. p. 21.

[DM80] David Dobkin and J. Ian Munro. Determining the mode. *Theoretical Computer Science*, 12(3):255–263, 1980. doi:10.1016/0304-3975(80)90057-2. p. 7.

[DMS91] Martin Dietzfelbinger, Wolfgang Maass, and Georg Schnitger. The complexity of matrix transposition on one-tape off-line Turing machines. *Theoretical Computer Science*, 82(1):113–129, 1991. doi:10.1016/0304-3975(91)90175-2. Author's copy. p. 4.

[FHLS23] Alireza Farhadi, MohammadTaghi Hajiaghayi, Kasper Green Larsen, and Elaine Shi. Lower bounds for external memory integer sorting via network coding. *SIAM Journal on Computing*, 52(2):STOC19–87–STOC19–111, 2023. doi:10.1137/20M1321887. p. 6.

[FJ59] Lester R. Ford, Jr. and Selmer M. Johnson. A tournament problem. *The American Mathematical Monthly*, 66(5):387–389, 1959. doi:10.1080/00029890.1959.11989306. p. 3.

[FS74] Robert W. Floyd and Alan J. Smith. A linear time two tape merge. *Information Processing Letters*, 2(5):123–125, 1974. doi:10.1016/0020-0190(74)90046-5. pp. 4 and 32.

[FW93] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993. doi:10.1016/0022-0000(93)90040-4. p. 3.

[Han04] Yijie Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *Journal of Algorithms*, 50(1):96–105, 2004. doi:10.1016/j.jalgor.2003.09.001. p. 3.

[HPV75] John E. Hopcroft, Wolfgang J. Paul, and Leslie G. Valiant. On time versus space and related problems. In *16th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 57–64. IEEE Computer Society, 1975. doi:10.1109/SFCS.1975.23. Author's copy. pp. 4, 5, and 8.

[HR24] Justin Holmgren and Ron Rothblum. Linear-size Boolean circuits for multiselection. In *39th Computational Complexity Conference (CCC)*, volume 300 of *Leibniz International Proceedings in Informatics*, pages 11:1–11:20, 2024. doi:10.4230/LIPIcs.CCC.2024.11. pp. 9, 24a, and 24b.

[HT02] Yijie Han and Mikkel Thorup. Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 135–144. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181890. p. 3.

[HvdH25] David Harvey and Joris van der Hoeven. Integer multiplication is at least as hard as matrix transposition. In *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1366–1381. IEEE, 2025. doi:10.1109/FOCS63196.2025.00072. Full version available as arXiv:2503.22848v1. p. 4.

[Hwa71] F. K. Hwang. On Beneš rearrangeable networks. *The Bell System Technical Journal*, 50(1):201–207, 1971. doi:10.1002/j.1538-7305.1971.tb02542.x. p. 29.

[KK21] Michal Koucký and Karel Král. Sorting short integers. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics*, pages 88:1–88:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.88. pp. 4 and 9.

[KMR72] Richard M. Karp, Raymond E. Miller, and Arnold L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, pages 125–136, 1972. doi:10.1145/800152.804905. p. 17.

[KMR26] Michal Koucký, Ian Mertz, and Ninad Rajgopal. Efficient error amplification for sorting and orthogonal vectors, 2026. Author manuscript. pp. 5 and 6.

[KSB06] Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858. p. 36.

[Lei85] Tom Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34(4):344–354, 1985. doi:10.1109/TC.1985.5009385. p. 6.

[LLL26] Baochun Li, Zongpeng Li, and Sirui Liu. Undirected multiple-unicast network coding and faster external-memory integer sorting. arXiv:2610.20079v4, 2026. pp. 5, 6a, 6b, 6c, 6d, and 13.

[LS22] Wei-Kai Lin and Elaine Shi. Optimal sorting circuits for short keys. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3583–3629. SIAM, 2022. doi:10.1137/1.9781611977073.142. p. 4.

[MSS18] Adam W. Marcus, Daniel A. Spielman, and Nikhil Srivastava. Interlacing families IV:

Bipartite Ramanujan graphs of all sizes. *SIAM Journal on Computing*, 47(6):2488–2509, 2018. doi:10.1137/16M106176X. p. 21.

[Ope26a] OpenAI. Integer multiplication below $n \log n$, September 2026. Preprint, September 23 manuscript. p. 4.

[Ope26b] OpenAI. The logarithmic exponent of finite Fourier transforms, November 2026. Preprint, November 24 manuscript. p. 10.

[Ope26c] OpenAI. The logarithmic exponent of integer multiplication on tapes, November 2026. Preprint, November 24 manuscript. pp. 4a, 4b, 9, 33a, and 33b.

[Pau79] Wolfgang J. Paul. Kolmogorov complexity and lower bounds. In Lothar Budach, editor, *Fundamentals of Computation Theory (FCT 1979)*, pages 325–334. Akademie-Verlag, 1979. Author's copy. pp. 4a and 4b.

[Pip96] Nicholas Pippenger. Self-routing superconcentrators. *Journal of Computer and System Sciences*, 52(1):53–60, 1996. doi:10.1006/jcss.1996.0005. p. 9.

[PK85] Martti Penttonen and Jyrki Katajainen. Notes on the complexity of sorting in abstract machines. *BIT*, 25(4):611–622, 1985. doi:10.1007/BF01936140. p. 4.

[PV76] Nicholas Pippenger and Leslie G. Valiant. Shifting graphs and their applications. *Journal of the ACM*, 23(3):423–432, 1976. doi:10.1145/321958.321962. p. 39.

[Sch78] Claus-Peter Schnorr. Satisfiability is quasilinear complete in NQL. *Journal of the ACM*, 25(1):136–145, 1978. doi:10.1145/322047.322060. pp. 4 and 16.

[Ser20] Igor S. Sergeev. On the monotone complexity of the shift operator. arXiv:1905.10747v2, 2020. p. 39.

[Sto73] Hanns-Jörg Stoß. Rangierkomplexität von Permutationen. *Acta Informatica*, 2(1):80–96, 1973. doi:10.1007/BF00571465. p. 8.

[SW23] András Z. Salamon and Michael Wehar. Effective guessing has unlikely consequences. *Theory of Computing Systems*, 67(3):548–568, 2023. doi:10.1007/s00224-023-10119-x. p. 4.

[vL76] Jan van Leeuwen. The complexity of data organization. In K. R. Apt and J. W. de Bakker, editors, *Foundations of Computer Science II, Part 1*, number 81 in Mathematical Centre Tracts, pages 37–147. Mathematisch Centrum, Amsterdam, 1976. Full text at CWI. p. 32.

A Wide Binary Strings

In Lemma 3.3, the pieces of a string can reach their destination row in a different order. We now restore their order by routing smaller groups of pieces recursively.

Theorem A.1. *Let $\alpha > 0$ be fixed, let Q be a power of two, and let π be a permutation of $[Q]$. Suppose Q strings of B bits each are given with their destinations $\pi(0), \dots, \pi(Q-1)$ in source order. There exists a uniform multitape Turing machine, depending only on α, that writes string i in position $\pi(i)$ for every $i \in [Q]$. If $B \geqslant 2^{Q^\alpha}$, the machine uses*

$$O_\alpha\left(QB(\log Q)^{1/2+o(1)}\right) \quad \textit{time and} \quad O(QB) \quad \textit{space,}$$

including the computation of the routing instructions. For all $B \geqslant Q^2$, the same machine uses $O(QB)$ space and $O_\alpha\left(QB(\log Q)^{1/2+o(1)}\right) + C(Q)$ time for a computable function C of Q alone.

The balanced word of Lemma 3.2 and the routing procedure of Lemma 3.3 implement the local permutations in the proof of Theorem A.1. We then recover the original order of the pieces.

Proof. Let γ be a fixed rational in $(0, \min\{\alpha, 1\})$. The recursive algorithm applies when $\lfloor \log B \rfloor \geqslant \lceil Q^\gamma \rceil$. The machine tests this condition by raising $\lfloor \log B \rfloor$ to the denominator of γ and comparing the result with Q raised to its numerator. It holds for all sufficiently large Q when $B \geqslant 2^{Q^\alpha}$. For bounded Q, tape mergesort by destination costs $O(QB)$ steps.

For a fixed integer $d > \max\{16, 4/\gamma\}$, we use $h = \log Q$, $s = \lfloor h/d \rfloor$, and $q = 2^s$. We may assume $s \geqslant 2$ by increasing the range of the preceding base case. The address of a string splits into $\lceil h/s \rceil$ fields of at most s bits each. A mixed-radix Beneš network [Ben64] realizes the permutation in $H = 2\lceil h/s \rceil - 1$ layers. In each layer, fixing all but one address field gives a group of strings, and a switch permutes the remaining field within that group. In Hwang's recursive construction for arbitrary factor sizes [Hwa71, Theorem 1], the outer switches are determined by a decomposition of a regular bipartite multigraph into perfect matchings. Augmenting paths and sequential table scans compute the switch tables in time polynomial in Q for fixed d.

Let W be the word of length M on q letters from Lemma 3.2 with tolerance $1/s$. In block $p \in [M/q]$ of q successive positions, each letter occurs once. Thus at offset $e_0 \in [q]$ the position $y = (p, e_0)$ has color $W(y) = \sigma_p(e_0)$ for a permutation σ_p of $[q]$.

With $b = \lceil B/(q^3 M) \rceil$, padding at the end brings each string to $\widehat{B} = q^3 M b$ bits. In order from outermost to innermost, the address fields of a padded string are

$$(\nu_f, \nu_0, p, e_0, \nu_1, z), \qquad \nu_f, \nu_0, \nu_1, e_0 \in [q], \quad p \in [M/q], \quad z \in [b].$$

In a layer whose switch acts on $e \leqslant s$ address bits, we interchange those bits with the low e bits of ν_f. In the former ν_f position we now have a field a_f whose low e bits give the switch address. The other string-address fields determine the switch. Its permutation on $[q]$ acts on the low e bits of a_f and fixes its prefix, independently of the displaced bits of ν_f. Interchanging a_f and ν_0 gives the field order $(\nu_0, a_f, y, \nu_1, z)$. For every ν_0, the routing procedure of Lemma 3.3 acts on (a_f, y, ν_1, z) with the same common-subsequence matching and the same order for the unmatched pieces. Reversing the interchanges restores ν_f, ν_0, ν_1, z and sends the string to its prescribed address. For fixed input string and ν_f, the resulting permutation of y and the matched positions are independent of ν_0, ν_1, z. For any fixed $0 < \varepsilon < 1/2$, Section 2.3 and Lemma 3.3 give a cost of $O_\varepsilon\left(Q\widehat{B}s^{1/2+\varepsilon}\right)$ steps per layer apart from computing the instructions.

We next restore the order of y within each output string for each value of ν_f. In a switch from local address x to x', a piece of color c leaves with color $c+x-x'$ modulo q by (5). Hence all pieces of the string with this ν_f undergo the same total color shift. At each layer, among pieces of one color, Lemma 3.3 preserves order within the matched pieces and within the unmatched pieces. Pieces with the same initial color and the same H matched/unmatched bits therefore retain their relative order through all layers.

For each ν_0, p, the map $e_0 \mapsto c = \sigma_p(e_0)$ permutes the q consecutive offsets. Each offset holds qb bits indexed by (ν_1, z), so this is a recursive call on q strings of qb bits each. The interchange of ν_0 and c across p in (1) gives the order (c, p, ν_0, ν_1, z). Within each current color c the blocks p now occur in increasing order. All bits indexed by (ν_0, ν_1, z) at a position (c, p) form one contiguous string, and all underwent the same changes of y.

Within each color, we group the strings stably by their H matched/unmatched bits. The initial color is determined by the current color and the known total shift. Each run is therefore in its initial order, with strictly increasing initial block indices, since each block originally had just one position of that color. Two-stream merges combine the at most 2^H runs in order of initial block index. We undo the color shift and reverse the interchange of c and ν_0. For each ν_0, p, a recursive call applies $c \mapsto \sigma_p^{-1}(c)$ to the q strings indexed by c. The destination string now has its original bit order. Tracing one representative for each piece position gives its grouping bits and initial block index. Simulating the merges on these indices gives the stream choices. Computing the instructions costs time polynomial in Q, q, M for fixed d, and executing them costs $O\left(HQ\widehat{B}\right)$ steps.

By Lemma 3.2, $\log M = O\left(2^{2s}s^2\right)$. Since $d > 4/\gamma$, every fixed polynomial in Q, q, M is $2^{o(Q^\gamma)}$. Even after multiplication by any fixed polynomial in $\log B$, it is $O(B)$ under the width condition. In particular this pays for computing the instructions, and $\widehat{B} \leqslant 2B$ for sufficiently large Q. The row width in the call to Lemma 3.3 is $qMb = \widehat{B}/q^2$. It therefore exceeds every fixed power of qM for sufficiently large Q, so Lemma 3.3 applies. Also

$$\log(qb) \geqslant \log B - \log(q^2M) \geqslant Q^\gamma - o(Q^\gamma).$$

Since $q \leqslant Q^{1/d}$, we have $\lfloor \log(qb) \rfloor \geqslant \lceil q^\gamma \rceil$ for all sufficiently large Q. Each recursive call on q strings thus meets the same width condition.

Each call applying σ_p processes q^2b bits. There are $Qq^2(M/q)$ such calls, one for each input string, ν_f, ν_0 and p. Their total volume is $Q\widehat{B}$, as is the total for the inverse calls. For a parent of volume $A = QB$, each phase therefore has volume at most $2A$ and each child has volume at most $2A/(QqM) \leqslant A/8$ above the base case. Under these bounds the two-tape implementation in Section B.3 arranges the calls in $O(A)$ steps per phase and uses $O(A)$ space including the recursive calls.

Let $F(h)$ bound the time per input bit at $h = \log Q$ under the tested width condition. A child has $\log q = \lfloor h/d \rfloor$, and its time is charged only to its own input volume. The two phases together process at most $4A$ bits. Including the $H = O(d)$ switch layers and the removal of padding gives

$$F(h) \leqslant 4F(\lfloor h/d \rfloor) + O_{d,\varepsilon}\left(h^{1/2+\varepsilon}\right).$$

Since $4/\sqrt{d} < 1$, induction gives $F(h) = O_{d,\varepsilon}\left(h^{1/2+\varepsilon}\right)$. The same machine and the same d work for every such ε. Below the tested width threshold, ordinary tape mergesort orders the strings by destination in $O\left(Q(B + \log Q)\log Q\right)$ time. For $B \geqslant Q^2$ this uses $O(QB)$ space. Here $\gamma < 1$ gives $B < 2^Q$ for $Q \geqslant 2$, so $C(Q) = O\left(Q2^Q \log Q\right)$ covers this case. □

Applying Theorem A.1 to the items in each padded row lets the radix sorter use wider digits as the item width grows.

Corollary A.2. *Let $X_1, \ldots, X_n$ be $n \geqslant 2$ items of common width $\beta = \Omega(\log n)$ whose keys occupy the same contiguous ℓ-bit field, where $0 \leqslant \ell \leqslant \beta$. Let $V = n\beta$. There exists a multitape Turing machine that stably sorts every such input using*

$$O\left(\frac{V \log n}{\min\left\{\log n, \log^{(2)} \beta\right\}^{1/2-o(1)}}\right) \quad \text{time and} \quad O(V) \quad \text{space.}$$

In particular, for every fixed $a > 0$ and $\beta \geqslant n^a$, both stable sorting and routing by destination keys use $O_a\left(V \log n/(\log^{(2)} n)^{1/2-o(1)}\right)$ time. If $\beta \geqslant 2^{n^\alpha}$ for fixed $\alpha > 0$, the running time per input bit is $O_\alpha\left((\log n)^{1/2+o(1)}\right)$.

Proof. Let $h = \lceil \log n \rceil$. For keys of $O(h)$ bits, we use the radix sorter of Section 4.3 with digit width $t = \lfloor \frac{1}{16} \min\left\{\log n, \log^{(2)} \beta\right\} \rfloor$ and radix $K = 2^t$. Bounded n is handled by mergesort. Each item carries its original position as a tie breaker. The digit labels determine the permutation of each padded row of $q = 8K^2$ items in Section 4.2. The row permutation is computable in $q^{O(1)}$ time from the compact digits, which is $O(\beta)$ for this choice of t. For sufficiently large n, $q \leqslant \log \beta$, so Theorem A.1 with $\alpha = 1$ routes the entire row in $O_\varepsilon\left(q\beta t^{1/2+\varepsilon}\right)$ steps for every fixed $0 < \varepsilon < 1/2$. The items arrive in bit order, and a scan reads the next digit. The block shifts cost $Kt^{O(1)} = O(\beta)$ per item. Thus the $O(\log n/t)$ digit levels cost $O_\varepsilon\left(V \log n/t^{1/2-\varepsilon}\right)$ steps.

In a terminal bucket of $m < 2^{4t}$ items, we sort the short keys, breaking ties by original input position and carrying the local positions. Numbering the sorted entries and sorting them back by local position gives the destinations in $O(mt \log n)$ steps on compact fields. Padding to a power of two uses at most $2m$ positions and at most $2^{4t} \leqslant \log \beta$. Routing by Theorem A.1 consequently costs $O_\varepsilon\left(m\beta t^{1/2+\varepsilon}\right)$ steps. Both costs fit $O_\varepsilon\left(m\beta \log n/t^{1/2-\varepsilon}\right)$, since $t \leqslant \log n$ and $t^{3/2} = O(\beta)$. Each item enters at most one terminal bucket, and all operations use linear space.

If $\ell \leqslant h$, the subroutine applies directly. For $\ell > h$ and $\beta \leqslant (\log n)^4$, Corollary 5.1 gives the stated bound since $\log^{(2)} \beta = \log^{(3)} n + O(1)$ in this range. It remains to consider $\ell > h$ and $\beta > (\log n)^4$. As in Corollary 5.1, we retain the items in input order and pad the key table to $n' = 2^h$ rows with copies of the first key. Here the number of columns J is the least power of two at least h^2. Each key is padded on the right to Jb bits, where $b = \lceil \beta/J \rceil$, and divided into J chunks of b bits. For large n, $J \leqslant n'$. Interchanging the high $\log J$ bits of the row address with the chunk index makes each column contiguous in the same physical row order as in Section 5.1. This costs $O_\varepsilon\left(V(\log J)^{1/2+\varepsilon}\right)$, within the stated bound.

As the machine reads the columns from most to least significant, each row carries the rank of its prefix among the distinct prefixes processed so far. Initially all rows have the same rank. The position of a row within a physical column serves as its identifier, since every column has the same row order. In each column the machine attaches the old rank and row identifier to each chunk, then groups the chunks by rank using the short-key algorithm. These items have width $\Theta(b)$ and keys of $O(h)$ bits. Here $b = \Omega\left(\sqrt{\beta}\right)$ and $b = O(\beta)$, so $\log^{(2)} b = \log^{(2)} \beta + O(1)$. Since the combined volume of the columns is $O(V)$, the total time for these groupings is within the stated bound.

To refine each equal-prefix class by its current chunk, we use the majority vote algorithm of Boyer and Moore [BM91, Section 5.2] and verify the candidate's frequency by a second scan. The cost of comparing chunks and rewinding the candidate is linear in the volume of the class. If the class splits and has a strict majority, ordinary tape mergesort orders the other chunks and one majority representative. The row identifiers saved for the majority receive the representative's

new rank. If the class has no strict majority, the mergesort includes all its chunks. An unsplit class remains one subgroup. Processing the old classes in rank order allows us to assign consecutive ranks to the distinct subgroups in order of chunk value. We retain the new rank and identifier for each row. Sorting these pairs by identifier aligns the ranks with the next column. In the last column the inverse rotation of each identifier's bits gives the original row position, so this sort returns the ranks in input order.

In a split class a row is an exception if its chunk is outside the strict majority or the class has no strict majority. Its new class contains at most half as many rows as its old class. Each row is therefore an exception at most h times over all J columns. At most one majority representative enters a mergesort per exception. Each comparison and move costs $O(b)$ and a mergesort costs at most $O(h)$ such operations per participating chunk. These mergesorts use $O(n'bh^2) = O(V)$ steps in total, since $J = \Theta(h^2)$. Sorting the ranks by position over all columns costs $O(n'Jh^2) = O(V)$ because $b = \Omega(h^2)$. The remaining scans use $O(V)$ time. With the ranks of the first n rows attached to the original items, the stable short-key algorithm gives the desired order. □

B Implementation on Two Writable Tapes

The bounds of Theorems 1.1 and 1.2 and Appendix A hold on two writable tapes in the model of Section 2. A tape symbol stores a fixed tuple, one component per *track*. All tracks on a tape share its head. A final scan writes the output contiguously on the designated tape.

B.1 Copying and Merging Streams

One head per stream suffices to copy symbols between corresponding marks in order. Given the masks, two such copies implement a stable binary split or its inverse. Aligned streams can also be combined pointwise. In a rectangular layout one head marks field boundaries from the innermost field outwards, counting previously marked boundaries in one sweep per dimension. The other head keeps a binary countdown. Over one period, the borrow chains, returns to the low bit and reset have total cost linear in the period length.

Let $A_1, \dots, A_r$ and $B_1, \dots, B_r$ be two lists of delimited strings of combined length D. They can be interleaved in the order $A_1, B_1, \dots, A_r, B_r$ in $O(D)$ time and space. We group each list by eights, allowing a shorter final group. A finite-state scan encodes the total length x of a group, including its delimiters, by $\lfloor x/8 \rfloor$ tally marks and a delimiter carrying $x \bmod 8$. The recursion interleaves the two lists of lengths. Even an empty string has a delimiter, so $2r \leqslant D$ and the recursive input has length at most

$$D/8 + 2\lceil r/8 \rceil \leqslant D/4 + 2.$$

Expanding these lengths marks the output cells for alternating left and right groups. Two marked copies bring corresponding groups together, and at most sixteen local scans per pair write its strings in alternating order. Both input lists are staged on one tape, with neighboring intervals for the recursive call. The work outside the call is $O(D)$, including rewinds. For $D \geqslant 8$ the recursive length is at most $D/2$. Direct copying for bounded D completes the proof.

Floyd and Smith [FS74] gave a linear-time two-tape merge for fixed-width items, described in detail by van Leeuwen [vL76, Sections 2.9–2.14]. For variable lengths, one sorted list lies on each tape. The heads compare the leading keys and rewind the inspected prefixes. Only the head at the winning item advances past it, with ties resolved in favor of the left list. The inspected prefixes have length at most that of the winning key plus one. Charging the comparison and rewinds to that item gives linear total cost even if the other key is compared several times. A separate track

marks the first winner and each change of source, including the transition to the remaining list. We prepend an empty left run if the first winner is on the right and append an empty right run if the last winner is on the left. Interleaving the runs gives a stable merge in time linear in the combined length, including associated data. This cost sums over the adjacent pairs in a balanced merge round.

B.2 Address-Field Interchange

Let $\mathcal{V} = \mathrm{PGB2}^{2u}$ be the volume of the layout (p, x, g, y, z) in Section 2.3. We implement the recursion of OpenAI [Ope26c, Theorem 2.1 and Section 2.5] on two tapes. For a fixed $1/2 < \alpha < 1$, a finite program (the *template*) specifies the recursive operations, a prime radix b and constants γ, m, a, W with $1/2 < \gamma < \alpha$, $b > W > 2$ and $2m^{\gamma} + 2a < m^{\alpha}$. At a node of volume v and field width e in radix b, the child width is $f = \lfloor (e-1)/(m+2a) \rfloor$. For $f = 0$, a bounded number of single-digit moves suffices. Otherwise the values of the first digit of the earlier field are grouped into batches of W, with padding in the last batch. Entries at the same position in their respective batches form one stream. Each of the W streams has volume at most $2v/W$.

Marked copies align the streams on tracks, so a scan evaluates a pointwise gate. When the recursion compares two f-digit fields, the second head holds the earlier value while the first scans the later field. Boundary marks determine when to change that value. For each fixed prefix, the three comparison outcomes form consecutive runs. If each value of the later field spans S cells, computing the run lengths costs $\mathrm{poly}(f + \log S) = O_{\alpha}(b^{f} S)$ steps. Countdowns mark the runs within the same bound.

In a lower triangular address change, each target field is a linear combination of its own and earlier source fields. The machine processes the targets from last to first. Each earlier-field summand gives a cyclic shift whose cut is computed once per fiber. Two marked copies rotate its blocks. Multiplication of a coordinate y by a fixed positive unit h modulo b^{f} is increasing on each source interval of constant $k = \lfloor hy/b^{f} \rfloor$. The targets of interval k have residue $-kb^{f} \bmod h$, so one marked copy suffices per piece. Exchanging the two masks implements division by h. Negation visits the nonzero blocks in reverse order and copies each forward. Single-digit moves have a bounded number of increasing pieces. Thus all work between recursive calls costs $O_{\alpha}(v)$.

At most Wm^{γ} calls use a single stream. Each uses a neighboring interval, from which its result is copied back. The other $2a$ calls reuse the recombined array's interval successively, shifting the array to leave room for a return descriptor containing the dimensions and a return instruction. Only the first kind of call parks an array, and its child volume is at most $2v/W$. Hence the parked volumes sum geometrically. Along a path the descriptors occupy $O_{\alpha}((\log e)(\log v)) = O_{\alpha}(v)$ cells. Including copies and head returns, the time per bit satisfies

$$T(e) \leqslant C_{\alpha} + (2m^{\gamma} + 2a)T(f).$$

Since $f \leqslant e/m$, induction gives $T(e) = O_{\alpha}(e^{\alpha})$ for $e \geqslant 1$, with $O_{\alpha}(v)$ visited space. For binary input, padding each coordinate range to the least power of b at least 2^{u} increases the volume by a factor less than b^{2}. An output scan removes the padding.

A single machine. For $\alpha_j = 1/2 + 1/(j+2)$, $j \geqslant 1$, the finite searches of OpenAI [Ope26c, Sections 2.4 and 6.1] enumerate the templates. For each template the number of scans, the polynomial cost of field arithmetic, the radix padding and the finite base cases all have computable bounds. Induction in the time recurrence and the geometric space sum therefore give computable time and space constants. A fixed two-tape interpreter encodes each simulated cell in a block carrying its

symbol and a copy of the transition table. It looks up and transfers the state and symbols within the current blocks, using an amount of work bounded in terms of the table alone. To expand the input or enter a fresh block, the head copies the table from its neighbor by shuttling between the two in time quadratic in the block length. Including initialization and decoding, we obtain computable nondecreasing integers $C_j \geqslant 2$ bounding the time and space of program j on volume v and binary field width w by $C_j v(1+w)^{\alpha_j}$ and $C_j v$.

At width u, the interpreter runs the enumeration for u steps and selects the last completed j with $C_j^{j+2} \leqslant 1+u$. If none qualifies, it uses individual digit moves. Capped arithmetic keeps selection within $\mathrm{poly}(u) = O(\mathcal{V})$ steps, and the selected j tends to infinity. With $d = \lceil \frac{1}{4} \log(1+u) \rceil$, the 4^d pairs of high d-bit patterns of x, y partition the array into equal parts. For each pair, countdowns mark its two coordinate intervals. An ordered copy of their intersection gives the compact stream on which program j interchanges the remaining $u - d$ bits. A second copy places the result at the output positions with the high patterns exchanged. The scans cost $O(\mathcal{V})$ per pair. Since $C_j \leqslant \sqrt{1+u} \leqslant 4^d$, the compact run visits at most $C_j \mathcal{V}/4^d \leqslant \mathcal{V}$ cells. Reusing this space for all pairs gives $O(\mathcal{V})$ space. The $4^d = O(\sqrt{u})$ scans per input bit fit the interchange bound since $\alpha_j > 1/2$. As $C_j \leqslant (1+u)^{1/(j+2)}$, the total time is

$$O(C_j \mathcal{V} u^{\alpha_j}) = \mathcal{V} u^{1/2+o(1)}.$$

B.3 The Sorting Passes

We process the parents of a radix level in tape order. On each tape, scratch space begins at the frontier marked on a protected track. For m items of width β, the heads work within $O(m\beta)$ cells of their respective marks. After each parent they clear the scratch tracks and advance the marks by the length consumed or written. Completed items retain their padded width.

To read a digit, one head scans the item's lanes while the other keeps the indices and offsets of the at most two pieces containing that digit. The position calculations fit the $O(\beta)$ bound of Sections 3.2 and 4.3. A scan collects the digits in a compact stream. Binary splits on a copy give the class counts.

Scratch tracks beside each block of R digits hold the cumulative histogram and the weights. In (9), two marked copies align the halves of a table for addition or align a rotated table with its partner. At each level, successive local doublings copy the weights to each residue. Products of aligned entries and pairwise sums use polynomial time and linear space in the field widths. Thus (10) gives $RKt^{O(1)}$ time per block and the tables occupy $O(RKt)$ cells. Once the chosen cut is marked on the digit stream, the updated histogram and weights move to the opposite tape before the old scratch tracks are cleared. The next block starts $O(Rt)$ cells farther along, so copying the tables forward costs $O(RKt)$. A scan transfers the cut marks to the full items, and two segment copies rotate each block.

After the first interchange, we compute each row's permutation and matching on its compact labels. The cost is polynomial in q, M and the digit width, hence $O(\beta)$ by Section 4.3. Countdowns expand the matching masks, allowing the matched lanes to be copied in order. The $O(t)$ binary splits and inverse splits act on the compact stream of unmatched lanes, whose volume is an $O(1/t)$ fraction of the total. Their output fills the vacant positions. The controlled shifts rotate fibers as in Section B.2. After the second interchange, binary splits sort the compact overflow. With each class marked, including the empty ones, the interleaving of Section B.1 places each main class immediately before its overflow. Binary splits also restore lane order. Terminal sorts use the stable merge of Section B.1.

The column sorts in Section 5.1 keep the original items on a protected track while each contiguous column is sorted in its local work area. A scan joins the final ranks to the items in input order. The short-width batches use the same sort on successive intervals. In the length bins of Section 5.2, both heads advance from one piece to the next. They copy the comparison key from the current header to a scratch track beside the next piece and then into its header. In bin j the key has width $bH_j \leqslant p_j$, so the copies and head returns cost $O(p_j)$ per piece.

Wide strings. For either recursive phase in the proof of Theorem A.1, let $A = QB$ be the parent volume. The children occupy successive intervals, each of volume at most $A/8$ and together of volume at most $2A$. Each batch holds as many consecutive inputs as fit in $A/4$. All but the last have volume greater than $A/8$, so the number of batches is bounded by a constant. The parent occupies CA cells per tape for a constant C large enough that its own buffers fit in half this interval. Within a batch, the heads copy the inputs to successive slots, each C times the length of its input. Each child runs in its slot, and a scan collects the outputs in order. Preparing the short instructions in traversal order fits the polynomial bound in Theorem A.1. Layout, collection and travel cost $O(A)$ per phase.

For the rank refinement in Corollary A.2, one tape holds the majority candidate and its head rewinds after each comparison. The destination and exceptional-chunk sorts use the stable merge of Section B.1.

C The Movement Bound for Bit Copies

Let the input consist of $n \geqslant 2$ distinct keys of common width w, and number their positions by $i \in [n]$. For each bit offset $j \in [w]$, the projection onto j retains only the labels (i, j) of Definition 2.1, writing each as i. On each tape the indices on either side of the head form two stacks, with the nearest index at the top. Registers hold the indices at the heads and in the finite control. With all heads initially at their left markers, every offset starts with the same sequence $0, \dots, n-1$.

One tape step induces $O(1)$ stack moves, local copies and erasures in total over all offsets, since it touches only a bounded number of registers and tape cells, including those entered by the heads. Each instruction names an operation and its locations from a fixed finite alphabet. With the output on a fresh tape, the sequence of instructions for any one offset determines its output order. Thus sequences of at most L instructions realize at most $2^{O(L)}$ permutations for $L \geqslant 1$.

Among the $n!$ permutations there is consequently one, π, whose shortest sequence has length $\Omega(\log(n!))$. Any fixed set of n distinct w-bit keys has an input order for which the sorted list of input positions is π. On this input all w offsets realize π, giving $\Omega(w \log(n!)) = \Omega(wn \log n)$ tape steps. An in-place output can be copied to a fresh tape in $O(T + nw)$ further steps, where T is the worst-case running time. This gives the same bound for large n. For bounded $n \geqslant 2$, exchanging two keys already costs $\Omega(w)$ steps under the copying rule.

D Circuits for Delimited Strings

As in Section 5.2, we split items into pieces and preserve the input order of pieces with equal keys. On tape, the machine measures each bin before choosing the piece width. A circuit has fixed wires for all streams of length N, so its arrays must be sized before the key boundaries are known. As in Corollary 5.2, we sort each batch and group its pieces into runs of equal keys. One address then suffices for the output of each run. Suffixes of a common text then order the short-key runs together with the long keys.

Lemma D.1. *Let $N \geqslant 1$. There exists a Boolean circuit C_N that stably sorts every valid delimited stream of length N by key, carrying the associated data. For every fixed $0 < \varepsilon < 1/2$, C_N uses*

$$O_\varepsilon\left(\frac{N \log N}{(\log^{(4)} N)^{1/2-\varepsilon}}\right)$$

gates. The family $(C_N)_{N \geqslant 1}$ can be constructed in polynomial time.

Proof. Here N counts symbols, including delimiters, and G is the function defined in Section 6 for the chosen ε. We may assume N is sufficiently large, since a fixed circuit suffices at each of the remaining lengths. Let h be the least power of two at least $\log(2N)$. Both an h-symbol word and a stream address occupy $O(h)$ bits. There is a fixed positive integer c such that at most $2^{c\ell}$ keys have length less than 2ℓ, for every integer $\ell \geqslant 1$. Let r be the largest power of two at most $h/(16c)$. Thus $r = \Theta(h)$ and $r \mid h$. Keys of length less than $2r$ are called short and the remaining keys are called long. If a common delimiter follows both fields, a prefix parity of the delimiter flags distinguishes the two roles.

We use indexed lookups on a table with s slots and $O(s)$ requests. Sorting the requests with the table entries by address, the entry first within each class, lets a scan transmit its value to all requests. A second sort restores request order by local return index. For placement at distinct occupied addresses, a marker for each address receives its value or a blank, and a second sort restores marker order. In both operations the sort keys have $O(\log s)$ bits. For entries of width $b \geqslant \lceil \log s \rceil$ the gate count is $O_\varepsilon(sb \log s / G(s))$ by Proposition 6.4. An ordinary comparison network gives $O(sb \log s)$.

Suffix ranks of a word text. For a text of prescribed length $\Theta(N/h)$ whose words have $O(h)$ bits, all suffix ranks in source order cost $O_\varepsilon(Nh/G(N))$ gates. We adapt the difference-cover recursion of Kärkkäinen, Sanders and Burkhardt [KSB06, Sections 3 and 7]. In a text of length ν, the sampled positions have residues 1 and 2 modulo three, with positions indexed from zero. Their initial triples, padded beyond the text with a sentinel smaller than its alphabet, receive positive names in lexicographic order by sorting and a scan. The recursive text concatenates the names in position order within each sampled class, with a separator following each class. The separator is smaller than every name and is a text symbol distinct from the padding sentinel of the recursive call. Up to its separator, each sampled suffix is represented by its consecutive triples. A difference within a final partial triple is handled by the padding, and a proper prefix ending at a triple boundary by the separator. Distinct suffixes of the original text have different lengths, so their entire lists of padded triples cannot coincide. The recursive ranks therefore order the sampled suffixes. Numbering them from zero in this order gives their sampled ranks. The recursive text has length at most $2\nu/3 + 4$.

For any two distinct residue classes there is a $j \in \{0, 1, 2\}$ such that adding j sends both classes to sampled classes. We compare the first j symbols, using the padding sentinel past the end, followed by a code for the suffix at offset j. A genuine sampled suffix of rank z has code $z + 1$, and an offset at or beyond the end has code zero. These tuples order the suffixes of the two classes together. Sorting the tuples, counting predecessors of the opposite class by a scan, and returning counts by source position give the cross-class counts. The sampled ranks order each sampled class, and the first symbol and the suffix code at offset one order class zero. Numbering each class from zero in its induced order gives the within-class ranks. Adding the two cross-class counts to each such rank gives the full zero-based suffix rank. This uses a constant number of sorts, scans and placements on $O(\nu)$ items of width $O(h)$. For $\nu \geqslant N^{1/3}$ the sort keys have $O(\log \nu)$ bits and

$G(\nu) = \Theta(G(N))$. The node lengths sum to $O(N/h)$ down to a constant base case. The smaller nodes together cost $O(N^{1/3}h^2)$ using comparison networks, which proves the claimed bound.

Batches of short keys. For each power of two $\ell \leqslant r$, one bin handles keys with lengths in $[\ell, 2\ell)$ symbols, with the empty key also in the bin $\ell = 1$. We divide each item, including its final delimiter, into packets of ℓ symbols, merging any shorter final packet into the preceding one. The last packet thus has length in $[\ell, 2\ell)$. An offset from the item start begins a new packet if it is divisible by ℓ and at least ℓ symbols remain. Distinct packet starts of this bin are separated by at least ℓ symbols, so each aligned ℓ-symbol cell needs at most one packet slot.

These slots can be filled in $O(N \log \ell)$ gates per bin. Reverse scans at symbol positions give distances to the next key and final delimiters, capped at 2ℓ. At an item start the former gives bin membership, which a scan propagates along the item. The latter gives the remaining raw length after including the final delimiter. Propagating each item's starting position modulo ℓ gives all relative offsets modulo ℓ. These operations use $O(\log \ell)$ bits per symbol and identify packet starts and lengths. Within each cell a masked selection finds its unique start, if any. Shifting a fixed window of 3ℓ symbols by this local offset retrieves the next 2ℓ symbols, which are masked at the packet length. At the first packet of an item, the same window contains the key and its delimiter. The ordered symbol codes and zero padding from Section 5.2 give its 2ℓ-symbol comparison key. A scan broadcasts the new key from each such cell to subsequent packet cells of that item. The fixed windows may cross cell boundaries.

Let $D_\ell = 2^{c\ell}$ and $M_\ell = D_\ell h^8$. In this bin the input is divided into batches of M_ℓ consecutive source symbols. Each packet belongs to the batch containing its start, even if its window crosses the batch boundary. Both ℓ and h divide M_ℓ. For large N, $M_\ell \leqslant N^{1/4}$ since $c\ell \leqslant h/16$ and $h < 2\log(2N)$. In each batch, sorting M_ℓ/ℓ slots by validity, key and source cell gives one run per distinct occupied key, with its packets in their original order. There are at most D_ℓ runs.

We concatenate the packets of a batch into a local array of $2M_\ell$ symbols. A prefix sum of lengths, zero on vacant slots, gives their starting positions. With the array viewed in ℓ-symbol words, each packet meets at most three words. A local shift produces its masked contributions and the corresponding word indices. A sort groups the contributions by word with one marker ordered last in each group. A segmented bitwise OR and restoration of marker order give the local array. From the changes in packet keys and the prefix sums of lengths, scans give the beginning and end of each run in this array. One descriptor per run, consisting of these endpoints and its key, is compacted into D_ℓ slots per batch. Up to this compaction all items and descriptors have width $O(\ell + \log h)$. Occupied descriptors then receive the absolute batch start and their slot number among the descriptor slots of all bins. In total these slots number

$$J = \sum_\ell \lceil N/M_\ell \rceil \, D_\ell = O(N \log h/h^8).$$

Here and below, sums over bin sizes range over powers of two.

Long keys. Each long item contributes one descriptor, the words of its key, and the words of its entire input item including both delimiters. Both sequences are cut into h-symbol words from the start of the item. The key uses the same symbol codes as the short keys, with a zero terminator and padding to a word boundary. A key of length divisible by h thus has a separate end word.

To produce these sequences, the source is divided into r-symbol cells. As above, a capped distance test at item starts marks long keys. Membership, field type and the item's starting position modulo h propagate along the input in $O(N \log h)$ gates. These fields identify the word starts of

the two sequences, including a separate end word where appropriate. Distinct long-item starts are separated by at least $2r$. Between such starts the word spacing is $h \geqslant r$, so each cell contains at most two marked positions per sequence. For each mark, a shift by its within-cell offset selects an h-symbol window of the source. The delimiters give the key or item boundary for masking the final word. The length of each raw word is retained.

Full addresses can be computed at the cell scale. Local scans on $O(\log r)$-bit offsets give the nearest preceding start and following final delimiter inside the cell, whenever present. Across cells, a prefix scan of last starts and a suffix scan of first ends give the neighboring addresses when a local one is absent. Combining these results at only a constant number of marks per cell gives the item's absolute start, its length at the start, and the relative offset of each word. A marked long-key delimiter also receives its owner and key length this way. Distinct such delimiters are separated by at least $2r$, since the later key itself has this length. The cells thus produce all sequences and descriptors in $O(N \log h)$ gates and $O(N/r)$ word slots, including vacant slots. The raw words of the long items, which serve as their output fragments, occur in source order.

Ordering the runs. Each long item gives one run. The runs of a short key are ordered by batch start, and those of a long key by item start. These tie breakers use $O(h)$ bits. For short keys the packets within a run are already in source order. When an item spans two or more batches, all its packets still precede those of any later item with that key. Short and long keys cannot be equal. Consequently, lexicographic order of the runs by key and tie breaker, followed by concatenation of their raw substrings, gives the stable output.

For each run we form a compound key consisting of its key words, terminated and padded as for the long keys, followed by a common fixed number of words for the tie breaker, most significant first. Short keys fit in one key word since $2r < h$. Unequal primary keys are distinguished within their terminated encodings, including when one is a proper prefix. Equal primary keys have identical padding and distinct tie breakers. The comparison of two compound keys is therefore settled within their own words, so the suffix ranks at their starts determine the run order.

Each word is tagged by run identifier and ordinal in its run. Short identifiers are the J descriptor slot numbers. A long item beginning at s has identifier $J + \lfloor s/r \rfloor$, which is unique because its cell contains at most one long-item start. Its key of length L contributes key words numbered zero through $\lfloor L/h \rfloor$. Its key-delimiter cell emits the tie words with the following ordinals. The first word of each run is also tagged with its raw length and identifier. A fixed-width sort by validity, identifier and ordinal places the compound keys in a text of prescribed length $\Theta(N/h)$, with trailing vacancies filled by a fixed symbol. The word alphabet may be shifted by one to reserve a suffix sentinel. Computing suffix ranks as above and sorting the marked first words by rank gives the desired run order. The target base of a run is its starting position in the output. An exclusive prefix sum of the raw run lengths gives these bases, and one indexed placement returns them to the $J + \lceil N/r \rceil$ identifier slots. For the long items, a scan in source order broadcasts each base from the fragment of relative offset zero. Adding the relative offset gives the destination of every long fragment.

Writing the short runs. Let $[a, b)$ be the local interval of a short run with target base d. In the batch array the indices of the first and last h-symbol words meeting this interval are $i = \lfloor a/h \rfloor$ and $j = \lfloor (b-1)/h \rfloor$. Batched lookups in the $2M_\ell/h$ words retrieve the two end portions, counting a word once if $i = j$. There are at most $2D_\ell$ requests, with local lookup and return indices. Masking and shifting a returned word costs $O(h \log h)$ for each of the $O(J)$ end portions. Its first local symbol offset u gives its target address $d + u - a$.

For $i+1 < j$, one indexed placement marks the first interior word $i+1$ with $(j-1, d-a)$. Distinct intervals give distinct such positions. A scan across the word slots propagates the latest mark and tests its last index, giving every interior word v the destination $vh+d-a$. The placement and lookup use $O(M_\ell/h)$ slots with local index keys and $O(h)$-bit data. In particular the full address difference $d-a$ travels as associated data. Across all bins the interior candidates occupy $O(N \log h/h)$ slots. At most N/h are genuine because each contains h symbols of the input. The end portions and the long-item lists add $O(N/r)$ slots, since $J = O(N \log h/h^8)$ and $r = \Theta(h)$. Compaction therefore leaves $O(N/h)$ genuine fragments in a prescribed prefix at cost $O(N \log h)$ including its control gates. Each fragment carries its length and target base. Shifting it by its target offset modulo h forms at most two masked contributions, each with its target word index. Sorting by this index and combining by segmented bitwise OR reconstructs the contiguous output stream, since the occupied symbol ranges at a target word are disjoint. Markers restore word order. The final word uses only its first $N \bmod h$ symbols when $h \nmid N$.

Gate count. Each batch uses a constant number of sorts with $s = \Theta(M_\ell/\ell)$ and item width $O(\ell + \log h)$, and a constant number with $s = \Theta(M_\ell/h)$ and width $O(h)$. In both cases $\log s = \Theta(\ell + \log h)$ and every sort key has $O(\log s)$ bits. The comparison-network bounds over the batches of one bin are respectively

$$O\left(N \frac{(\ell + \log h)^2}{\ell}\right), \qquad O(N(\ell + \log h)).$$

A final partially occupied batch changes these estimates by a constant factor. The scans, compactions at the local scale, masks and local packet shifts cost $O(N(\log \ell + (\log h)/\ell))$ per bin, including packet extraction. The $O(Jh \log h)$ gates for handling run ends are paid once.

The comparison bounds summed over $\ell < \sqrt{h}$ are $O(N\sqrt{h})$. For $\ell \geqslant \sqrt{h}$, the relation $\log s = \Theta(\ell + \log h)$ gives $\log s = \Omega(\sqrt{h})$ and $\log s = O(h)$. Thus $\log^{(4)} s = \log^{(4)} N + o(1)$ and $G(s) = \Theta(G(N))$ uniformly. Applying Proposition 6.4 to both sorts reduces their combined bound to $O_\varepsilon(N\ell/G(N))$ per bin, and these ℓ sum to $O(h)$. The other local costs sum to $O(N(\log h)^2)$. The global sorts, indexed operations and word-text suffix ranks cost $O_\varepsilon(Nh/G(N))$, including the final word assembly. Since $G(N) = o(\sqrt{h})$ and $h = \Theta(\log N)$, these costs sum to $O_\varepsilon(N \log N/G(N))$ gates. All work arrays, recursion lengths and sorting circuits have dimensions fixed by N. Their topology and wiring can be generated in polynomial time. The circuit can also be arranged in depth $O((\log N)^4)$. The fixed-width sorts used here have logarithmic-width keys relative to their tables when Proposition 6.4 is used, and have depth $O(h^3)$ by Section 6.3. The other sorts use comparison networks on words of $O(h)$ bits and also fit this depth. The scans and word operations have depth $O(h^2)$. Batches and bins on a given scale run in parallel, and the suffix-rank recursion has $O(h)$ successive levels. □

E Lower Bounds for Restricted Circuit Constructions

The n cyclic rotations of n items of b bits send each data bit to n distinct destinations. The shifting-graph bound of Pippenger and Valiant [PV76] therefore gives $\Omega(nb \log n)$ edges for a network that realizes these rotations by vertex-disjoint paths. To count binary selectors directly, we adapt Chashkin's argument for cyclic shifts of real numbers [Cha07]. Sergeev [Ser20, Theorem 3] used a related path argument for shifts of bit vectors in monotone circuits. We then bound the number of lanes in the two-shift construction of Lemma 6.1.

Proposition E.1 (Sorting by binary selectors)**.** *Let* $n \geqslant 2$ *and* $b \geqslant 1$ *be integers. Suppose a circuit sorts every permutation of a fixed set of* n *distinct keys from a totally ordered domain, carrying an arbitrary* b*-bit string with each key. If its data outputs are formed from input data bits by binary selectors whose controls depend only on the keys, then the circuit uses at least* $nb \log n$ *selectors, even with free control and unrestricted fan-out.*

Proof. For a fixed set of keys, consider the n cyclic rotations of their sorted order. For a given output bit and a given rotation, the selector choices trace a path back to an input. Distinct rotations lead to distinct source bits because all data bits can be chosen independently. The strings of binary choices on these n paths form a prefix code because reaching an input ends the path. By Kraft's inequality their mean length under a uniform rotation is at least $\log n$. For each rotation, paths from distinct data outputs share no selector, since from a shared selector both would reach the same source. Summing the mean lengths over the nb outputs gives the asserted size bound. □

When n is a power of two and both fields of each item have $\log n$ bits, the input length is $N = 2n \log n$ and the bound is $\Omega(N \log N)$. The polynomial multiplication circuit for the shifts in Lemma 6.1 can combine data bits by field operations.

Proposition E.2 (Number of lanes for two opposite shifts)**.** *Let* $q \geqslant 8$ *be a power of two. Let* $X = [q]$ *with arithmetic modulo* q *and let* Y *be a nonempty finite set of lanes. For each* $a \in X$ *let* g_a *be a permutation of* Y*. Suppose that for every permutation* π *of* X *there exists a function* $f_\pi : Y \to X$ *such that the composition*

$$(x, y) \longmapsto (a, y) \longmapsto (a, g_a(y)) \longmapsto \big(a - f_\pi(g_a(y)), g_a(y)\big), \qquad a = x + f_\pi(y),$$

sends every $(x, y) \in X \times Y$ *to row* $\pi(x)$*. Then* $|Y| \geqslant q^{(q-4)!}$.

Proof. Writing $c_a(z) = a - z$ and substituting $x = a - f_\pi(y)$ gives

$$f_\pi(g_a(y)) = (c_a \pi c_a)(f_\pi(y)).$$

For each $(q-3)$-cycle τ on $\{3, \dots, q-1\}$, let $\pi_\tau = (0\,1\,2)\tau$. There are $D = (q-4)!$ such cycles. Since their length is odd, the π_τ belong to the alternating group A_q. Let F be the free group on symbols s_a, $a \in X$, and define $\phi_\tau : F \to A_q$ by $s_a \mapsto c_a \pi_\tau c_a$. As $3 \nmid (q-3)$, the images of s_a^{q-3} are three-cycles on consecutive triples, independent of τ. They generate A_q by induction. The first triple generates A_3, and adjoining the next triple extends the action transitively to one more point while its stabilizer contains the preceding alternating group. Thus every ϕ_τ is onto. If two such maps had the same kernel, an automorphism of A_q would carry one to the other. It would fix each of the generating three-cycles, hence be the identity. The image of s_0 then determines τ.

We claim that the joint image of the D maps is A_q^D. In general, let G be a finite nonabelian simple group and consider epimorphisms to G with distinct kernels. By induction their joint image projects onto G^{D-1} in the first $D-1$ factors. The kernel of that projection has last-coordinate image normal in G. If this image is G, the joint image is the full product. Otherwise the last coordinate factors through a surjection $G^{D-1} \to G$. The individual factor images are normal in G and commute pairwise, so only one maps onto G, by an automorphism. This would repeat a kernel. The claim follows since A_q is simple for $q \geqslant 5$.

Finally, the map $y \mapsto (f_{\pi_\tau}(y))_\tau$ has nonempty image in X^D. By the identity above, each g_a acts on this image by the tuple $(\phi_\tau(s_a))_\tau$. The image is therefore invariant under A_q^D, which acts transitively on X^D. Hence the map is onto and $|Y| \geqslant q^D$. □

In particular $\log\log|Y| = \Omega(q \log q)$, matching the order of $\log\log(q^{q!})$ in Lemma 6.1. If the item width is $O(\log n)$ and every lane carries at least one bit, then $\log q = O(\log^{(4)} n)$ for this two-shift construction. In the tape routing of Lemma 3.3, the pairing inside each intermediate row depends on π. The fixed maps g_a in the proposition are what distinguish its hypothesis from that construction.

F Reductions Between Shifting, Routing and Sorting

F.1 Cyclic Shifts and Stable Sorting in Boolean Circuits

In the circuit size analysis, polynomial multiplication enters only through the cost of programmable cyclic shifts. Conversely, shifts reduce to stable sorting even when only a single bit of each item participates in the comparison. A *nonuniform* size bound allows a separate choice of circuit for each input size, with no algorithm needed to construct the family.

Corollary F.1. *The following assertions are equivalent for nonuniform bounded-fan-in Boolean circuits.*

1. *A Q-bit string can be cyclically shifted by a given offset in $o(Q \log Q)$ gates for powers of two Q.*
2. *Stable sorting of n items with $O(\log n)$-bit keys and $O(\log n)$-bit associated data uses $o(n(\log n)^2)$ gates.*

Proof. For the forward implication, let $k = \lceil \log n \rceil$. We pad the items to a common width of $\Theta(k)$ bits, at least k. Fixed output wires discard the padding. Let $S(q)$ be the minimum size of a circuit that cyclically shifts q bits by a given offset. For $q \geqslant 2$ its q outputs compute distinct functions, each depending on more than one input bit, so $S(q) \geqslant q$. In Proposition 6.3 the destination digits have t bits and the shifts have length $q = 2^{3t}$. Since t tends to infinity, the cost per bit at each digit is $O(S(q)/q) = o(t)$. Including the terminal sorts and lane restoration, the complete permutation costs $o(nk^2)$ gates. In the proof of Proposition 6.4, division of the stable-position recurrence by $2^k k^2$ now gives a recursive coefficient tending to $1/2$ and an additive term tending to zero. Induction first bounds the normalized sequence, and its limit superior is then at most half itself. This gives the claimed sorting size.

Conversely, for a power of two $Q \geqslant 2$, let L be the largest power of two at most $\log Q$. We divide the input into $m = Q/L$ consecutive words of L bits each. The rotation offset has the form $Ls + r$ with $0 \leqslant r < L$. The final s words receive key zero and the others key one. Computing these bits from s costs $O(m \log m) = O(Q)$ gates. Stable sorting on this single bit moves the suffix to the front and preserves the order of both parts. As $L = \Theta(\log m)$, the cost is $o(m(\log m)^2) = o(Q \log Q)$. A barrel shifter on each pair of adjacent words, using cyclic adjacency, selects its L output bits at the residual offset r. In total this costs $O(Q \log L) = o(Q \log Q)$ and completes the rotation. □

In the forward direction it even suffices that $S(Q)/(Q \log Q)$ tends to zero along an unbounded subsequence of powers of two. Rounding $\log Q$ down to a multiple of three loses at most a factor four in length, since an input to the shorter shifter can be repeated periodically in the longer one. For each n, let $q = 2^{3t}$ minimize $S(q)/(qt)$ over $1 \leqslant t \leqslant \lfloor (\log^{(4)} n)/32 \rfloor$. The minimum tends to zero by the subsequence assumption, so the selected t tends to infinity and still meets the upper bound on the number of lanes. The preceding proof then applies.

For delimited inputs, two items suffice for the reduction from cyclic shifts to stable sorting. Given a Q-bit input AB and the cut between the two substrings, sorting $(1, A), (0, B)$ writes

$(0, B), (1, A)$. A one-hot decoding of the cut and its prefix mask indicate the headers and delimiters. At every other position the circuit selects an input bit at one of a constant number of fixed offsets. The same operation at the complementary cut removes the headers and delimiters from the output. The two conversions cost $O(Q)$ gates, so linear-size circuits for two delimited items would give linear-size cyclic shifters.

F.2 Permutation Routing and Stable Sorting on Tapes

On tape a contiguous block can be rotated in linear time by copying its two segments, as in Section 4.1. The computation of stable positions gives a converse to the tape reduction in Section 5. The frequency table that the circuit reads in two different orders can be transposed on tape by routing it to known destinations.

Proposition F.2. *Let $0 < \delta < 1$ be fixed. The following statements are equivalent for deterministic Turing machines with a fixed number of tapes.*

1. *There exists such a machine that, for every $n \geqslant 2$, stably sorts n items of common width $\beta = \Theta(\log n)$ by any specified common key field in $O(n(\log n)^{2-\delta})$ time and $O(n \log n)$ space.*
2. *There exists such a machine that, for every $n \geqslant 2$, routes n items of this width within the same bounds according to any permutation of $[n]$ given in source order.*

Proof. Sorting on the destination fields gives routing by Section 5. For the converse, fix the routing machine. Padding at the end to the next power of two increases the number of items by less than a factor of two. Stability places maximal-key dummies after all genuine items. Let $n = 2^k$ be the padded size. A stable mergesort handles bounded k. We first compute stable positions for k-bit keys by the marker construction in the proof of Proposition 6.4. We use a fixed number $p > 2^{1/(1-\delta)}$ of stable digit passes, with $d = \lceil k/p \rceil$ bits per digit. For sufficiently large k, each pass uses 2^{k-d} recursive calls on 2^{d+1} keys of $d + 1$ bits.

In block order, a scan of the marker positions gives the digit frequencies. To compute each run's starting position in a stable digit sort, the machine routes the n entries into order by digit and then by block, computes exclusive prefix sums, and returns the offsets to block order. The routing addresses concatenate the block and digit indices and can be written in $O(k)$ steps per entry. The two routing calls on $2n$ slots and the scan that propagates each marker's offset then proceed as in Proposition 6.4. A final routing returns the stable positions in source order. Each nonrecursive operation is a scan of $O(2^k k)$ bits or a routing call on $O(2^k)$ items of $O(k)$ bits. All counters and additions in the scans use $O(k)$ steps per item.

Let $T(k)$ denote the time for the stable positions. The preceding bounds give

$$T(k) \leqslant p\, 2^{k-d} T(d+1) + O_\delta(2^k k^{2-\delta}).$$

After division by $2^k k^{2-\delta}$, the coefficient of the recursive term is $2p((d+1)/k)^{2-\delta}$, which tends to $2p^{\delta-1} < 1$. Induction gives the claimed time. A routing call by the resulting positions moves the original items. Keys of $O(k)$ bits use a constant number of stable passes on fields of at most k bits.

Within each pass the parent reads its blocks successively and appends the returned positions in block order. On a fixed set of transfer tapes it stages the keys of one child, which copies them to its own workspace and returns its output on the same tapes. Two stacks represent each simulated tape, one on either side of its head. A marker on each stack identifies the start of the child's frame, so popping the frame restores the saved head positions. The current call uses $O(2^k k)$ cells for its own arrays and the linear-space router. Only one child is active at a time, and the quantities $2^k k$

decrease geometrically down the recursion. Hence the total space is $O(2^k k)$ on a fixed number of tapes. Copying and clearing a child's frame costs $O(2^d d)$ steps, or $O(2^k k)$ over all children of one pass, as allowed in the recurrence. □

The column layouts of Section 5.1 extend this implication to arbitrary widths.

Corollary F.3. *Let $0 < \gamma \leqslant \delta < 1$ be fixed with $\gamma < 1/2$, and suppose the routing bound of Proposition F.2 holds for this δ. There exists a Turing machine with a fixed number of tapes that, given $Q \geqslant 2$ strings of common width $B \geqslant 1$ and their distinct destinations in $[Q]$ in source order, routes the strings in $O(N(\log Q)^{1-\gamma})$ time and $O(N)$ space, where $N = Q(B + \lceil \log Q \rceil)$ includes the destinations.*

For the input format of Theorem 1.1, there exists a Turing machine with a fixed number of tapes that stably sorts $n \geqslant 2$ pairs of total encoded length N by their keys in $O(N(\log n)^{1-\gamma})$ time and $O(N)$ space.

Proof. Let $k = \lceil \log Q \rceil$. Padding with dummy strings destined for the unused positions allows us to assume $Q = 2^k$. Let L be the least power of two with $Lk \geqslant B$. Each row contains L chunks of k bits, including trailing padding, and each chunk carries a copy of its row's k-bit destination. The table occupies $O(N)$ bits. Its dimensions are computed once in time polynomial in $k + \log B$, hence in $O(N)$ time. On a separate tape the machine keeps the destination of the current row, copying and rewinding its k bits once per chunk. Initializing length counters costs $O(k + \log B)$ per row, and their successive decrements cost amortized constant time.

The column layouts of Corollary 5.1 allow all Q chunks of each column to be routed together. Let $i \in [Q]$ index the row and $j \in [L]$ the chunk within it. If $L \geqslant Q$, the chunk index has the form $j = gQ + s$ with $s \in [Q]$. The interchange of (1) changes row order (i, g, s) to (s, g, i). The destination within each column is the original destination. If $1 < L < Q$, writing $i = a(Q/L) + b$ gives the interchange $(a, b, j) \mapsto (j, b, a)$. A destination $a'(Q/L) + b'$ thus has physical position $b'L + a'$ in the column. Exchanging its two bit fields costs $O(k)$ time per chunk. For $L = 1$ the single column is already in row order. A counter on $k + 1$ bits marks each group of Q slots. The routing algorithm processes successive columns in $O(Nk^{1-\delta})$ time and reuses its workspace. Reversing the interchange restores row order and a scan removes the labels and padding. The fields have at most k bits, so the two interchanges cost $O_\varepsilon(Nk^{1/2+\varepsilon})$ time by Section 2.3, uniformly in the intervening field. Choosing $0 < \varepsilon \leqslant 1/2 - \gamma$ gives the first assertion, including for bounded Q and arbitrarily large B. For $B \leqslant Q^C$ with fixed C, even the full saving δ carries over to routing. In this range $\log(QL) = O_C(k)$, so a single call to the router can send all QL chunks directly to the addresses $(\pi(i), j)$, where j is the chunk index in row i and $\pi(i)$ is its destination.

For sorting, write $h = \lceil \log n \rceil$ and first suppose that all items have width $\beta \geqslant h$ and keys of $O(h)$ bits. Sorting key–position pairs by key with Proposition F.2, numbering the sorted slots, and sorting back by input position gives each item its stable destination. Routing the original items as above costs $O(n\beta h^{1-\gamma})$ time and $O(n\beta)$ space, including both sorts. For longer keys the doubling construction of Corollary 5.1 sorts columns of $O(h)$-bit pairs. The column volumes sum to $O(n\beta)$, and each interchange costs $O_\varepsilon(h^{1/2+\varepsilon})$ per bit. Sorting once by the resulting ranks gives the same bounds. For $\beta < h$, the proof of Corollary 5.2 uses constant-factor padding, at most $2 \log h$ binary passes, or sorts of logarithmic-width batches followed by one sort of their equal-key pieces. The sorts process $O(n\beta)$ bits in total on lists of at most $2n$ items, and $\log h = O(h^{1-\gamma})$.

Finally, the bins of Section 5.2 contain at most $2n$ pieces each and have total volume $O(N)$. The fixed-width sorts within them cost $O(Nh^{1-\gamma})$ time altogether, and distribution and merging cost $O(N \log^{(2)} n)$. Within each bin the sorts preserve the order of equal-key pieces, and equal keys belong to the same bin. Processing the bins successively uses $O(N)$ space. □

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il