Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 331 (2026)

Sorting in $o(N \log N)$ Time on a Multitape Turing Machine

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 requires $\Omega(N \log N)$ 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.

*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 Algorithm . . . 5
- 1.3 Organization of the Paper . . . 6

2 Preliminaries **7**
- 2.1 The Tape Model . . . 7
- 2.2 The Cost of Moving Whole Keys . . . 7
- 2.3 Interchanging Two Address Fields . . . 7

3 Local Permutations of Binary Strings **8**
- 3.1 A Balanced Word for All Relabelings . . . 8
- 3.2 Routing the Pieces of a Binary String . . . 9

4 Radix Sorting **10**
- 4.1 Distributing One Digit . . . 10
- 4.2 Gathering Items by Digit . . . 12
- 4.3 Proof of Theorem 1.2 . . . 13

5 Permutation Routing and General Inputs **14**
- 5.1 Removing the Width Restrictions . . . 14
- 5.2 Proof of Theorem 1.1 . . . 16

6 Open Problems **17**

AI Use Disclosure **17**

Bibliography **18**

A Wide Binary Strings **20**

B Implementation on Two Writable Tapes **23**
- B.1 Copying and Merging Streams . . . 23
- B.2 Address-Field Interchange . . . 24
- B.3 The Sorting Passes . . . 25

C The Movement Bound for Bit Copies **26**

1 Introduction

1.1 Earlier Algorithms and Main Results

How fast can we sort numbers on a fixed multitape Turing machine? 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, Section 8.1]. On tapes, the machine must also move the keys and any associated data into sorted order. On a binary input of length N, this movement alone can cost $\Omega(N \log N)$ steps when every output bit must be copied from the corresponding bit of its input occurrence (Section 2.2). We call a sorter that orders keys by comparison and copies them in this way a *whole-key comparison sorter*.

Our algorithm may combine bits from several keys as it moves the items. With this freedom it sorts in $o(N \log N)$ tape steps in the worst case, carrying each key's associated data. The sort is *stable*, meaning that items with equal keys appear in their original relative order. We use lexicographic order on strings, which also allows us to sort finite expansions in numerical order.

Throughout, log denotes the base-two logarithm, and $\log^{(j)} x$ denotes its j-fold iterate. Thus $\log^{(2)} n = \log \log n$ and $\log^{(3)} n = \log \log \log n$. In asymptotic bounds, including width hypotheses, we use $\max\{1, \log x\}$ for the logarithm of a size parameter x, including at each iteration. In identities and integer bit counts, logarithms have their usual values.

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, Corollary 3.1] 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, Theorem 3.1 and 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. They also gave 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 multihead tape machines 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, Corollary 1.2(i)] 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 [Ope26b, 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. Sorting also exchanges fields separated by other address bits. 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 [Ope26b, Theorem 2.1]. We obtain this bound on one machine for all ε (Section 2.3).

Main results.

Theorem 1.1 (Stable sorting). *Let Σ be a finite alphabet with a total order and let $(k_1, x_1), \ldots, (k_n, x_n)$ be $n \geqslant 1$ pairs of strings over Σ, with total input length N including delimiters. There exists a deterministic Turing machine with a fixed number of tapes that, on every such input, outputs the pairs in lexicographic order of their keys k_i, preserving the input order among pairs with equal keys. The machine depends only on Σ and uses*

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

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

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

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

The same 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.

For the fixed-width results, symbols are represented in alphabet order by binary words of a common length, and widths are measured in bits. Equal-width binary keys in most-significant-bit order 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 deterministic 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 smallest width admitting 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. Every deterministic whole-key comparison sorter (Definition 2.1) 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.

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, Theorems 3.1 and 3.2] gave a counterexample to that conjecture and 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, 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, 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 Algorithm

For the fixed-width algorithm, we append the input position to each key to break ties. Each bucket consists of items agreeing on the prefix read so far. In a bucket of at least 2^{4t} items, we group them by their next t key bits. After padding, consecutive blocks of equal length form the columns of a table. A cyclic shift of each column changes which items meet in a row.

In every row we reserve the same interval of positions for each digit, with its size determined by the digit's frequency in the bucket. We choose the rotations by conditional expectation so that, over the whole bucket, all but an $O\left(2^{-t}\right)$ fraction of the items fit in their assigned intervals. We perform the rotations and use a bucket-wide address-field interchange to make each row contiguous on tape. Within a row, we assign each item a destination in the interval for its digit while space remains, and assign the excess items separate destinations. Filling the remaining positions with dummies gives a permutation π of the positions in the row.

Each item is split once into indexed pieces. To carry out π, we associate an offset with each piece position, using the same list of offsets for every item. A piece of the item at x with offset c first moves to $a = x + c$, where it is assigned to a position of offset $a - \pi(x)$. Subtracting the new offset sends the piece to $\pi(x)$. The arithmetic is modulo the number of items in the padded row.

At a fixed a, the piece arriving with offset c came from $a - c$. Since π is a permutation, the destination offsets at a form a relabeling of the original list. We choose a list in which every offset occurs equally often and which shares a long subsequence with every relabeling. The matched pieces can then be copied in order. We route the remaining $O(1/t)$ fraction of the pieces in $O(t)$ radix passes, at linear cost in the volume of the row. All pieces of each item now reach its assigned destination, and their indices let us read later digits in any piece order.

A second bucket-wide interchange gathers the reserved intervals from all rows in digit order. We group the excess items by digit with binary radix passes and append them to the corresponding buckets. For t a sufficiently small constant multiple of $\log^{(3)} n$, computing the block shifts, row permutations and matchings costs linear time in the volume processed. The interchanged fields have $O(t)$ bits, so one level of digit partitioning costs $O\left(Vt^{1/2+o(1)}\right)$ for $V = n\beta$ input bits. Over $O(\log n/t)$ levels the total is $O\left(V \log n/t^{1/2-o(1)}\right)$, including one restoration of piece order and at most one terminal mergesort per item. For arbitrary-length inputs, we sort equal-width pieces within groups of similar key lengths, reassemble the items and merge the sorted lists. The pieces occupy $O(N)$ bits in total.

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, defines whole-key comparison sorters and states the address-field interchange used by the algorithm. 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 Theorem 1.1. Section 6 poses two questions about sorting logarithmic-width items and routing strings to given destinations.

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.

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

2 Preliminaries

2.1 The Tape Model

We write $[a] = \{0, \dots, a-1\}$ and store arrays in lexicographic order, with the last index varying fastest. The machine has a fixed finite 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, the dimensions and the location and length of the key field are given in binary without leading zeros. For n items of width β, the input volume is $V = n\beta$ bits. 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 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 N counts all symbols, including delimiters. Running time includes preprocessing, head movements and output, and space is measured across all tapes. Bounds expressed solely in terms of N are worst-case bounds over inputs of encoded length at most N.

2.2 The Cost of Moving Whole Keys

Definition 2.1 (Whole-key sorters)**.** A *whole-key comparison sorter* determines the order of the keys by comparisons and may use ranks and indices to direct its copies. The *rank* of a key in a list is the number of distinct smaller key values in that list. Bit j of input occurrence i has label (i, j). Local copies inherit their source label, and bits computed by arithmetic or comparison have no label. The output is a permutation of the input occurrences in which bit j of occurrence i still has label (i, j).

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 [Ope26b, Theorem 2.1] gives a deterministic 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 deterministic 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 a template 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 template.

Throughout, a factor $x^{1/2+o(1)}$ in a time bound means that the same machine attains the bound with $x^{1/2+\varepsilon}$ for every fixed $0 < \varepsilon < 1/2$. The same convention applies to a denominator $x^{1/2-o(1)}$, with exponent $1/2 - \varepsilon$. Only the time constant may depend on ε.

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 $y \in [M]$. We write $\mathrm{Sym}([q])$ for the set of these permutations.

A *subsequence matching* pairs equal letters in two words, in increasing order of position in both. For words U, V of length $M \geqslant 1$, we write $d(U, V) = 1 - h/M$, where h is the length of a longest common subsequence. Thus $d(U, V)$ is the smallest fraction of positions left unmatched in either word.

Lemma 3.2. *For every integer $q \geqslant 2$ and $0 < \varepsilon \leqslant 1/2$, there is 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. Given q and $\lceil 1/\varepsilon \rceil$, a fixed multitape machine constructs W in $(qM)^{O(1)}$ time and space. Given also $\gamma \in \mathrm{Sym}([q])$, it finds a matching of length at least $(1-\varepsilon)M$ between W and γW within the same bounds.

Proof. The initial word lists the q letters once. Let $p = \lceil 1/\varepsilon \rceil$ and $A = 4qp$. Each iteration forms a new word W' from the current word W, with one section for each of the $h = \binom{q}{2}$ transpositions s, in a fixed order. The section for s consists of A alternating copies of W and sW. Relabeling preserves the property that each successive block of q letters contains every letter once. Thus W' also has this property and is balanced.

Let $D(W) = \max_g d(W, gW)$ over all transpositions g. In comparing W' with gW', corresponding copies outside the g-section have distance at most $D(W)$. For sW and gsW this follows by relabeling both by s^{-1}, since $s^{-1}gs$ is a transposition. In the g-section, shifting the pairing by one copy gives $A - 1$ pairs of identical copies. The matchings concatenate in order, so

$$D(W') \leqslant (1 - 1/h) D(W) + 1/(Ah).$$

For the initial word, deleting the two letters moved by a transposition gives $D(W) \leqslant 2/q$. For the word W after $J = h\lceil \log(4p) \rceil$ iterations, the recurrence gives $D(W) \leqslant (2/q)e^{-J/h} + 1/A \leqslant 1/(qp) \leqslant \varepsilon/q$. Every permutation is a product of at most $q-1$ transpositions. The triangle inequality for d follows by composing two matchings on their common positions in the middle word. Together with invariance under simultaneous relabeling, it gives $d(W, \gamma W) \leqslant (q-1)D(W) \leqslant \varepsilon$ for every $\gamma \in \mathrm{Sym}([q])$. The length is $q(Ah)^J$. Since $p < 2/\varepsilon$, this gives the stated estimate, and in fact $\log M = O\left(q^2 \log(q/\varepsilon) \log(1/\varepsilon)\right)$. 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 a fixed alphabet, the existence of balanced words 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 $a \mapsto (ag_1, \ldots, ag_{q!})$ lists every permutation once, so it 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 $q = 2^k$ for an integer $k \geqslant 1$. Row indices and offsets are computed modulo q. Let W be the word of Lemma 3.2 with $\varepsilon = 1/\max(2, k)$.

For a positive integer b, the array has coordinates $(x, y, v, z) \in [q] \times [M] \times [q] \times [b]$. In row x, the qb consecutive bits at position y form a *lane*, with v 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. *Given any permutation π of $[q]$, the machine realizes a bijection of bit positions of the form*

$$(x, y, v, z) \longmapsto (\pi(x), \lambda_x(y), v, z), \tag{2}$$

where each λ_x is a permutation of $[M]$. The permutations λ_x and the designation of positions as matched or unmatched 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 $O\left(\mathcal{V} k^{1/2+o(1)}\right) + (qM)^{O(1)}$ time and uses $O(\mathcal{V}) + (qM)^{O(1)}$ 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

$$\phi_a(c) = a - \pi(a - c). \tag{3}$$

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. Stable binary radix passes group the unmatched lanes by their k-bit destination colors. 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 $O(\varepsilon\mathcal{V}k) = O(\mathcal{V})$, 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 (2). 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 tape templates mark 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 $\log(2 + qMb)$. Their total cost is at most $(qM)^C(\log(2 + qMb))^C$ 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 $O(qMb)$. The time bound follows from Section 2.3, and the row workspaces are reused successively. □

During sorting we pad the augmented item at its 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 $(qM)^{O(1)}(\log(2+\beta))^{O(1)}$ 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 has s_c slots of K item positions reserved for c, giving it capacity T_c, where

$$s_c = \lceil 2\mu_c/K \rceil + 1, \qquad T_c = Ks_c. \tag{4}$$

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 $(X(r,c) - T_c)_+$, 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 $O\left(mt^2\right)$ 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. Both s_c and the weights below can be computed from m_c, R, K by integer arithmetic, comparing products to find quotients of size $O(K)$.

Lemma 4.2. *For every input one can choose and apply one cyclic shift per block so that the total excess satisfies $E = O(m/K)$. Computing the shifts from the digit streams costs $O\left(mKt^{O(1)}\right)$ bit steps on fixed tapes. Rotating the items and laying them out in contiguous rows costs $O\left(m\beta t^{1/2+o(1)}\right)$ additional time.*

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

$$1 \leqslant d_c \leqslant 4K, \qquad 2L \leqslant d_c(\mu_c + K) \leqslant 4L. \tag{5}$$

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 $\ell \in \mathbb{Z}/R$, the current block contributes $B(r+\ell, c)$. Up to the common factor two, the only term of the conditional expected potential depending on ℓ is

$$\sum_{r,c} d_c S(r,c) B(r+\ell, c). \tag{6}$$

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 (6) 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 (6) is

$$\frac{2^j}{R} \sum_{u \in [2^j],\, c \in [K]} d_c S_j(u,c) B_j(u+e, c). \tag{7}$$

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

$$O\left(K\sum_{j=0}^{\nu} 2^j (t+\nu-j)^a\right) = O\left(RKt^a\right). \tag{8}$$

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 $LR < 2m$, (8) 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 (4) and (5) give

$$E = \sum_{r,c}(X(r,c)-T_c)_+ \leqslant \frac{\Phi}{8L} \leqslant \frac{R}{8L}\sum_c d_c\mu_c \leqslant \frac{RK}{2} = O(m/K). \tag{9}$$

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 $\min(X(r,c),T_c)$ 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 (9). 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 allow also for the computation of the matchings, we use $u = \log n$ and, for sufficiently large n,

$$t = \lfloor \tfrac{1}{16} \log^{(3)} n \rfloor, \qquad K = 2^t, \qquad q = 8K^2. \tag{10}$$

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

$$\log M = O\left(q^2(\log q)^2\right) = o(\log^{(2)} n), \qquad (qM)^{O(1)} = u^{o(1)}. \tag{11}$$

Let $\widehat{\beta}$ be the item width after adding its copied key and input position. Then $u \leqslant \widehat{\beta} = O(\beta)$ under the hypotheses of Theorem 1.2. By (11), 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 $O(\beta)$.

By Section 3.2, the labels, padding and header add $O(M\log M + qM + t)$ 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 $(qM)^{O(1)} = u^{o(1)}$ and every fixed power of $\log(2+\beta)$ is $O(\sqrt{\beta})$. Counts local to a bucket of m items use $O(\log(2m))$ 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 $O(u/t)$ 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. The total time over all levels is $O_\varepsilon\left(Vu/t^{1/2-\varepsilon}\right)$. The terminal sorts cost $O(Vt)$ and restoring the piece order costs $O(V\log M)$. Both are $o(Vu/t^{1/2-\varepsilon})$, since $t = \Theta\left(\log^{(3)} n\right)$ and $\log M = o(\log u)$. The same machine satisfies this estimate for every fixed ε,

giving the running time

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

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 $u^{o(1)}$ 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.

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 deterministic 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 deterministic 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 $O\left(\log^{(2)}(2 + \ell_i)\right)$ bits. Sequential length counts give the tags in time polynomial in $\log(2 + \ell_i)$ and $\log(2 + T)$ per pair, hence in $O\left(\sum_i (1 + \ell_i + T)\right) = O(N)$ 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 symbols of Σ from one in alphabet order. Each symbol number is written in $b = \lceil \log(1 + |\Sigma|) \rceil$ bits, most significant first. In bin j, a zero block terminates each key and trailing zeros fill the remaining bits up to width $\lambda_j = bH_j$. At the first differing symbol, the codes compare in alphabet order. 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 λ_j-bit key per pair. Since Σ is fixed, 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 $(\log(2N))^{O(1)} = O(N)$ further steps.

For each nonempty bin, the capacity of a piece is $p_j = \lceil N_j/n_j \rceil \geqslant \lambda_j$. 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 $O(\log(2 + p_j))$ 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 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 6.1 (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 $0<\delta<1$?*

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. For the arbitrary-length keys and associated data of Theorem 1.1, the corresponding goal is $O\left(N(\log n)^{1-\delta}\right)$ steps on an input of length N.

Question 6.2 (Routing to given destinations)**.** *Each of Q strings of B bits is given with its position in the output. Are there constants $a>0$ 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 $B \geqslant Q^a$?*

Here Q is a power of two, and the destinations form a permutation listed in input order. The time includes computing the moves. For $B \geqslant 2^{Q^\alpha}$ with fixed $\alpha>0$, Theorem A.1 gives $(\log Q)^{1/2+o(1)}$ steps per bit.

A positive answer to Question 6.2 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 6.1. 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) in developing the balanced-word routing construction and the method for choosing block shifts deterministically. ChatGPT also assisted with the two-tape implementation, the tape complexity analysis and with the presentation of the proofs.

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.

[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. 5.

[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. 6.

[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. 20.

[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. 22.

[BP25] Andrew Best and Yuval Peres. Edit distance in substitution systems. arXiv:2501.00440v3, 2025. p. 8.

[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.

[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. 6.

[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. 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. 5.

[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. 3 and 23.

[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. pp. 3, 5, and 7.

[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 Comput-*

er 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. 20.

[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. 14.

[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. 5.

[LLL26] Baochun Li, Zongpeng Li, and Sirui Liu. Undirected multiple-unicast network coding and faster external-memory integer sorting. arXiv:2610.20079v3, 2026. pp. 5a, 5b, 5c, 5d, and 10.

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

[Ope26b] OpenAI. The logarithmic exponent of integer multiplication on tapes, November 2026. Preprint, November 24 manuscript. pp. 4a, 4b, 7, 24a, and 24b.

[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.

[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. 3.

[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. 3 and 14.

[Sto73] Hanns-Jörg Stoß. Rangierkomplexität von Permutationen. *Acta Informatica*, 2(1):80–96, 1973. doi:10.1007/BF00571465. p. 7.

[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. 3.

[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. 23.

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 deterministic 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 now give the switch layers for Theorem A.1. We prove the theorem by showing how to recover the original order of the pieces after the last layer.

Proof. Let $\gamma = a_0/b_0$ be a fixed rational in $(0, \min\{\alpha, 1\})$. The recursive algorithm applies when $\lfloor \log B \rfloor \geqslant \lceil Q^\gamma \rceil$, a condition the machine tests by comparing $(\lfloor \log B \rfloor)^{b_0}$ with Q^{a_0}. 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 $D = \lceil h/s \rceil$ fields of at most s bits each. A mixed-radix Beneš network [Ben64] realizes the permutation in $H = 2D - 1$ layers, each permuting one field within every fiber of the other fields. 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

$$(v_f, v_0, p, e_0, v_1, z), \qquad v_f, v_0, v_1, e_0 \in [q], \quad p \in [M/q], \quad z \in [b].$$

In a layer whose switch acts on $e \leqslant s$ address bits, we interchange those bits with the low e bits of v_f. In the former v_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 v_f. Interchanging a_f and v_0 gives the field order (v_0, a_f, y, v_1, z). For every v_0, the routing procedure of Lemma 3.3 acts on (a_f, y, v_1, z) with the same common-subsequence matching and the same order for the unmatched pieces. Reversing the interchanges restores v_f, v_0, v_1, z and sends the string to its prescribed address. For fixed input string and v_f, the resulting permutation of y and the matched positions are independent of v_0, v_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 (3). 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 $u = \log n$ and $V = n\beta$. There exists a deterministic multitape Turing machine that stably sorts every such input using*

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

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 $h = \lceil u \rceil$. For keys of $O(h)$ bits, we use the radix sorter of Section 4.3 with digit width $t = \lfloor \frac{1}{16} \min\left\{u, \log^{(2)} \beta\right\} \rfloor$ 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 $O(u/t)$ digit levels cost $O_\varepsilon\left(Vu/t^{1/2-\varepsilon}\right)$ 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 $O(mut)$ 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 $O_\varepsilon\left(m\beta u/t^{1/2-\varepsilon}\right)$, since $t \leqslant u$ 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 $\beta \leqslant u^4$, 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 $\beta > u^4$. 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. Each tape has one head shared by all its tracks. 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 a period of length d, borrow chains, returns to the low bit and the reset cost $O(d)$ in total.

Two lists of delimited strings can be interleaved in the order $A_1, B_1, \ldots, A_r, B_r$ in time and space linear in their combined length D. 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 2.6–2.14]. 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 $A = \mathrm{PGB2}^{2u}$ be the volume of the layout (p, x, g, y, z) in Section 2.3. We implement the recursion of OpenAI [Ope26b, Theorem 2.1] on two tapes. For a fixed $1/2 < \alpha < 1$, the recursion uses 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. For a spectral gate comparing 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 $\mathrm{poly}(1 + f + \log S) = O_{\alpha}(b^{f} S)$ steps. Countdowns mark the runs within the same bound.

The triangular address changes proceed from the last target field to the first. Each earlier-field summand gives a cyclic shift whose cut is computed once per fiber as above. 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 $-kb^{f}$ mod 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. Only the first kind of call parks an array, and its child volume is at most $2v/W$. Hence the parked volumes sum geometrically. Each descriptor stores the dimensions and a return instruction. Over a path they occupy $O_{\alpha}((1+\log(e+1))(1+\log v)) = O_{\alpha}(v)$ 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 $T(e) = O_{\alpha}((1+e)^{\alpha})$ in $O_{\alpha}(v)$ 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 [Ope26b, 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 $\mathrm{poly}(u) = O(A)$ 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 $O(A)$ per pair. Since $C_j \leqslant \sqrt{1+u} \leqslant 4^d$, the compact run visits at most $C_j A/4^d \leqslant A$ cells. Reusing this space for all pairs gives $O(A)$ space. Since $C_j \leqslant (1+u)^{1/(j+2)}$, the total time is

$$O\big(A\sqrt{1+u} + C_j A(1+u)^{\alpha_j}\big) = A(1+u)^{1/2+o(1)}.$$

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 (7), 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 (8) 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 $\lambda_j \leqslant p_j$, 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 and $s = q^2 b$ the volume of a child. The children occupy successive intervals. By the volume bounds in that proof, $s \leqslant A/8$ and their total is 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 an interval of CA cells per tape, where its own buffers use cA cells and C satisfies $cA + CA/4 \leqslant CA$. Within a batch, the heads copy the inputs to successive slots of Cs cells. 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 i denote the input position of a key. For each 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 $0, \ldots, n-1$.

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 length at most L realize at most C^{L+1} permutations, for a machine constant $C > 1$.

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.

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