Electronic Colloquium on Computational Complexity, 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

$$O\left(\frac{N \log n}{(\log\log\log n)^{\eta}}\right) \quad \text{time and} \quad O(N) \quad \text{space}$$

for an explicit absolute constant $\eta > 0$. 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 **6**
- 2.1 The Tape Model . . . 6
- 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 **11**
- 4.1 Distributing One Digit . . . 11
- 4.2 Gathering Items by Digit . . . 13
- 4.3 Proof of Theorem 1.2 . . . 14

5 Permutation Routing and General Inputs **15**

6 Wide Binary Strings **18**

7 Open Problems **22**

AI Use Disclosure **22**

Bibliography **23**

A A Slight Improvement in η **25**

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 requires at least $\lceil \log(n!) \rceil$ comparisons in the worst case, since a binary comparison tree must distinguish all $n!$ input orders. On tapes, the machine must also move the keys and any associated data into sorted order. If N is the total encoded input length in bits, this movement alone can require $\Omega(N \log N)$ steps when every output bit must be copied from the corresponding bit of its input occurrence [Sto73, HPV75, Pau79]. We call a sorter that orders keys by comparison and copies them in this way a *whole-key comparison sorter*.

We study this question for signed integers and fractions given by finite expansions in any fixed base, with arbitrarily many digits on either side of the radix point. We encode each numeral as a binary key so that sorting the keys lexicographically sorts the numbers by value. The machine may combine bits from several keys while rearranging the items. It returns each key with its associated data. We require the sort to be *stable*, meaning that items with equal keys appear in their original relative order. Our guiding question in this work is as follows: Can a deterministic algorithm use this freedom to improve asymptotically on whole-key comparison sorting, taking $o(N \log N)$ tape steps in the worst case? We answer that question in the affirmative.

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 take each logarithm of a size parameter to be at least one, also 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, Section 8.3]. For arbitrary single-word keys, fusion trees gave a deterministic $o(n \log n)$ bound using constants that depend on the word width [FW93]. 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]. For word widths of at least $(\log n)^{2+\varepsilon}$ bits and fixed $\varepsilon > 0$, Andersson, Hagerup, Nilsson and Raman [AHNR98] gave a randomized algorithm taking $O(n)$ expected word operations.

On tapes, distributing keys among buckets requires moving their bits to the destination. For n integers of $\lceil \log n \rceil$ bits each, a word RAM can write the sorted list in $O(n)$ word operations. Writing the same list to a fresh tape takes $\Omega(n \log n)$ 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 taking $o(N \log N / \log^* N)$ 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, pp. 138–139] described a three-tape merge procedure for an encoded list of integers. The machine merges runs of successively doubled length, spending linear time at each of $O(\log n)$ levels. 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. They also gave a linear-time algorithm on a pointer machine. Salamon and Wehar [SW23, Section 8.3, Lemma 20] give the fixed-width form of the tape bound. Sorting n integers of w bits each takes $O(nw \log n)$ steps. In particular, for $w = \Theta(\log n)$, merging takes $O\left(n(\log n)^2\right)$ bit operations.

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 applies under 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. His example stores $x \oplus y$ and y in separate blocks and uses both to recover x [Pau79, pp. 328, 332–333].

Dietzfelbinger, Maass and Schnitger [DMS91] proved that sorting strings of $3 \log N$ bits each requires $\Omega\left(N^{3/2}/\sqrt{\log N}\right)$ steps on a machine with one work tape and a two-way read-only input tape.

From transposition to sorting. Consider an $r \times s$ matrix with $r, s \geqslant 2$ and entries of a common width. Write $\mathcal{V}$ for the total number of bits in its entries and $M(\mathcal{V})$ for the time to multiply two integers of at most $\mathcal{V}$ bits each on a fixed multitape Turing machine. Harvey and van der Hoeven [HvdH25, Corollary 6.2 of the full version] showed that the matrix can be transposed in

$$O\left(M(\mathcal{V}) + \mathcal{V} \log^{(2)} \max\{r, s\}\right)$$

tape steps. OpenAI [Ope26] obtained $M(\mathcal{V}) = O\left(\mathcal{V}(\log \mathcal{V})^{1-\kappa}\right)$ for a fixed $0 < \kappa < 1$. Since $\log^{(2)} \mathcal{V} = o((\log \mathcal{V})^{1-\kappa})$, the reduction gives the same time bound for transposition. For a $d \times d$ binary matrix this improves the classical $O\left(d^2 \log d\right)$ bound to $O\left(d^2 (\log d)^{1-\kappa}\right)$.

When $d = 2^u$, the address of an entry in row-major order consists of its u-bit row index followed by its u-bit column index. Transposition exchanges these two fields. Our sorting algorithm uses a procedure from the proof of the multiplication bound that exchanges two u-bit address fields in an array of $\mathcal{V}$ bits in $O(\mathcal{V} u^\tau)$ tape steps for a fixed $0 < \tau < 1$, even when other address bits separate the fields [Ope26, Lemma 12]. We write $\eta = 1 - \tau$ for the saving in the sorting bounds.

Main results.

Theorem 1.1 (Stable sorting of binary keys)**.** *Let each of $n \geqslant 1$ items consist of a binary key and an associated binary string, each followed by a delimiter. Either string may be empty. Let N be the total encoded input length in bits. For a fixed binary encoding of the tape alphabet, N is within a constant factor of the sum of the key and data lengths plus $2n$ for the delimiters. A deterministic multitape Turing machine outputs the original items in stable lexicographic order of the keys in*

$$O\left(\frac{N \log n}{(\log^{(3)} n)^\eta}\right) \quad \textit{time and} \quad O(N) \quad \textit{space},$$

where $\eta > 0$ is an absolute constant that can be made explicit. Alternatively, the running time as a function of the input length N is

$$O\left(\frac{N \log N}{(\log^{(3)} N)^\eta}\right) = o(N \log N).$$

The same bounds apply to numerical order whenever a linear-time tape procedure converts the numerals to binary keys whose lexicographic order agrees with their values, assigning equal keys to equal values. Signed finite expansions of varying precision in any fixed base admit such an encoding. The output preserves the input spelling of each numeral and the input order among equal values.

We first consider items of equal width with short binary keys. Throughout the fixed-width case we read each key most significant bit first as an unsigned integer. For keys of a common width, this order agrees with lexicographic order.

Theorem 1.2 (Fixed-width inputs)**.** *Let $n \geqslant 1$ items have common width $\beta = \Omega(\log n)$. In each item the same contiguous field of $O(\log n)$ bits holds the binary key. There is a deterministic multitape Turing machine that outputs the items intact in stable nondecreasing key order. For $V = n\beta$, its worst-case bounds are*

$$O\left(\frac{V \log n}{(\log^{(3)} n)^{\eta}}\right) \quad \textit{time and} \quad O(V) \quad \textit{space.}$$

The proof appears in Section 4.3.

For any fixed set of $n \geqslant 2$ distinct keys of width $\lceil \log n \rceil$, Theorem 1.2 sorts every ordering in $o(N \log N)$ tape steps. Here the keys occupy $n\lceil \log n \rceil$ bits and the encoded input length is $N = \Theta(n \log n)$. This is the smallest common key width that can represent n distinct values. Every deterministic whole-key comparison sorter (Definition 2.1) requires $\Omega(N \log N)$ tape steps on some ordering of the same keys.

For the logarithmic-width integers of Hopcroft, Paul and Valiant, $\log^* N = o((\log^{(3)} N)^{\eta})$ for every fixed $\eta > 0$. Theorem 1.2 therefore meets their sorting hypothesis for the fixed number of tapes used by our machine [HPV75, Theorem 2].

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] gave matching transfer bounds for comparison sorting. Farhadi, Hajiaghayi, Larsen and Shi [FHLS19] 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 a constant number of boundary transfers per interval, regardless of the distance between intervals, and permits a column that fits in memory to be sorted there without further transfers [LLL26, Lemma 7.2]. On tapes, both the local computation and the travel between intervals take time. For the digit widths t used below, we partition a bucket of at least 2^{4t} items in $O(t^{1-\eta})$ tape steps per bit, including the time to choose the block shifts and compute the permutations within rows.

1.2 Overview of the Sorting Algorithm

For the fixed-width algorithm, we append the input position to each key to break ties and read the resulting keys from the most significant end. Consider a large bucket whose items agree on the prefix read so far. One radix step groups these items by their current digit, the next t bits of the key.

We pad the bucket and divide it into consecutive blocks of equal length. View each block as one column of a table. Each row takes one item from each column. We rotate the columns to change which items meet in a row. A cyclic shift by ℓ moves the first ℓ items of a column to its end in the same order. For example, a shift by one changes (A, B, C, D) to (B, C, D, A). The tape procedure copies the suffix of the block followed by its prefix.

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.

To carry out π, we move pieces between item positions within this row. We split each item once into indexed pieces and use the same list of offsets for the piece positions in every item. Follow one piece from an item at position x in the row. If its offset is c, we first move it to item position $a = x + c$, wrapping around at the end of the row. Its destination is $\pi(x)$. At a we place it in a piece position with offset $a - \pi(x)$. The return move subtracts this offset, taking the piece from a to $\pi(x)$. All position arithmetic here 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 required 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. We copy the pieces in the common subsequence in order. The remaining $O(1/t)$ fraction of the pieces take $O(t)$ 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 offsets and matchings takes linear time in the volume processed. The interchanged fields have $O(t)$ bits, so one level of digit partitioning costs $O\left(Vt^{1-\eta}\right)$ for $V = n\beta$ input bits. Over $O(\log n/t)$ levels the total is $O(V \log n/t^{\eta})$, including one restoration of piece order and at most one terminal merge-sort 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.

1.3 Organization of the Paper

The proof of Theorem 1.1 starts with the case of fixed-width items. Section 2 describes the tape model and the address-field interchange. In Section 3 we use this interchange to route the pieces of each item to a chosen row. In Section 4 we apply this construction to group items by successive digits, proving Theorem 1.2. Section 5 extends the algorithm to keys and associated data of arbitrary lengths. Section 6 gives a faster bound for sufficiently wide items. Appendix A derives the improved exponent η from the circuit of Li, Li and Liu.

2 Preliminaries

2.1 The Tape Model

Write $[a] = \{0, \ldots, a-1\}$. We store arrays in lexicographic order on one-dimensional tapes, with the last index varying fastest. The machine has a fixed finite alphabet and a fixed number of 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. For fixed-width inputs, the machine reads the dimensions and the position of the key field in binary. For variable-length inputs, one delimiter separates a key from its associated data and another ends the pair. The running time includes preprocessing, head movements and writing the sorted items. We count the total working space across the tapes. The input and output each occupy space linear in the encoded input length. When time is expressed solely as a function of N, we take the worst case over inputs of encoded length at most N.

For n items of common width β, the input volume is $V = n\beta$ bits. In Theorem 1.2 we carry each original item alongside a copy of its key followed by its input position. Sorting by this augmented key orders equal keys by input position and hence preserves stability. The two auxiliary fields occupy $O(\log n)$ bits per item. We remove them on output.

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. Its output is a permutation of the input occurrences. To specify the copying requirement, label bit j of input occurrence i by (i, j). Local copies inherit the label of their source, and bits computed by arithmetic or comparison have no label. The output occurrence corresponding to i must have label (i, j) at every offset j.

For every fixed set of $n \geqslant 2$ distinct w-bit keys, some input order requires $\Omega(wn \log n)$ steps in this model. Following the rearrangement argument of Stoß [Sto73] and its application to sorting [HPV75, Section 3], we count moves separately for each bit offset. The counting argument uses only the copying requirement of Definition 2.1.

For each j, project every tape and bit register onto the labels of that offset, retaining only their indices i. Represent the surviving indices on either side of each head by two stacks and keep the index under the head in a register. With all heads initially at their left markers, every projection has the same input sequence $0, \ldots, n-1$ and an empty output tape. One transition touches a bounded number of cells and registers, including those entered by the heads. Across all projections it therefore induces only a bounded number of stack moves, local copies and erasures. Each such instruction names its locations from a fixed alphabet, and a copy takes its index from the indicated location.

In a run of t steps some offset thus uses $O(t/w)$ instructions. Its instruction sequence, executed from the common initial state, determines the output order of the indices. Every offset gives the same permutation of input occurrences. The $n!$ input orders require distinct such permutations, so if T is the worst running time, $n! \leqslant C^{1+T/w}$ for a machine constant $C > 1$. Taking logarithms gives the bound. Output left in place can be copied to a fresh tape in $O(T + nw)$ additional steps, giving the same lower bound.

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,

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

OpenAI [Ope26, Lemma 12] implements this interchange on a fixed deterministic multitape machine in $O(\mathcal{V} u^\tau)$ steps, where $\mathcal{V} = PGB2^{2u}$ is the array volume in bits and $0 < \tau < 1$ is an explicit rational constant. The bound is uniform in the four dimensions and includes padding and its removal. Any rational $\tau \in (0, 1)$ giving this time bound yields the sorting exponent $\eta = 1 - \tau$.

Linear space follows from the same time bound by processing slices in succession. Choose $d = \lfloor \tau \lfloor \log u \rfloor / 2 \rfloor$ using the bit length of u, so $4^d = \Theta(u^\tau)$, and fix the high d bits of each field. For every pair of patterns (i, j), a scan extracts a slice of $\mathcal{V}/4^d$ bits. Interchange its remaining fields of width $u - d$, then scan the output tape to insert the slice at patterns (j, i). Each selection pattern can be generated by two periodic bit masks, one for each field. Creating the masks by sequential repetition and rewinding them once per period costs $O(\mathcal{V})$ per slice. At width zero the slice is copied directly. Each scan costs $O(\mathcal{V})$, including rewinds, and the interchange within one slice takes $O\left(\mathcal{V} u^\tau / 4^d\right) = O(\mathcal{V})$ steps. Reusing its work tapes gives $O(\mathcal{V})$ space and $O(\mathcal{V} u^\tau)$ time over all slices.

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 a row. Permuting the rows relabels the offsets that the pieces require. We seek a word of offsets that shares a long subsequence with each of its relabelings, so that most pieces can 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 $M \geqslant 1$, 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. Start with the word listing the q letters once. Let $p = \lceil 1/\varepsilon \rceil$ and $A = 4qp$. From a current word W, make one section for each of the $h = \binom{q}{2}$ transpositions s, in a fixed order. The section consists of A alternating copies of W and sW. Concatenating the sections gives W'. Relabeling preserves the property that each successive block of q letters contains every letter once. Thus W' also has this property and is balanced.

Relabeling both words by the same permutation preserves every subsequence matching. Let $D(W)$ be the maximum of $d(W, gW)$ over all transpositions g. For a transposition g, align the corresponding copies in all sections except the g-section. If the two copies under comparison are sW and gsW, relabeling both by s^{-1} gives the transposition $s^{-1}gs$ on the second word. Thus the matched fraction in each such copy is at least $1 - D(W)$. In the g-section, shift the pairing by one whole copy. The $A - 1$ pairs of copies then coincide. These subsequences 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. To see the triangle inequality for d, compose two matchings on the positions they both use in the middle word. Each omitted middle position was unmatched in at least one of the two matchings. Relabeling invariance now 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. Using sequential scans for table access takes 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]. To see this, fix an ordering of $\mathrm{Sym}([q])$ and replace each permutation a by the list of all ag in that order of g. This substitution has length $q!$, contains every permutation in each list, and 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 $k \geqslant 1$, and use arithmetic modulo q on row indices and offsets. Take the word W from Lemma 3.2 with $\varepsilon = 1/\max(2, k)$.

For a positive integer b, index the array by $(x, y, v, z) \in [q] \times [M] \times [q] \times [b]$. 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.

Each row has width qMb bits and the array has volume $\mathcal{V} = q^2Mb$ 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 $[M]$. There is an absolute constant D such that, for row widths $qMb \geqslant (qM)^D$, the machine takes $O(\mathcal{V}k^\tau) + (qM)^{O(1)}$ time and uses $O(\mathcal{V}) + (qM)^{O(1)}$ space.

Proof. To shift the row index by the color of the lane, $x \leftarrow x + W(y)$, first interchange x with v using (1). In the resulting layout (v, y, x, z), each fixed (v, y) has a contiguous fiber of q cells of width b. Rotate it by $W(y)$ cells by splitting it into two consecutive segments and copying the segments in the other order. Interchange the two short fields back. The number of field interchanges is two and the rotations scan linear volume. The inverse shift uses subtraction and the same layout.

Write $a = x + W(y)$ for the physical row after the first shift. A lane with letter $c = W(y)$ originated in row $a - c$. It should end at row $\pi(a - c)$. Choose the target letter

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

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

Compute a subsequence matching of $\phi_a W$ and W of length at least $(1 - \varepsilon)M$. Send 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 $\phi_a W$. Each matched pair uses one occurrence of the same

color on each side. We can therefore pair the remaining positions color by color, in occurrence order.

To implement f_a on tapes, separate the matched and unmatched source lanes by a sequential mask scan. Keep the matched stream in order. Stable binary radix passes group the unmatched lanes by their required k-bit colors. For the vacant target positions, compute the masks of the stable radix passes that would group these positions by color. Applying the passes in reverse to the grouped source stream places the unmatched lanes in the order of the vacant positions. A stable binary split and its inverse each take a constant number of passes. The inverse splits after the known size of the zero part and merges in the order given by the original mask. The masks and counts depend solely on W and π. Finally interleave the two streams according to the target mask. The full-row scans cost $O(\mathcal{V})$ over all rows. The radix passes, including moves and rewinds, act on at most $\varepsilon\mathcal{V}$ bits, costing $O(\varepsilon\mathcal{V}k)$. Since $\varepsilon k \leqslant 1$, these passes take $O(\mathcal{V})$ time. Rows and work streams are reused serially.

Undo the first shift. The output row is $a - W(f_a(y)) = \pi(a - W(y)) = \pi(x)$. All three stages are bijections and preserve v, z at their endpoints. Thus the M lanes of each input row x reach the M positions of $\pi(x)$ bijectively, preserving the bit order within each lane.

Constructing the matching masks, radix schedules and rotation offsets takes time polynomial in qM. To locate segment boundaries, the machine may use binary lengths and reusable tape templates for a part or lane. Each template scan is charged to the bits copied. Across all fibers, the number of lengths and resets is polynomial in qM, and each uses arithmetic polynomial in $\log(2 + qMb)$. For a fixed constant C their total cost is at most $(qM)^C(\log(2 + qMb))^C$. Choose $D \geqslant 2C$. Under the width condition of the lemma, $(qM)^C \leqslant (qMb)^{1/2}$ and $(\log(2 + qMb))^C = O\left((qMb)^{1/2}\right)$. The length computations therefore cost $O(qMb)$ steps. Together with (1) and the sequential passes this gives the time and space bounds. □

One intermediate row. Work modulo 4 with $W = (0, 1, 2, 3, 2, 0, 3, 1)$ and $(\pi(0), \pi(1), \pi(2), \pi(3)) = (2, 0, 3, 1)$. In intermediate row 1, the required color list $\phi_1 W$ is $(1, 3, 0, 2, 0, 1, 2, 3)$. Source positions $(0, 1, 3, 4, 7)$ match target positions $(1, 3, 4, 5, 6)$ in that order. The three remaining source positions $(2, 5, 6)$ require colors $(0, 1, 2)$ and go to the three vacant target positions $(0, 7, 2)$, respectively. For instance, source position 2 in this intermediate row came from input row $1 - W(2) = 3 \pmod 4$. Its target lane has color $W(0) = 0$, so undoing the shift sends it to output row $1 - 0 = 1 = \pi(3)$.

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 through every local permutation. The color $W(y)$ is determined by the lane's current position y. Between local permutations we move each encoded item as a whole. An external header holds its flags and current radix digit. We permute the q headers by the same π as their items, using scans of the headers in time polynomial in q and the digit width.

To read a t-bit digit, compute which original data pieces contain its bits. Once the data-piece width is at least t, the digit meets at most two pieces. A single scan of the lanes identifies these pieces by their original lane numbers and recovers the digit in its logical order. 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 take $O(\beta)$ time. When the last digit has been processed, or earlier if the bucket is small enough for mergesort, we restore each item's pieces to their original order by fixed-tape mergesort on the lane numbers and remove the labels and padding. This restoration costs $O(\beta \log M)$ time once per item.

4 Radix Sorting

In this section we work with the fixed-width binary keys of Theorem 1.2. We process the augmented key from the most significant end. A radix digit is a block of consecutive bits of this augmented key. At a given level, a *bucket* consists of all items with a common prefix of processed digits. Its children are the buckets obtained by extending this prefix by the next digit. As in the high-order recursion of Li, Li and Liu [LLL26, Section 8.2], we form the children consecutively in digit order.

4.1 Distributing One Digit

Fix a radix $K = 2^t$, $t \geqslant 2$, and let $L = K^2$. Consider one current 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 $m \leqslant LR < 2m$. Pad the bucket with dummies and divide it into L consecutive blocks of R items each. From the next digits we choose a cyclic shift for each block and apply it to the complete items. Row $r \in [R]$ consists of the L items at position r in the shifted blocks. We bring those items together on the tape.

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. In every row, reserve s_c slots of K item positions 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.

Compute the m_c by copying the t-bit digits and binary radix sorting the compact copy in $O(mt^2)$ time. Under the width choice in Section 4.3, $t^2 = O(\beta)$, so the sort and digit extraction together take $O(m\beta)$ time. Binary counters can be incremented with amortized carry cost. Since $\sum_c \mu_c \leqslant L$, $\sum_c s_c \leqslant 4K$. The s_c and the weights below use integer arithmetic on m_c, R, K. Their required quotients lie in $[0, O(K)]$ and can be found by binary search and multiplication by short integers.

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 takes $O(mKt^{O(1)})$ bit steps on fixed tapes. Rotating the items and laying them out in contiguous rows takes $O(m\beta t^\tau)$ additional time.*

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

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

Consider 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 preserve this upper bound as we choose the shifts.

Fix the block shifts one 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 $c \in [K]$. Write $B(r+\ell, c)$ for the current contribution if we choose shift $\ell \in \mathbb{Z}/R$. Up to the common factor two, the only term of the conditional expected potential depending on ℓ is

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

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 $0 \leqslant j \leqslant \nu$, define residue histograms

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

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 taken modulo 2^j. Choose the shift from low bit to high bit. At step $j = 1, \ldots, \nu$, evaluate the unscaled sum for the two extensions of the residue chosen at step $j-1$ and take the smaller. The mean of the two scaled scores is the preceding score, so the completed shift meets the desired inequality. Update S by the chosen rotation before processing the next block.

Store each residue table in (u,c) order. To form B_ν, scan the current block and emit its K zero-one entries per position by enumerating the digit classes. For each residue, read the K weights d_c in order and rewind their tape. These passes take $O\left(RKt^{O(1)}\right)$ steps per block. Starting with level ν, add the corresponding entries of its two contiguous halves to build level $\nu-1$, then repeat. Save the levels on a stack of consecutive tape segments and read them back starting at level zero. Reversing a segment into its forward order uses two sequential copies. At level j, every count has $O(t+\nu-j)$ bits because each residue has $R/2^j$ positions and $S(r,c) \leqslant L$. To evaluate either child, split the B_j stream at the required whole group of K entries and cyclically rotate the two segments. Read the two streams in parallel, multiply corresponding counts using ordinary binary multiplication, and apply the power-of-two weights by shifts. Sum the nonnegative terms by adding consecutive pairs, then pairs of sums, until one remains. At the i-th reduction level the count of terms falls by 2^i and the width grows by at most i. Thus only the last additions involve the full width of the sum. Separators and amortized binary countdowns locate the rotation cut in a scan.

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 takes a fixed polynomial in ν, t per level, within this bound since $\nu^{O(1)} = O(2^\nu)$. One scan builds the next full table S_ν from the old S_ν and the rotated current block. By the entry widths above, the histograms occupy $O\left(K \sum_{j=0}^{\nu} 2^j (t+\nu-j)\right) = O(RKt)$ bits, reused for the next block. Summing over the L blocks gives the claimed time since $LR < 2m$.

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, rotate the items in each block by splitting at the chosen offset and concatenating the two segments. This takes $O(m\beta)$ time. Write a position within a block as $uL + s$. Since $L \mid R$, the tape has layout (a, u, s) with the block index a and s both in $[L]$. Apply the address-field interchange (1) to a and s to obtain (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

Use $4K$ slots per row, each holding K complete items. Assign successive slot intervals to classes $0, 1, \ldots, K-1$, using s_c consecutive slots for class c. The surplus slots contain dummies. All R rows of the bucket use the same slot intervals.

Process one row at a time. Append dummies until it has

$$q = 8L = 8K^2$$

input cells. Define a permutation of these cells as follows. Among the occurrences of each digit class $c \in [K]$, send the first $\min(X(r,c), T_c)$ to that class's slots in the first $4L$ output cells. Assign the excess input items to a separate L-cell suffix of the output. Assign the unused dummy sources to all other targets in increasing order. The row holds at most L input items, so a bijection extends these assignments. The permutation depends on at most L short digit labels, the row order, dummy flags, and the small table (s_c). These determine the permutation in time polynomial in q.

For this application of Lemma 3.3, the q items in one padded row are the q rows of the local permutation. Set aside their short external headers, apply the lemma to the item encodings, and reattach the headers in the same permuted order. Each genuine item routed to the first $4L$ output cells occupies a slot for its digit, and each of its pieces still carries its original index. Copy these $4L$ cells, including dummies, to the main tape. Copy the excess items to the overflow tape. The main tape has room for $4RL = O(m)$ items.

We must now bring together corresponding slots from different rows. Both K and R are powers of two, and $R \geqslant K^2 \geqslant 4K$, so $4K \mid R$. Number the rows in their current tape order and write this ordinal as $a \cdot \frac{R}{4K} + u$ with $a \in [4K]$. Every row uses the same slot intervals, so this numbering preserves the destination class of each slot. Write s for the slot number. The padded main tape has layout (a, u, s, z), where z ranges over the contents of one whole slot. Apply the address-field interchange (1) to a and s. In (s, u, a, z) order all the slots for any one class are contiguous. This costs $O(m\beta t^\tau)$. The two equal fields have width $t + 2$, independently of the size of the parent bucket.

Filter the main-stream dummies, retaining class boundaries, including empty classes. Sort the compact overflow tape by its current t-bit digit using ordinary stable binary radix splits that move all fields of each item together. The overflow passes take $O(m\beta t/K)$ time by (9). Including the scan of the main stream gives $O(m\beta)$ time, since $t \leqslant K$. Read both streams in class order and concatenate the main and overflow portions for each class. This last pass uses just a constant number of heads. After the two streams have been merged, suppress empty child markers. The nonempty children of the parent bucket are now in ascending digit order. Items within a child

can occur in any order. Processing the remaining digits orders all items by their augmented keys, which 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 sufficiently large n, let $u = \log n$ and take

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

Choose the constant threshold for n so that $t \geqslant 2$ and the width estimates below hold throughout the radix algorithm. Use ordinary mergesort below this threshold. Integer bit lengths can replace real logarithms in this rule with $O(1)$ changes in t. These choices can be made from the binary input dimensions on the fixed machine. The implicit constants may depend on the fixed width bounds in Theorem 1.2.

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 width of an item together with its copied key and $\lceil \log n \rceil$-bit input position. Thus $\widehat{\beta} \geqslant u$ on every input and $\widehat{\beta} = O(\beta)$ under the hypotheses of Theorem 1.2. The encoded item occupies one row of Lemma 3.3. Its width is at least $\widehat{\beta} \geqslant u \geqslant (qM)^D$ for every sufficiently large n, by (11). This gives a single threshold for the radix algorithm. The machine constructs the word, the matching masks, and the local permutations by sequential scans in $O\left(\widehat{\beta}\right) = O(\beta)$ time per row. The same storage is reused for successive rows. The balancing cost $Kt^{O(1)}$ per item is also $O(\beta)$.

The encoding in Section 3.2 adds $O(M \log M + qM + t)$ bits per augmented item for labels, padding and the external header. By (11) the resulting width is $O(\beta)$ and each data piece has at least t bits for large n. Reading a digit, including the $(qM)^{O(1)}(\log(2+\beta))^{O(1)}$ time for position calculations, therefore takes $O(\beta)$ steps per item. Standard delimiters or reusable tape templates locate fixed-width items in a scan. Computing a per-bucket count uses its local bit length. A fixed polynomial in $\log(2m)$ incurred once per bucket is $O(m)$ up to an absolute constant.

Apply Sections 4.1 and 4.2 in most-significant-digit order. The last digit is padded on its low-order end with zeroes if needed. If $m < L^2$, restore the original piece order in each item. Sort the bucket by the remaining bits of the augmented key with ordinary tape mergesort in $O(m\beta t)$ time, since $\log m < 4t$. Mark the bucket completed and copy its contents in order at each remaining level. After the final digit restore the original piece order in every remaining bucket. Each item undergoes one restoration and at most one terminal mergesort. The original input position in the augmented key ensures stability.

There are $O(u/t)$ levels. The sum of active parent sizes at a level is at most n. Lemma 3.3 takes $O(t^\tau)$ time per bit, as do the two global interchanges within a parent. The digit levels therefore take $O(Vu/t^\eta)$ time. The terminal sorts cost $O(Vt)$ and restoring the piece order costs $O(V \log M)$. Both are $o(Vu/t^\eta)$, since $t = \Theta\left(\log^{(3)} n\right)$ and $\log M = o(\log u)$. Thus the whole run takes

$$O\left(\frac{Vu}{(\log^{(3)} n)^\eta}\right)$$

time.

Store the old and new level on a constant number of tapes. Copy the active parent into a reusable local work area, leaving the outer read and write heads at its level boundaries. The temporary heads rewind within the current parent's work area. Within this area one parent uses the two address-field interchanges, temporary rows, histograms, and the main and overflow streams serially. The padded items occupy $O(m\beta)$ bits and the histograms of Section 4.1 occupy $O(RKt) = O(m\beta)$ bits. Constructing the word and its matchings uses $u^{o(1)}$ bits. Previously visited tape regions can be rewound and reused, with each rewind charged to the scan of that region. The number of tapes and the peak occupied space are respectively constant and $O(V)$. □

For items of width $\Theta(\log n)$, the layout also gives a simple bound on the digit width. Each item occupies at least $qM \geqslant q^2$ bits, since the balanced word contains all q letters. By (10), $q = 8 \cdot 2^{2t}$, so linear space entails $t = O\left(\log^{(2)} n\right)$ for this layout.

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, pp. 138–139]. For $n \geqslant 2$ strings of width $\beta \geqslant 1$, attach each $\lceil \log n \rceil$-bit destination as the key of its string, apply Theorem 1.2, and discard the keys in a scan. The strings then appear intact in destination order. For the total input length $N = \Theta(n(\beta + \lceil \log n \rceil))$, this takes $O\left(N \log n / (\log^{(3)} n)^n\right)$ time and $O(N)$ space. If $\beta = \Omega(\log n)$, the strings alone occupy $\Theta(N)$ bits.

Replacing each key by its rank extends Theorem 1.2 to key fields of arbitrary width.

Corollary 5.1. *The bounds of Theorem 1.2 hold for items of common width $\beta = \Omega(\log n)$ whose designated key field has any width $0 \leqslant \ell \leqslant \beta$.*

Proof. For $n = 1$ a scan suffices. Let $h = \lceil \log n \rceil$ and $n' = 2^h$. For $\ell \leqslant h$, apply Theorem 1.2. For $\ell > h$ we first compute an h-bit rank for each key. Keep the original items on a separate tape. Form a table with one row per key, adding dummy zero keys to obtain n' rows. Split each key from the most significant end into h-bit chunks. Pad the last chunk on the right with zeros, then append zero chunks to make the number of columns a power of two. This table has $O(n\ell)$ bits.

We rank successive pairs of blocks by doubling their length, as in the naming method of Karp, Miller and Rosenberg [KMR72, Section II]. Initially a chunk is its own label. Pair adjacent labels in each row. Within each column of pairs, attach the position in the physical column to each pair and sort the pairs by their two labels with Theorem 1.2. Assign consecutive ranks in $[n']$ to distinct pairs, testing equality on the two labels alone. Sort by the attached positions to restore the column order. A new rank orders the concatenation of the two blocks among the rows at that column. Iterating gives one rank per row, with equal ranks for equal original keys.

For the column sorts, let D' be the number of pairs in a row. If $D' \geqslant n'$, write an address as (i, g, j) with $i, j \in [n']$ and (g, j) the pair position. Interchange i and j to obtain (j, g, i). Each pair position now occupies a contiguous column. If $1 < D' < n'$, write the row index as (a, b) with $a \in [D']$ and apply $(a, b, j) \mapsto (j, b, a)$. Here the physical row order within a column is (b, a). If $D' = 1$, that column is already contiguous. In each case its local positions fit in h bits. Undo the interchange after the two sorts. The next round again starts in row-major order with adjacent blocks in their original order.

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. By Theorem 1.2, the two sorts in each column take $O\left(h/(\log^{(3)} n)^{\eta}\right)$ steps per bit. Each interchange exchanges fields of at most h bits and takes $O(h^{\tau})$ steps per bit by Section 2.3. Thus rank computation takes $O\left(n\ell h/(\log^{(3)} n)^{\eta}\right)$ time and $O(n\ell)$ space. Discard the dummy ranks, attach each remaining rank to its original item, and apply Theorem 1.2 once more. The added rank has width $h = O(\beta)$. Stability of this last sort preserves the input order at each rank. □

For items of small width, we use binary radix sorting. At intermediate widths, we sort consecutive batches and combine their runs of equal keys.

Corollary 5.2. *Let $n \geqslant 2$ items have common positive width β and store their keys in the same contiguous field of ℓ bits, where $0 \leqslant \ell \leqslant \beta$. With $V = n\beta$, one deterministic fixed-tape machine stably sorts the items in $O\left(V \log n/(\log^{(3)} n)^{\eta}\right)$ time and $O(V)$ space, uniformly in the two widths.*

Proof. Write $h = \lceil \log n \rceil$. 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 takes at most β linear-volume passes. Its time $O(V\beta)$ fits the bound. It remains to consider $2 \log h < \beta < h/4$.

Split the input into consecutive batches of $r = 2^{2\beta}$ items. Pad only the last batch to r with flagged dummies. Since $r < 2^{h/2} \leqslant \sqrt{2n}$, the padded batches contain $O(n)$ items altogether. Stably sort each batch by its key with Theorem 1.2 and discard the dummies. The item width with the padding flag is $\Theta(\beta) = \Theta(\log r)$. The time over all batches is $O\left(V \log r/(\log^{(3)} r)^{\eta}\right)$. This fits the asserted bound since $\log r \leqslant h$ and $x/(\log^{(2)} x)^{\eta}$ is eventually increasing.

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 $2^{\beta} > h^2$. When writing the batches consecutively, mark the start of each nonempty batch and each change of key within it. Count the marked positions using amortized constant-time binary increments.

Set $p = \lceil n/s \rceil$. In a second scan split each run into consecutive pieces of p items, padding its last piece. The number s' of pieces satisfies $s \leqslant s' \leqslant s + n/p \leqslant 2s$. Give each piece a header with its key and its 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 $O(V)$. Computing p by binary division takes $h^{O(1)}$ steps. Initializing a counter for each run or piece takes $O(sh)$ time in total. Copying the items and decrementing the counters takes time linear in the volume scanned.

Apply Theorem 1.2 to the pieces, sorting by key alone. Since $\log s' = \Theta(\log n)$ and $s' \leqslant 2n$, this takes the claimed time and space. In a final scan output 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. □

To prove Theorem 1.1, we group the items by ranges of key lengths and use Corollary 5.2 to sort equal-width pieces within each group. We then reassemble the items and merge the sorted lists.

Proof. For $n = 1$ a scan suffices. Compute N, n by a scan, and set $T = \lceil N/n \rceil$. Let ℓ_i be the length of key i. Assign its pair to the least integer bin $j \geqslant 0$ such that $1 + \ell_i \leqslant H_j$, where $H_j = 2^j T$. 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 $O(N)$.

Distribute the pairs stably by bin index using binary radix sorting. Store the index with its least significant bit first, using $O\left(\log^{(2)}(2 + \ell_i)\right)$ bits for pair i. Keep one copy to distinguish the bins and remove one low bit from a working copy at each pass, treating an exhausted copy as zero. The tags are produced from sequential length counts and joined to the input by scans. Computing a tag takes time polynomial in $\log(2 + \ell_i)$ and $\log(2 + T)$, hence $O(1 + \ell_i + T)$ per pair and $O(N)$ overall. Each radix pass costs $O(N)$ time. There are $O\left(\log^{(2)} n\right)$ passes.

In bin j, encode each zero bit of a key as 01 and each one bit as 10. Append the terminator 00, then pad with zero pairs to a common width $\lambda_j = 2H_j$. A first differing bit has the same order after encoding, and a terminator precedes either digit. Thus sorting the encoded keys compares the original keys lexicographically, with a proper prefix first. 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. The original symbols use a constant number of bits apiece, and the capacity bound gives $\sum_j N_j = O(N)$. A scan counts pairs and measures the original volume in each bin. Add $n_j \lambda_j$ once per bin to obtain N_j. Arithmetic on the resulting bin dimensions costs a fixed polynomial in $\log(2N)$ overall, absorbed by $O(N)$.

For each nonempty bin choose $p_j = \lceil N_j / n_j \rceil \geqslant \lambda_j$. Split the original encoding of each pair into consecutive p_j-bit pieces, padding its last piece. Give every piece the comparison key of its pair 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 $O\left(\log(2 + p_j)\right)$ header. A separate tape holds a copy of the current comparison key for writing and rewinding at each piece. These scans and the initialization of the length counters cost $O(p_j)$ per piece. Since $n_j p_j = O(N_j)$, the expanded bin has volume $O(N_j)$ and is constructed in the same time.

For $s_j \geqslant 2$, apply Corollary 5.2 to the s_j pieces with the comparison key as the designated field. Copy a singleton stream directly. Stability preserves the relative order of all pieces with a given key. The pieces of each original pair are therefore adjacent and in their original order. A scan concatenates their valid prefixes, which together contain the original delimiters. Since $s_j \leqslant 2n_j \leqslant 2n$ and $\log x/(\log^{(3)} x)^\eta$ is eventually increasing, the sum of the sorting times over the bins is $O\left(N \log n/(\log^{(3)} n)^\eta\right)$. Processing the bins successively with reusable work tapes takes $O(N)$ space.

Merge the sorted bins in balanced rounds on a fixed number of tapes. To compare the first keys of two lists, read up to the first mismatch or terminator and rewind the inspected prefixes. Copy the pair with the smaller key, choosing the left pair on a tie. The number of symbols inspected in either key is at most the length of the shorter key plus one. The comparison and the rewinds can thus be charged to the pair copied, even when the other list retains its first key through several comparisons. Drain an unmatched list by a scan. Each round takes $O(N)$ time and there are $O\left(\log^{(2)} n\right)$ rounds. The bin distribution and these merges fit the sorting bound since $(\log^{(2)} n)(\log^{(3)} n)^\eta = o(\log n)$. Equal keys belong to the same length bin, where their input

order was preserved. This proves stability and the bound in terms of n. The alternative bound follows from $n \leqslant N$ and the same monotonicity. □

Sorting by numerical value. Consider signed numerals given by finite expansions in a fixed base $b \geqslant 2$. For each magnitude let I, F be its integral and fractional digit strings after removing leading zeros from I and trailing zeros from F. Either may be empty. Encode a digit d by the $s = \lceil \log(b+1) \rceil$ bits of $d+1$, most significant first, and denote the resulting encoding of a digit string by C. Use the magnitude key

$$M = 1^{|I|}0\, C(I)\, C(F)\, 0^s.$$

The unary prefix compares integral digit counts. At equal counts, the first differing digit orders the magnitudes. If one fractional string is a proper prefix of the other, the longer string has a nonzero digit beyond the prefix. The terminator 0^s orders the shorter string first and makes the magnitude keys prefix-free. Complementing all bits then reverses their order.

Use $1M$ for nonnegative values, including either sign of zero, and $0\overline{M}$ for negative values, where the bar denotes bitwise complementation. Equal values have equal keys. Carry the original numeral and its associated data alongside this comparison key. A constant number of scans per numeral suffices to trim zeros, write the unary prefix and encode the digits. Since b is fixed, this takes $O(N)$ tape steps and produces total length $O(N)$. Applying Theorem 1.1 therefore sorts the original numerals stably by value within its time and space bounds.

6 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 6.1. *Fix $\alpha > 0$. Let Q be a power of two and let $B \geqslant 2^{Q^\alpha}$. Given the destinations of Q strings of B bits each in source order, with each destination occurring once, a uniform deterministic multitape machine writes the strings in destination order in $O_\alpha(QB(\log Q)^\tau)$ time and $O(QB)$ space, including the time to compute the routing instructions. For $B \geqslant Q^2$ the time bound is $O(QB(\log Q)^\tau) + C(Q)$ in the same space, for a computable function C. Here $\tau = 1 - \eta$ is the exponent in the $O(\mathcal{V}u^\tau)$ time bound for interchanging two u-bit address fields in an array of $\mathcal{V}$ bits (Section 2.3).*

Proof. Choose positive integers a_0, b_0 with $\gamma = a_0/b_0 < \min\{\alpha, 1\}$. We prove the time bound for $\lfloor \log B \rfloor \geqslant \lceil Q^\gamma \rceil$, which holds under the stated width hypothesis for all sufficiently large Q. The machine can test this condition by comparing $(\lfloor \log B \rfloor)^{b_0}$ with Q^{a_0}. For bounded Q we may attach the destinations and sort the strings by ordinary tape mergesort. A single string needs only a scan.

Fix an integer d, to be chosen below. Write $h = \log Q$, $s = \lfloor h/d \rfloor$, and $q = 2^s$. We may assume $s \geqslant 2$ by increasing the range of the preceding base case. Split the address of a string 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. For each outer pair of layers, group the sources and targets by those other fields. The strings form the edges of a regular bipartite multigraph between the two sets of groups. Decompose it into perfect matchings and assign one field value to each matching. The outer layers route each matching through its assigned value, leaving a permutation of the remaining fields within that matching. Recursing gives the switch tables. Augmenting paths and sequential table scans find the matchings in time polynomial in Q for fixed d.

Take the word W of length M on q letters from Lemma 3.2 with tolerance $1/s$. It shares a subsequence of length at least $(1-1/s)M$ with each relabeling. The same lemma gives one occurrence of each letter in every successive block of q positions. Write $y = (p, e_0)$ for the position at offset $e_0 \in [q]$ in block $p \in [M/q]$. Let σ_p be the permutation for which $W(y) = \sigma_p(e_0)$.

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

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

For a layer whose switch acts on $e \leqslant s$ address bits, interchange those bits with the low e bits of ν_f. The field in the former ν_f position now consists of an unchanged prefix followed by the switch address. Denote this full field by a_f. The other original string-address fields determine the switch. Use the same switch table for each value of the displaced low e bits of ν_f. The permutation of $[q]$ preserves the prefix of a_f and applies the switch to its low e bits. Interchange a_f and ν_0, bringing the fields into the order $(\nu_0, a_f, y, \nu_1, z)$. For every ν_0 apply Lemma 3.3 to (a_f, y, ν_1, z), using the same common-subsequence matching and the same order for the unmatched pieces. Then reverse the two address-field interchanges. The string address has undergone the required switch, and ν_f, ν_0, ν_1, z have their initial values. For a fixed input string and ν_f, both the permutation of the piece positions y and the choice of matched pieces are independent of ν_0, ν_1, z. The same statement holds for a layer with $e < s$, since ν_0 is separate from ν_f. Apart from computing the instructions, each layer costs $O\left(Q\widehat{B}s^\tau\right)$ steps.

We next restore the order of y. Fix one output string and one value of ν_f. If a switch sends the string from local address x to x', a piece of color c enters intermediate row $x + c$ and leaves at color $\phi_{x+c}(c) = c + x - x'$ modulo q, by (3). Consequently all its pieces have the same total color shift after the H layers. Associate to each piece the H bits indicating whether it was matched or unmatched at each layer. Among pieces of the same initial color, a matching preserves the order of the matched pieces, and the pairing by occurrence order preserves the order of the unmatched pieces. Pieces with the same initial color and the same sequence of bits thus retain their original relative order.

For each ν_0, p, permute the q consecutive offsets by $e_0 \mapsto c = \sigma_p(e_0)$. Each offset holds qb bits indexed by (ν_1, z), so this is a recursive call on q strings of qb bits each. Interchange ν_0 and c across p by (1), giving 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.

For each color, stably group these strings 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. Merge the at most 2^H runs by their initial block indices using two-stream merges. Undo the color shift, reverse the interchange of c and ν_0, and, for each ν_0, p, recursively apply $c \mapsto \sigma_p^{-1}(c)$ to the q strings indexed by c. The destination string now has its original bit order. The grouping bits, initial block indices and merge choices can be computed from the permutation by tracing one representative of each piece position. Simulating the groupings and merges gives the choice of input stream for each output position. For fixed d, these computations take time polynomial in Q, q, M. Using the resulting instructions, grouping and merging the strings takes $O\left(HQ\widehat{B}\right)$ tape steps.

Choose $d > 4/\gamma$. By Lemma 3.2, $\log M = O\left(2^{2s}s^2\right)$, so 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, as the lemma requires. Also

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

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.

To implement the recursion, represent each work tape by two stacks and the current symbol. On entry to a recursive call, save the current symbols and control state and place a marker on each stack. The child runs the same tape routine directly above these markers. A reusable tape transfers its input and output in linear time. On return, rewind each stack to its marker. Each frame uses $O_d(QB)$ bits for its own buffers, so its final rewinds also cost $O_d(QB)$ steps. Calls run successively, and a child has input volume $q^2 b$ out of the parent's padded volume Qq^3Mb. Its volume is at most $2/(QqM)$ of the parent's original volume. Summing the buffers of the active frames gives $O(QB)$ working space.

Let $F(h)$ bound the steps per input bit of this implementation, including the return from the call. For the calls applying σ_p, each of the Q strings has q^2 choices of (ν_f, ν_0) and M/q choices of p. Each call processes $q^2 b$ bits, giving $QqM \cdot q^2 b = Q\widehat{B}$ bits in total. The calls applying σ_p^{-1} process the same volume. Each child's steps are counted once, and the two groups together process at most $4QB$ bits, independently of d. The $H = O(d)$ switch layers use Lemma 3.3 directly. Including the linear transfers between frames and the removal of padding on output, we obtain

$$F(h) \leqslant 4F(\lfloor h/d \rfloor) + O_d(h^\tau).$$

Choose d also large enough that $d^\tau > 4$. Induction gives $F(h) = O_d(h^\tau)$. For $B \geqslant Q^2$ below the tested width threshold, sort on the destination keys by ordinary tape mergesort in $O(QB)$ space and $O(Q(B + \log Q)\log Q)$ time. 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 6.1 to the items in each padded row lets the radix sorter use wider digits as the item width grows.

Corollary 6.2. *Let $n \geqslant 2$ items have common width $\beta = \Omega(\log n)$, and let their designated binary keys have any common width at most β. For $u = \log n$ and $V = n\beta$, stable deterministic sorting takes*

$$O\left(\frac{Vu}{\min\left\{u, \log^{(2)} \beta\right\}^\eta}\right) \quad \textit{time and} \quad O(V) \quad \textit{space.}$$

For $\beta \geqslant n^a$ with fixed $a > 0$, the time is $O_a\left(V \log n/(\log^{(2)} n)^\eta\right)$, also for routing by destination keys. For $\beta \geqslant 2^{n^\alpha}$ with fixed $\alpha > 0$, the running time per input bit is $O_\alpha\left((\log n)^{1-\eta}\right)$.

Proof. First suppose the keys have $O(u)$ bits. Take the digit width $t = \lfloor \frac{1}{16} \min\left\{u, \log^{(2)} \beta\right\} \rfloor$ and $K = 2^t$, using ordinary mergesort for the bounded initial range of n as in Section 4.3. Use the digit distribution of Sections 4.1 and 4.2 on whole items with their stability tie breakers. In a padded row of $q = 8K^2$ items, the digit labels determine the row permutation. For sufficiently large n the choice of t gives $q \leqslant \log \beta$. Thus Theorem 6.1 with $\alpha = 1$ applies to that row, with time $O(q\beta t^\tau)$ including the routing instructions. Each item arrives with its bits in order, so a scan reads its next

digit. Computing the block shifts costs $Kt^{O(1)} = O(\beta)$ per item. The $O(u/t)$ digit levels together take $O(Vu/t^{\eta})$ steps.

A terminal bucket has $m < 2^{4t}$ items. Pad it to the next power of two, using at most $2m$ positions. To determine destinations, sort the $O(u)$-bit keys with their original input positions and their positions in the bucket, assign destinations in sorted order, and sort the destinations back by local position. These two tape mergesorts take $O(mut)$ time. For sufficiently large n, the padded size is at most $2^{4t} \leqslant \log \beta$, so Theorem 6.1 performs the permutation in $O(m\beta t^{\tau})$ steps. Both costs fit $O(m\beta u/t^{\eta})$ since $t \leqslant u$ and $t^{1+\eta} = O(\beta)$. Each item reaches at most one terminal bucket. The original input position breaks ties, and the sorting and routing procedures use linear space.

For longer keys with $\beta \leqslant u^4$, Corollary 5.1 gives the stated bound since $\log^{(2)} \beta = \log^{(3)} n + O(1)$ in this range. Now suppose $\beta > u^4$. Round n up to $n' = 2^h$ with $h = \lceil u \rceil$ and add dummy zero keys. Let D be the least power of two at least h^2, and let $b = \lceil \beta/D \rceil$. Right-pad a copy of each key with zeros to Db bits and split it into D successive b-bit chunks. Keep the original items on another tape. For sufficiently large n, $D \leqslant n'$. Interchange the high $\log D$ bits of the row address with the chunk index. The columns now occur in order of significance. The row order within each column differs from the original order by a fixed cyclic permutation of the address bits. The interchange costs $O(V(\log D)^{\tau})$, within the stated bound.

Read the columns from most to least significant. Give each row the rank of its prefix among the distinct prefixes processed so far. Initially all rows have the same rank. Use the position in the physical column as the row identifier. This order is common to all columns. Attach the old rank and the row identifier to each current chunk. Group the chunks by rank using the short-key algorithm just proved. These items have width $\Theta(b)$ and keys of $O(h)$ bits. Here $b = \Omega(\sqrt{\beta})$ and $b = O(\beta)$, so $\log^{(2)} b = \log^{(2)} \beta + O(1)$. Since the combined volume of the columns is $O(V)$, these groupings take the claimed time.

Refine each equal-prefix class by its current chunk. Use the majority vote algorithm of Boyer and Moore [BM91] and verify the candidate's frequency by a second scan. Comparing chunks and rewinding the candidate take time linear in the volume of the class. If the class splits and has a strict majority, sort the other chunks and one representative of the majority by ordinary tape mergesort. Save the row identifiers for the majority and give them the representative's new rank. If the class has no strict majority, sort all its chunks. An unsplit class remains one subgroup. Process the old classes in rank order, assigning consecutive ranks to the distinct subgroups in order of chunk value. Before the next column, sort the new ranks by row identifier. On the last column, undo the cyclic permutation of the address bits in each identifier and sort the ranks directly into original input order.

In a split class, call a row 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 D 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 take $O(n'bh^2) = O(V)$ steps in total, since $D = \Theta(h^2)$. Sorting the ranks by position over all columns costs $O(n'Dh^2) = O(V)$ because $b = \Omega(h^2)$. The remaining scans take $O(V)$ time. Discard the dummy rows, attach the ranks to the original items, and apply the stable short-key algorithm. □

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* $0 < \delta < 1$*?*

At this width, tape mergesort takes $O\left(n(\log n)^2\right)$ steps. Theorem 1.2 saves a factor of $(\log^{(3)} n)^\eta$ 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 N bits.

Question 7.2 (Routing to given destinations)**.** *Each of* Q *strings of* B *bits is given with its position in the output. Can we write the strings in that order in* $O\left((\log Q)^{1-\eta}\right)$ *tape steps per bit whenever* $B \geqslant Q^a$*, for some fixed* $a > 0$*?*

Here Q is a power of two, and the destinations form a permutation listed in input order. The time includes computing the moves. Theorem 6.1 achieves this cost per bit for $B \geqslant 2^{Q^\alpha}$ with fixed $\alpha > 0$.

A positive answer to Question 7.2 would improve the bound for logarithmic-width items by replacing $\log^{(3)} n$ in its denominator with $\log^{(2)} n$, 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 take $t = \lfloor c \log^{(2)} n \rfloor$ for a sufficiently small constant $c > 0$. At this choice of t, the items meet the width threshold and the shifts and row permutations take $O(\log n)$ steps per item to compute. Summing the $O\left(t^{1-\eta}\right)$ steps per bit for digit partitioning over $O(\log n/t)$ levels gives the claimed saving, 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 tape complexity analysis and with drafting and revising 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. 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. 5

[Ben64] V. E. Beneš. Permutation groups, complexes, and rearrangeable connecting networks. *The Bell System Technical Journal*, 43(4):1619–1640, 1964. 18

[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. 21

[BP25] Andrew Best and Yuval Peres. Edit distance in substitution systems. arXiv:2501.00440v3, 2025. 9

[CLRS22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, fourth edition, 2022. 3

[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. 4

[FHLS19] Alireza Farhadi, MohammadTaghi Hajiaghayi, Kasper Green Larsen, and Elaine Shi. Lower bounds for external memory integer sorting via network coding. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 997–1008. ACM, 2019. 5

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

[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. 3

[Han04] Yijie Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *Journal of Algorithms*, 50(1):96–105, 2004. 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. 3, 5, 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 Computer Science (FOCS)*, pages 135–144. IEEE Computer Society, 2002. 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. Full version available as arXiv:2503.22848. 4

[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. 15

[Lei85] F. Thomson Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34(4):344–354, 1985. 5

[LLL26] Baochun Li, Zongpeng Li, and Sirui Liu. Undirected multiple-unicast network coding and faster external-memory integer sorting. arXiv:2610.20079v3, 2026. 5, 11, 25

[Ope26] OpenAI. Integer multiplication below $n \log n$, September 2026. Preprint, September 23 manuscript. 4, 7, 25

[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. 3, 4

[PK85] Martti Penttonen and Jyrki Katajainen. Notes on the complexity of sorting in abstract machines. *BIT*, 25(4):611–622, 1985. 3

[Sch78] Claus-Peter Schnorr. Satisfiability is quasilinear complete in NQL. *Journal of the ACM*, 25(1):136–145, 1978. 3, 15

[Sto73] Hanns-Jörg Stoß. Rangierkomplexität von Permutationen. *Acta Informatica*, 2(1):80–96, 1973. 3, 7

[SW23] András Z. Salamon and Michael Wehar. Effective guessing has unlikely consequences. *Theory of Computing Systems*, 67:548–568, 2023. 4

A A Slight Improvement in η

The interchange bound of [Ope26, Eq. (10) and Lemma 12] gives $\eta = 2^{-50}$. For the circuit of Li, Li and Liu, let m_0 be its dimension, W_0 its number of inputs, and S_0 the sum of its edge ranks. Their interchange recurrence gives the sorting bounds for any rational η satisfying

$$0 < \eta < 1 - \frac{\log(S_0/W_0)}{\log m_0}.$$

In particular their fixed parameters admit $\eta = 2^{-17}$ [LLL26, Corollary 5.4]. Write $\tau = 1 - \eta$.

In the fixed prime radix p used for addresses, consider two fields of width m_0^j with prefix count divisible by W_0^j. On A bits the recurrence makes S_0 width-m_0^{j-1} interchanges, each on A/W_0 bits with prefix count divisible by W_0^{j-1}. All other stream operations take $O(A)$ elementary tape steps on a fixed number of tapes [LLL26, Lemma 6.2 and Appendix A.1]. In the depth-first schedule of [Ope26, Proposition 11], saving and restoring the streams for the fixed number of children also costs $O(A)$. For a root of volume A_0 and field width m_0^J, the arithmetic on its dimensions and offsets takes a fixed polynomial in $\log(2 + A_0)$ per call. Since $m_0^J = O(\log(2 + A_0))$ and W_0^J is a fixed power of m_0^J, this cost fits within the smallest call's volume A_0/W_0^J.

Write F_j for the worst time per bit at width m_0^j. Two single digits can be exchanged in a fixed number of scans. Hence

$$F_0 = O(1), \qquad F_j \leqslant (S_0/W_0)F_{j-1} + O(1), \qquad F_j = O\left(m_0^{j\tau}\right).$$

For a binary field of width u, extend its range to the least power $p^\nu \geqslant 2^u$ of p. Sequential scans insert and remove the padding in both fields. This increases volume by a factor smaller than p^2, and $\nu = \Theta(u)$. The padding and decomposition of [Ope26, Lemma 12] and [LLL26, Appendix C] increase volume by a constant factor. Moving $O(\log(2 + u))$ high digits and padding the prefix count give the required divisibility. At most $m_0 - 1$ pieces of each width $m_0^j = O(u)$ remain. Summing F_j over these pieces costs $O(u^\tau)$ per bit, including the scans for the high digits. Thus (1) takes $O(\mathcal{V}u^\tau)$ time and, by Section 2.3, $O(\mathcal{V})$ space.

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