

Lower Bounds for Sorting with Few Random Accesses to External Memory

Martin Grohe
Humboldt-Universität
Institut für Informatik
Unter den Linden 6
10099 Berlin, Germany

grohe@informatik.hu-berlin.de

Nicole Schweikardt
Humboldt-Universität
Institut für Informatik
Unter den Linden 6
10099 Berlin, Germany

schweika@informatik.hu-berlin.de

ABSTRACT

We consider a scenario where we want to query a large dataset that is stored in external memory and does not fit into main memory. The most constrained resources in such a situation are the size of the main memory and the number of random accesses to external memory. We note that sequentially streaming data from external memory through main memory is much less prohibitive.

We propose an abstract model of this scenario in which we restrict the size of the main memory and the number of random accesses to external memory, but do not restrict sequential reads. A distinguishing feature of our model is that it admits the usage of unlimited external memory for storing intermediate results, such as several hard disks that can be accessed in parallel. In practice, such auxiliary external memory can be crucial. For example, in a first sequential pass the data can be annotated, and in a second pass this annotation can be used to answer the query. Koch's [9] ARB system for answering XPath queries is based on such a strategy.

In this model, we prove lower bounds for sorting the input data. As opposed to related results for models without auxiliary external memory for intermediate results, we cannot rely on communication complexity to establish these lower bounds. Instead, we simulate our model by a non-uniform computation model for which we can establish the lower bounds by combinatorial means.

1. Introduction

The massive datasets that have to be processed in many applications are often far too large to fit completely into the computer's (internal) main memory and have to reside in external memory (such as disks). When querying such data, the most constrained resources are the size of the main memory and the number of random accesses to external memory. It is well-known that access to external memory is by orders of magnitude slower than access to main memory. However, there is an important distinction to be made between a

random access to data in a particular external memory location, which involves moving the head to that location, and sequentially streaming data off the disks.

While there is an extensive literature on external memory algorithms (see, for example, [13, 10]), much less is known on complexity theoretic lower bounds. In recent years, in the context of data stream applications some efforts have been made to study the limitations of answering queries in a setting where only one or few sequential passes over the data are admitted [11, 2, 8, 3, 12, 5, 1, 7]. They result in strong lower bounds for a number of natural problems and for answering queries in languages such as XPath. The model underlying all these lower bounds allows few sequential passes over the data (or just one sequential pass in the important special case of data streams) and apart from that no access to external memory. The query processing takes place entirely in main memory, which of course is of limited size. The model does not allow for intermediate results to be stored in *auxiliary external memory*. However, storing intermediate results in external memory can be a very powerful mechanism. For example, while the data are sequentially read in a first pass, an algorithm may annotate the data and write the annotation on a second disk. Then in a second pass, the algorithm may use the annotation to compute the answer to the query. Or an algorithm may copy the first half of the data onto a second disk and then read the copied first half of the data in parallel with the second half of the original data to compare the two halves and maybe compute some kind of join.

Our main result is a lower bound for the fundamental problem of sorting a sequence of numbers or strings in a model that admits auxiliary external memory.

Formally, our model, which is the natural extension of the model considered in [7] to the setting with auxiliary external memory, is based on standard multi-tape Turing machines. Our Turing machines have several tapes of unbounded size, among them the input and output tapes, which represent the external memory (for example, each of these tapes may represent a disk). In addition, the machine has several tapes of restricted size which represent the internal memory. We put no restriction on the running time. While the "external memory tapes" are not restricted in size, we do restrict the access to these tapes by putting an upper bound on the number of times the heads on these tapes can change the direction in which they move. This way we also restrict the random access to external memory, because each location on the tape can be reached by possibly changing the direction of a head and then moving it to the desired location. We also discuss an enhanced model which admits explicit random access to the external memory tapes and show how this enhanced model can be simulated by the original one. The as-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODS 2005 June 13-15, 2005, Baltimore, Maryland.

Copyright 2005 ACM 1-59593-062-0/05/06 ... \$5.00.

tute reader who wonders if it is realistic to assume that the external memory tapes can be read in *both* directions (which disks cannot so easily) be reminded that we prove *lower* bounds, so admitting tapes to be read in both directions only makes our results stronger. For functions $r(N)$ and $s(N)$ on the natural numbers and a number t , we call a Turing machine (r, s, t) -*bounded* if it has t external memory tapes and, on an input of size N , reverses the direction of the heads on the external memory tapes at most $r(N)$ times and uses total space at most $s(N)$ on the internal memory tapes. More precisely, our main result states, for any $t \in \mathbb{N}$, that an

$$(o(\log N), O(\sqrt[5]{N}/\log N), t)\text{-bounded}$$

machine cannot sort. Intuitively, this means that sorting is not possible with only $o(\log N)$ random accesses to external memory and internal memory of size $O(\sqrt[5]{N}/\log N)$ on a machine that may have an arbitrary number t of auxiliary disks for storing intermediate results. We also prove that an

$$(o(\log m), O(\sqrt[6]{m}), t)\text{-bounded}$$

machine cannot sort an input consisting of m numbers of size $m^{O(1)}$ (or equivalently, m strings of length $O(\log m)$). This bound is tight, because a straightforward adaption of the standard merge sort algorithm shows that m numbers of size $m^{O(1)}$ can be sorted by an $(O(\log m), O(\log m), 3)$ -bounded machine.

To the best of our knowledge, these are the first lower bound results for such a computation model. Let us remark that our model is not only stronger than the models that have been considered in the context of data streams (cited above), but also stronger than the related *bounded reversal Turing machines* that have been studied in classical complexity theory [14, 6], because these models restrict the number of head reversals on *all* tapes and do not admit internal memory tapes.

It may seem at first sight that our model is not far from the data stream models cited above, and that the lower bound techniques should easily generalise. Unfortunately, this is not the case. The proofs of all these results build on communication complexity. Essentially, they are variations of the following argument: Split the input in the middle and show that the problem to be solved requires a number of $b(N)$ bits to be communicated between the two halves of the input. In each pass of the data, at most $s(N)$ bits can be communicated, where $s(N)$ is the size of the internal memory. Thus if $s(N)$ times the number $r(N)$ of passes of the input is less than $b(N)$ the query cannot be answered.

We observe that such arguments utterly fail in our setting: For example, by copying the first half of the input onto the second external memory tape and then reading it in parallel with the second half of the input in a second pass over the second tape, we can communicate as many bits as we like from the first to the second half. So communication between different parts of the input is no longer a bottleneck in our model. However, copying segments of the input to different locations does not disturb the order of the input string too much, at least not if it only happens a bounded number of times (note that copying a segment and then re-reading it requires at least one reversal of a head direction). Intuitively, this is the reason that sorting a sequence of numbers is hard in our model.

Note that even though it seems “obvious” that we cannot sort with sufficiently restricted $r(N)$ and $s(N)$, proving this is not so easy, because the algorithms are not restricted to sorting in the “natural way” by shifting around the input numbers. For example, an algorithm might use the first pass over the input numbers to compute the bitwise sum of all input numbers modulo some prime

number. While it is not clear what the benefit of such an operation would be, it does spoil many naive “information theoretic” arguments we might be tempted to use for our lower bound proof, because we suddenly have generated a number that in a non-trivial way depends on all input numbers.

Our way to resolve such problems is that we simulate our Turing machines by an abstract non-uniform computation model, which we call *list machine*. In certain ways, these list machines are much more powerful than Turing machines, but they are restricted in one essential way: They are only allowed to write the original input numbers into their memory (albeit many copies of them). This prevents them to perform complicated arithmetic operations on the input and store the results for later usage. We can then establish lower bounds for sorting on list machines by combinatorial means.

The paper is organised as follows. We introduce our model in Section 2. In Section 3, we formally state the main results. In Section 4, we introduce list machines. In Section 5, we state and prove the “Simulation Lemma”, which shows how Turing machines can be simulated by list machines. In Section 6 we prove the lower bound for sorting on list machines. It is then fairly easy to prove the main result; we give the proof in Section 7.

2. Our Model

As our basic model of computation, we use standard multi-tape Turing machines (see, for example, [4]).

The Turing machines we consider will have $t + u$ tapes. We call the first t tapes *external memory tapes* (and think of them as representing t disks). We call the other u tapes *internal memory tapes*. The first tape is always viewed as the input tape, and the t th tape is viewed as the output tape. In the literature, the input tape often is read-only and the output tape write-only. In the present paper, we do not need such restrictions.

\mathbb{N} denotes the set of natural numbers (that is, positive integers).

Definition 1. Let $T = (Q, \Sigma, \delta, q_0, F)$ be a (deterministic) Turing machine (TM, for short) with $t + u$ tapes, where Q is the state space, Σ the alphabet, $q_0 \in Q$ the start state, $F \subseteq Q$ the set of final states, and

$$\delta : (Q \setminus F) \times \Sigma^{t+u} \rightarrow Q \times \Sigma^{t+u} \times \{L, N, R\}^{t+u}$$

the transition function. Here L, N, R are special symbols indicating the head movements.

We assume that all tapes are one-sided infinite and have cells numbered $1, 2, 3$, etc. A *configuration* of T is a tuple

$$(q, p_1, \dots, p_{t+u}, w_1, \dots, w_{t+u}) \in Q \times \mathbb{N}^{t+u} \times (\Sigma^*)^{t+u},$$

where q is the current state, p_1, \dots, p_{t+u} are the positions of the heads on the tapes, and w_1, \dots, w_{t+u} are the contents of the tapes.

We assume that $\square \in \Sigma$ is the “blank” symbol which, at the beginning of the TM’s computation, is the inscription of all empty tape cells.

A configuration is called *final* if its current state q is final, that is, $q \in F$. Note that a *final* configuration does not have a successor configuration.

A *run* of T is a sequence $p = (p_j)_{j \in J}$ of configurations satisfying the obvious requirements. We are only interested in finite runs here, where the index set J is $\{1, \dots, \ell\}$ for an $\ell \in \mathbb{N}$.

Without loss of generality we assume that our Turing machines are *normalised* in such a way that in each step at most one of its heads moves to the left or to the right. \dashv

For a sequence $\bar{v} = (v_1, \dots, v_n)$ of integers we let

$$\text{rev}(\bar{v}) := \min \{k \geq 0 \mid \text{there are } 1 = n_0 < n_1 < n_2 < \dots < n_k < n_{k+1} = n \text{ such that for all } j \leq k \text{ the subsequence } v_{n_j}, \dots, v_{n_{j+1}} \text{ is monotone, that is, either } v_{n_j} \leq \dots \leq v_{n_{j+1}} \text{ or } v_{n_j} \geq \dots \geq v_{n_{j+1}}\}.$$

If we think of v_j as being the position after step j of a particle moving on the integers (or of a head moving on a Turing machine tape), then $\text{rev}(\bar{v})$ is the number of times the particle changes its direction.

Let T be a Turing machine and ρ a finite run of T . Let $i \geq 1$ be the number of a tape and \bar{p}_i the sequence of positions of the head on tape i in the run ρ . Then we let

$$\text{rev}(\rho, i) := \text{rev}(\bar{p}_i).$$

Intuitively, $\text{rev}(\rho, i)$ is the number of times the i th head changes its direction in the run ρ . Furthermore, we let

$$\text{space}(\rho, i)$$

be the number of cells of tape i that are used by ρ .

Definition 2. Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ and $t \in \mathbb{N}$.

- (1) A deterministic Turing machine T is (r, s, t) -bounded, if every run ρ of T on an input of length N satisfies the following conditions:
 - ρ is finite.
 - $1 + \sum_{i=1}^t \text{rev}(\rho, i) \leq r(N)$.¹
 - $\sum_{i=t+1}^{t+u} \text{space}(\rho, i) \leq s(N)$, where $t+u$ is the total number of tapes of T .
- (2) A function or decision problem belongs to the class $\text{ST}(r, s, t)$, if it can be solved by an (r, s, t) -bounded Turing machine.
- (3) For classes R and S of functions we let

$$\text{ST}(R, S, t) := \bigcup_{r \in R, s \in S} \text{ST}(r, s, t).$$

We also let

$$\text{ST}(R, S, O(1)) := \bigcup_{t \geq 1} \text{ST}(R, S, t). \quad \dashv$$

Note that we put no restriction on the running time or the space used on the first t tapes of an (r, s, t) -bounded Turing machine. However, the following lemma shows that these parameters cannot get too large.

Lemma 3. Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ and $t \in \mathbb{N}$, and let T be an (r, s, t) -bounded Turing machine T . Then for every finite run

$$\rho = (\rho_1, \dots, \rho_\ell)$$

of T on an input of size N we have

$$\ell \leq N \cdot 2^{O(r(N) \cdot (t+s(N)))}$$

and thus $\sum_{i=1}^t \text{space}(\rho, i) \leq N \cdot 2^{O(r(N) \cdot (t+s(N)))}$.

¹It is convenient for technical reasons to add 1 to the number $\sum_{i=1}^t \text{rev}(\rho, i)$ of changes of the head direction. As defined here, $r(N)$ thus bounds the number of sequential scans of the external memory tapes rather than the number of changes of head directions.

Proof: Let $T = (Q, \Sigma, \delta, q_0, F)$ be an (r, s, t) -bounded Turing machine with $t+u$ tapes, and let $\rho = (\rho_1, \dots, \rho_\ell)$ be the run of T on an input $v \in \Sigma^*$ with $|v| = N$. Let $r := r(N)$ and $s := s(N)$.

Let \hat{Q} be the set of potential configurations of the tapes $t+1, \dots, t+u$, together with the current state of T , that is,

$$\hat{Q} = \{ (q, p_{t+1}, \dots, p_{t+u}, w_{t+1}, \dots, w_{t+u}) \mid q \in Q, p_{t+i} \in \{1, \dots, s\}, w_{t+i} \in \Sigma^{\leq s} \text{ for all } i \in \{1, \dots, u\} \}.$$

(Note that since T is (r, s, t) -bounded, the tapes $t+1, \dots, t+u$ always have length at most s .)

We have

$$|\hat{Q}| \leq |Q| \cdot s^u \cdot (|\Sigma| + 1)^s = 2^{O(s)}.$$

In a finite run, T can make at most $|\Sigma|^t \cdot |\hat{Q}|$ steps without moving any of the heads on the first t tapes (otherwise, T would loop forever, contradicting the assumption that every run is finite).

Furthermore, without changing the *direction* of a head on any of the tapes $1, \dots, t$, T can make at most

$$k \cdot |\Sigma|^t \cdot |\hat{Q}| \quad (1)$$

steps, where k is the sum of the current lengths of the strings on the first t tapes. In each step the sum of the lengths of the strings on the first t tapes increases by at most t , and hence remains $\leq k \cdot |\Sigma|^t \cdot |\hat{Q}| \cdot (t+1)$ if the direction of no head on the first t tapes changes.

Initially, the sum of the lengths of the strings on the first t tapes is the input length N . An easy induction based on (1) shows that with at most i changes of the direction of a head on the first t tapes, the machine can make at most

$$N \cdot (|\Sigma|^t \cdot |\hat{Q}| \cdot (t+1))^{i+1}$$

steps.

Thus with $r-1$ changes of head directions, the total number of steps is bounded by

$$N \cdot (|\Sigma|^t \cdot |\hat{Q}| \cdot (t+1))^r = N \cdot 2^{O((t+s) \cdot r)},$$

where the constant hidden in the O -notation only depends on the parameters Σ, Q, u of the Turing machine T (and not on N, r, s, t). \square

If we think of the first t tapes of an (r, s, t) -bounded Turing machine as representing hard disks, then admitting heads to reverse their direction may not be very realistic. But as we use our model to prove *lower* bounds, it does not do any harm either. We mainly use the reversals to simulate *random access*. We can explicitly include random access into our model as follows: A *random access Turing machine* is a Turing machine T which has a special *address tape* on which only binary strings can be written. These binary strings are interpreted as nonnegative integers specifying external memory addresses, that is, numbers of cells on the external memory tapes. For each external memory tape $i \in \{1, \dots, t\}$ the machine has a special state r_i . If r_i is entered, then in one step the head on tape i is moved to the cell that is specified by the number on the address tape, and the content of the address tape is deleted.

Definition 4. Let $q, r, s : \mathbb{N} \rightarrow \mathbb{N}$ and $t \in \mathbb{N}$. A random access Turing machine T is (q, r, s, t) -bounded, if it is (r, s, t) -bounded (in the sense of a standard Turing machine) and, in addition, if every run ρ of T on an input of length N involves at most $q(N)$ random accesses. \dashv

The address tape is considered as part of the internal memory; thus in a (q, r, s, t) -bounded random access Turing machine the length of the address tape is bounded by $s(N)$, where N is the length of the

input. This implies that we can only address the first $2^{s(N)}$ cells of the external memory tapes. If during a computation, these tapes get longer, we only have random access to initial segments of length $2^{s(N)}$.²

Lemma 5. *Let $q, r, s : \mathbb{N} \rightarrow \mathbb{N}$ and $t \in \mathbb{N}$. Then if a problem can be solved by a (q, r, s, t) -bounded random access Turing machine, it can also be solved by a $(r + 2q, O(s), t)$ -bounded Turing machine.*

Proof: Noting that a random access can be simulated with at most two reversals of the direction of the head movement, the proof is straightforward. \square

From now on, we will focus on standard Turing machines without address tapes.

3. Main results

We define the *sorting problem* as a mapping on sequences of strings over $\{0, 1\}^*$, which of course we may interpret as nonnegative integers. We use $\#$ as a separator symbol, that is, we encode inputs and outputs as strings over the alphabet $\{0, 1, \#\}$. We sort strings lexicographically and denote the lexicographical order on $\{0, 1\}^*$ by \preceq . If we want to sort nonnegative integers, we pad them with leading 0s.

Formally, the *sorting problem* is defined as follows:

SORT

Input: $v_1\# \dots v_m\#$, where $m \geq 1$ and $v_1, \dots, v_m \in \{0, 1\}^*$.

Output: $v_{\psi(1)}\# \dots v_{\psi(m)}\#$, where ψ is a permutation of $\{1, \dots, m\}$ such that $v_{\psi(1)} \preceq \dots \preceq v_{\psi(m)}$.

Our main technical result is a lower bound for the sorting problem restricted to inputs which are permuted in a specific way, which only depends on m .

For a sequence $\vec{v} := (v_1, \dots, v_m)$ of strings, we let $\sigma(\vec{v})$ be the length of the longest (increasingly or decreasingly) sorted subsequence of \vec{v} , that is, the maximum ℓ such that there are $1 \leq j_1 < \dots < j_\ell \leq m$ with $v_{j_1} < \dots < v_{j_\ell}$ or $v_{j_1} > \dots > v_{j_\ell}$. The number $\sigma(\vec{v})$ is a measure for the ‘‘sortedness’’ of \vec{v} . We shall now define an ordering on strings that generates sequences with a high σ -value. Let \preceq denote the ‘‘lexicographical ordering on reverse strings’’ on $\{0, 1\}^*$, that is, for $v, w \in \{0, 1\}^*$ let $v \preceq w$ if v^{-1} is lexicographically smaller than or equal to w^{-1} . For string $v = v_1 \dots v_n$ we let $v^{-1} = v_n \dots v_1$. For every $m \geq 1$, we let φ_m be the permutation of $\{1, \dots, m\}$ with

$$\text{bin}(\varphi_m(1)) \preceq \dots \preceq \text{bin}(\varphi_m(m)),$$

where $\text{bin}(i) \in \{0, 1\}^*$ denote the binary representation of i .

Lemma 6. *For every $m \geq 1$ we have*

$$\sigma(\varphi_m(1), \dots, \varphi_m(m)) \leq 2\sqrt{m} - 1.$$

Furthermore, for $i \leq m$, the value $\varphi_m(i)$ can be computed in space $O(\log m)$.

We omit the straightforward proof.

We keep φ_m fixed for the rest of the paper. If m is clear from the context, we just write φ instead of φ_m . We are now ready to state the main technical theorem:

²We could also put a length restriction of at most $2^{s(N)}$ on all the external memory tapes. This does not affect our lower bound results.

Theorem 7 (Main Technical Theorem). *Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ such that $r(N) = o(\log N)$ and $s(N) = o\left(\frac{\sqrt[5]{N}}{r(N)}\right)$.*

Then the following restriction of SORT is not in $\text{ST}(r, s, O(1))$:

φ -SORT

Input: $v_{\varphi(1)}\# \dots v_{\varphi(m)}\#$,

where $m \geq 1$, $n = m^4$, and $v_1 < \dots < v_m \in \{0, 1\}^n$.

Output: $v_1\# \dots v_m\#$

Sections 4–7 are devoted to a proof of the theorem. Before we start the proof, let us derive our main results from this theorem. The first follows immediately from Theorem 7:

Theorem 8 (Lower Bound for General Sorting). *SORT is not in*

$$\text{ST}\left(o(\log N), O\left(\frac{\sqrt[5]{N}}{\log N}\right), O(1)\right).$$

The following theorem shows that for sorting m numbers of size $m^{O(1)}$ (or strings of length $O(\log m)$) we get an asymptotically tight lower bound. For an arbitrary constant $c \geq 6$, we consider the following restriction of SORT:

SHORT-SORT

Input: $v_1\# \dots v_m\#$, where $m \geq 1$ and $v_1, \dots, v_m \in \{0, 1\}^*$ such that $|v_i| \leq c \cdot \lceil \log m \rceil$ for $1 \leq i \leq m$.

Output: $v_{\psi(1)}\# \dots v_{\psi(m)}\#$, where ψ is a permutation of $\{1, \dots, m\}$ such that $v_{\psi(1)} \preceq \dots \preceq v_{\psi(m)}$.

Theorem 9 (Lower Bound for Sorting Short Strings). *SHORT-SORT is in $\text{ST}(O(\log m), O(\log m), 3)$, but not in*

$$\text{ST}(o(\log m), O(\sqrt[5]{m}), O(1)).$$

Proof: The upper bound is obtained by the standard merge sort algorithm.

For the lower bound, we reduce φ -SORT to SHORT-SORT in such a way that the reduction can be carried out in

$$\text{ST}(O(1), O(\log N), 2).$$

Let $m \geq 1$, $m = n^4$, and $v_1 < \dots < v_m \in \{0, 1\}^n$. We consider the instance

$$\vec{v} = v_{\varphi(1)}\# \dots v_{\varphi(m)}\#$$

of φ -SORT.

Let $v_i = v_{i1} \dots v_{in}$, where $v_{ij} \in \{0, 1\}$. For $1 \leq i \leq m$, let $x_i \in \{0, 1\}^*$ be the binary representation of $i - 1$ padded to a string of length $\lceil \log m \rceil$ by adding leading 0s. Similarly, for $1 \leq i \leq n$, let $y_i \in \{0, 1\}^*$ be the binary representation of $i - 1$ padded to a string of length $\lceil \log n \rceil$ by adding leading 0s. For $1 \leq i \leq m$, $1 \leq j \leq n$ let

$$w_{ij} = x_i x_j v_{ij}.$$

Our reduction maps the instance \vec{v} of φ -SORT to the instance

$$\vec{w} = w_{\varphi(1)1}\# \dots w_{\varphi(1)n}\# \dots w_{\varphi(m)1}\# \dots w_{\varphi(m)n}\#$$

of SHORT-SORT. Note that $|w_{ij}| \leq 5 \cdot \lceil \log m \rceil + 1$, thus \vec{w} really is an instance of SHORT-SORT. The sorted output is

$$\begin{array}{lll} x_1 x_1 v_{11}\# & \dots & x_1 x_n v_{1n}\# \\ x_2 x_1 v_{21}\# & \dots & x_2 x_n v_{2n}\# \\ & \vdots & \\ x_m x_1 v_{m1}\# & \dots & x_m x_n v_{mn}\# \end{array}$$

From this, we can easily construct the sorted output $v_1\#\dots v_m\#$ for the original instance \bar{v} .

The size of the instance \bar{v} is $N := m \cdot (n + 1)$. The instance \bar{w} consists of $m \cdot n$ strings, each of length $5 \cdot \lceil \log m \rceil + 1$. In particular, \bar{w} has length $m' = m \cdot n \cdot (5 \cdot \lceil \log m \rceil + 2)$. Note that $m' \leq 5 \cdot N \cdot \log N$. Let $r(m') = o(\log m')$ and $s(m') = O(\sqrt[r(m')]{m'})$ such that $r(m') = \omega(1)$ and $s(m') = \omega(\log m')$. Suppose for contradiction that SHORT-SORT is in $\text{ST}(r(m'), s(m'), O(1))$. Since

$$r(m') = o(\log m') \leq o(\log(5 \cdot N \cdot \log N)) = o(\log N)$$

and

$$s(m') = O(\sqrt[r(m')]{m'}) \leq O(\sqrt[5]{5 \cdot N \cdot \log N}) \leq O(\frac{\sqrt[5]{N}}{\log N}),$$

we then obtain that φ -SORT is in

$$\text{ST}\left(o(\log N), O\left(\frac{\sqrt[5]{N}}{\log N}\right), O(1)\right).$$

This contradicts Theorem 7. \square

4. List Machines

Definition 10. A *list machine (LM)* is a tuple

$$M = (t, m, I, A, a_0, \alpha, B, \omega)$$

consisting of

- a $t \in \mathbb{N}$, the *number of lists*.
- an $m \in \mathbb{N}$, the *length of the input*.
- a finite set I whose elements are called *input numbers* (usually, $I \subseteq \mathbb{N}$ or $I \subseteq \{0, 1\}^*$).
- a finite set A whose elements are called (*abstract*) *states*.
We assume I and A to be disjoint and not to contain the two special symbols ‘ \langle ’ and ‘ \rangle ’. We call $\mathbb{A} := I \cup A \cup \{\langle, \rangle\}$ the *alphabet* of the machine.
- an *initial state* $a_0 \in A$.
- a *transition function*

$$\alpha : (A \setminus B) \times (\mathbb{A}^*)^t \rightarrow A \times \text{Movement}^t$$

with

$$\text{Movement} := \left\{ (\text{head-direction}, \text{move}) \mid \begin{array}{l} \text{head-direction} \in \{-1, +1\}, \\ \text{move} \in \{\text{true}, \text{false}\} \end{array} \right\}.$$

- a set $B \subseteq A$ of *final states*.
- an *output function* $\omega : \mathbb{A}^* \rightarrow \Gamma^*$, for a (finite) alphabet Γ . \dashv

Intuitively, an LM $M = (t, m, I, A, a_0, \alpha, B, \omega)$ operates as follows: The input is a sequence $(v_1, \dots, v_m) \in I^m$. Instead of tapes (as a Turing machine), an LM operates on t lists. As for tapes, there is a read-write head operating on each list. Cells of the lists store strings in \mathbb{A}^* (and not just symbols from \mathbb{A}). Initially, the first list, called the *input list*, contains (v_1, \dots, v_m) , and all other lists are empty. The heads are on the left end of the lists. In each step of the computation, the heads move according to the transition function. If a head changes its direction or moves to the left or right, then the current state and the content of all current head positions are written behind each head (it will be made precise below what ‘behind’ means). If a final state is reached, the machine stops. Then the output is computed by applying the output function cell-wise to the last list and concatenating the strings it returns.

Definition 11. (1) A *configuration* of an LM

$$M = (t, m, I, A, a_0, \alpha, B, \omega)$$

is a tuple (a, \vec{p}, \vec{d}, X) , where

- $a \in A$ is the *current state*,

$$- \vec{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_t \end{pmatrix} \in \mathbb{N}^t \text{ is the tuple of } \textit{head positions},$$

$$- \vec{d} = \begin{pmatrix} d_1 \\ \vdots \\ d_t \end{pmatrix} \in \{-1, +1\}^t \text{ is the tuple of } \textit{head directions},$$

$$- X = \begin{pmatrix} \vec{x}_1 \\ \vdots \\ \vec{x}_t \end{pmatrix}, \text{ where } \vec{x}_i = (x_{i,1}, \dots, x_{i,m_i}) \in (\mathbb{A}^*)^{m_i} \text{ for some } m_i \geq 1, \text{ contains the } \textit{content of the cells}. \text{ (The string } x_{i,j} \in \mathbb{A}^* \text{ is the content of the } j\text{th cell of the } i\text{th list.)}$$

- (2) The *initial configuration* for input $(v_1, \dots, v_m) \in I^m$ is the configuration (a, \vec{p}, \vec{d}, X) , where $a = a_0$, $\vec{p} = (1, \dots, 1)^\top$, $\vec{d} = (+1, \dots, +1)^\top$, and $X = (\vec{x}_1, \dots, \vec{x}_t)^\top$ with

$$\vec{x}_1 = (\langle v_1 \rangle, \dots, \langle v_m \rangle) \in (\mathbb{A}^*)^m$$

$$\text{and } \vec{x}_2 = \dots = \vec{x}_t = (\langle \rangle) \in (\mathbb{A}^*)^1.$$

- (3) The *successor* $S(a, \vec{p}, \vec{d}, X)$ of a configuration (a, \vec{p}, \vec{d}, X) is the configuration $(a', \vec{p}', \vec{d}', X')$ defined as follows: Suppose that

$$\alpha(a, x_{1,p_1}, \dots, x_{t,p_t}) = (b, e_1, \dots, e_t)$$

We let $a' = b$. For $1 \leq i \leq t$, let m_i be the length of the list \vec{x}_i , and let

$$e'_i := (\text{head-direction}_i, \text{move}_i)$$

$$:= \begin{cases} (-1, \text{false}) & \text{if } p_i = 1 \text{ and } e_i = (-1, \text{true}), \\ (+1, \text{false}) & \text{if } p_i = m_i \text{ and } e_i = (+1, \text{true}), \\ e_i & \text{otherwise.} \end{cases}$$

We define $f_i \in \{0, 1\}$ such that $f_i = 1$ if, and only if, $(\text{move}_i = \text{true} \text{ or } \text{head-direction}_i \neq d_i)$.

If $f_i = 0$ for all $i \in \{1, \dots, t\}$, then we let $\vec{p}' := \vec{p}$, $\vec{d}' := \vec{d}$, and $X' := X$.

So suppose that there is at least one i such that $f_i \neq 0$. Furthermore, let

$$y := a \langle x_{1,p_1} \rangle \cdots \langle x_{t,p_t} \rangle$$

We let

$$\vec{x}'_i = \begin{cases} (x_{i,1}, \dots, x_{i,p_i-1}, y, x_{i,p_i+1}, \dots, x_{i,m_i}) & \text{if } \text{move}_i = \text{true}, \\ (x_{i,1}, \dots, x_{i,p_i-1}, y, x_{i,p_i}, x_{i,p_i+1}, \dots, x_{i,m_i}) & \text{if } d_i = +1 \text{ and } \text{move}_i = \text{false}, \\ (x_{i,1}, \dots, x_{i,p_i-1}, x_{i,p_i}, y, x_{i,p_i+1}, \dots, x_{i,m_i}) & \text{if } d_i = -1 \text{ and } \text{move}_i = \text{false}, \end{cases}$$

and, finally,

$$p'_i = \begin{cases} p_i + 1 & \text{if } e'_i = (+1, \text{true}), \\ p_i - 1 & \text{if } e'_i = (-1, \text{true}), \\ p_i + 1 & \text{if } d_i = +1 \text{ and } e'_i = (+1, \text{false}), \\ p_i & \text{if } d_i = +1 \text{ and } e'_i = (-1, \text{false}), \\ p_i & \text{if } d_i = -1 \text{ and } e'_i = (-1, \text{false}), \\ p_i + 1 & \text{if } d_i = -1 \text{ and } e'_i = (+1, \text{false}). \end{cases}$$

- (4) A configuration (a, \vec{p}, \vec{d}, X) is *final* if $a \in B$. A (finite) *run* of the machine is a sequence $(\rho_1, \dots, \rho_\ell)$ of configurations, where ρ_1 is the initial configuration for some input, ρ_ℓ is final and $S(\rho_i) = \rho_{i+1}$ for $1 \leq i < \ell$. Note that for each input $(v_1, \dots, v_m) \in I^m$ there is at most one run from the initial configuration for (v_1, \dots, v_m) to a final configuration.
- (5) To define the output of the machine, let $(v_1, \dots, v_m) \in I^m$ be an input. If there is no finite run with input (v_1, \dots, v_m) , we let $f_M(v_1, \dots, v_m)$ be undefined. Otherwise, let (a, \vec{p}, \vec{d}, X) be the final configuration reached from the initial configuration for input (v_1, \dots, v_m) . Let

$$o := \omega(x_{r,1}) \cdots \omega(x_{t,m_r}).$$

Then $o \in \Gamma^*$. We let

$$f_M(v_1, \dots, v_m) := o.$$

Thus our machine M computes a partial function $f_M : I^m \rightarrow \Gamma^*$. \dashv

Remark 12. Let $M = (t, m, I, A, a_0, \alpha, B, \omega)$ be an LM. We define the following subsets of \mathbb{A}^* :

$$L_1 := \{a\langle y_1 \rangle \cdots \langle y_t \rangle \mid a \in A, y_1 \in \{\langle v \rangle : v \in I\}, \\ y_2 = \cdots = y_t = \langle \rangle\}$$

and, for all $i \geq 1$,

$$L_{i+1} := L_i \cup \{a\langle y_1 \rangle \cdots \langle y_t \rangle \mid a \in A \text{ and } y_1, \dots, y_t \in L_i\}.$$

It is straightforward to see the following: In the start configuration, every item in list 1 stores a string of the form $\langle v \rangle$, for some $v \in I$, whereas each of the lists $2, \dots, t$ have exactly one item, and this unique item stores the string $\langle \rangle$. Each of the list items generated during the computation until the first head reversal stores a string that belongs to L_1 . In particular, until the first head reversal, each list $j \in \{2, \dots, t\}$ is of the form

$$\vec{x}_j = (x_{j,1}, \dots, x_{j,m_j})$$

with $x_{j,v} \in L_1$, for all $v < m_j$, and $x_{j,m_j} = \langle \rangle$; and the head of list j points to its last item x_{j,m_j} .

Similarly, until the first reversal of any of the heads $j \in \{1, \dots, t\}$, list 1 is of the form

$$\vec{x}_1 = (x_{1,1}, \dots, x_{1,m_1})$$

where, for some $p \in \{1, \dots, m_1\}$, we have $x_{1,v} \in L_1$, for all $v < p$, and $x_{1,v} \in \{\langle v \rangle : v \in I\}$, for all $v \geq p$; and the head of list 1 points to item, $x_{1,p}$.

In general, during the computation until the i th head reversal, every list item stores a string from

$$L_i \cup \{\langle v \rangle \mid v \in I\} \cup \{\langle \rangle\}. \quad \dashv$$

Definition 13. For every run ρ of an LM

$$M = (t, m, I, A, a_0, \alpha, B, \omega)$$

and every $i \leq t$, we define $\text{rev}(\rho, i)$ to be the number of changes of the direction of head i in run ρ . We say that M is *r-bounded*, for some $r \in \mathbb{N}$, if every run ρ is of M is finite (that is, f_M is total) and

$$\sum_{i=1}^t \text{rev}(\rho, i) \leq r. \quad \dashv$$

5. The Simulation Lemma

Lemma 14 (Simulation Lemma). Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$, $t \in \mathbb{N}$, and let $T = (Q, \Sigma, \delta, q_0, F)$ be an (r, s, t) -bounded Turing machine with a total number of $t+u$ tapes, which computes a function

$$f_T : (\Sigma \setminus \{\square\})^* \rightarrow (\Sigma \setminus \{\square\})^*.$$

Then for all $m, n \in \mathbb{N}$ there exists an $r(m \cdot (n+1))$ -bounded LM

$$M_{m,n} = M = (t, m, I, A, a_0, \alpha, B, \omega)$$

with $I = (\Sigma \setminus \{\square, \#\})^n$ and

$$|A| \leq 2^{d \cdot t^2 \cdot r(m \cdot (n+1)) \cdot s(m \cdot (n+1)) + 3t \cdot \log(m \cdot (n+1))} \quad (2)$$

for some number $d = d(u, |Q|, |\Sigma|)$ that does not depend on r, m, n, t , such that

$$f_M(v_1, \dots, v_m) = f_T(v_1 \# \cdots v_m \#)$$

for all $(v_1, \dots, v_m) \in I^m$.

The proof of this lemma is the technically most difficult part of this paper. Before we get into technical details, let us briefly sketch the idea: Let T be a TM. We construct an LM M that simulates T . The lists of M represent the external memory tapes of T . More precisely, the cells of the lists of M represent segments, or *blocks*, of the corresponding external memory tapes of T in such a way that the content of a block at any step of the computation can be reconstructed from the content of the cell representing it. The blocks evolve dynamically in a way that is described below. The *states* of M encode:

- The current state of the Turing machine T .
- The content and the head positions of the internal memory tapes $t+1, \dots, t+u$ of T .
- The head positions of the external memory tapes $1, \dots, t$.
- For each of the external memory tapes $1, \dots, t$, the boundaries of the block in which the head currently is.

Representing T 's current state and the content and head positions of the u internal memory tapes requires $|Q| \cdot 2^{O(s(m \cdot (n+1)))} \cdot s(m \cdot (n+1))^u$ states. The t head positions of the external memory tapes increase the number of states by a factor of ℓ^t , where $\ell = \ell(m \cdot (n+1))$ is an upper bound on the tape length which we obtain from Lemma 3. The $2t$ block boundaries increase the number of states by another factor of ℓ^{2t} . So overall, the number of states is bounded by

$$|Q| \cdot 2^{O(s(m \cdot (n+1)))} \cdot s(m \cdot (n+1))^u \cdot \ell^{3t}.$$

By Lemma 3, this yields the bound (2).

Initially, the first tape is split into m blocks which contain the input segments $v_i \#$ (for $1 \leq i \leq m$ and $v_i \in I$), and all other tapes just consist of one block which contains the blank string \square^ℓ . The heads in the initial configuration of M are on the first cells of their lists. Now we start the simulation. As long as no head of the external memory tapes of T changes its direction or crosses the boundaries

of its current block, M does not do anything. If a head on a tape $i_0 \in \{1, \dots, t\}$ crosses the boundaries of its block, the head i_0 of M moves to the next cell, and the previous cell is overwritten with sufficient information so that if it is visited again later, the content of the corresponding block of tape i_0 of T can be reconstructed. The blocks on all other tapes are split behind the current head position ("behind" is defined relative to the current direction in which the head moves). A new cell is inserted into the lists behind the head, this cell represents the newly created tape block that is behind the head. The newly created block starting with the current head position is represented by the (old) cell on which the head still stands. The case that a head on a tape $i_0 \in \{1, \dots, t\}$ changes its direction is treated similarly.

After the simulation stops, the output is created by applying the function ω to the content of the last list, which represents the content of the last external memory tape (that is, the output tape). Thus, all ω has to do is reconstruct the tape content from the list content.

5.1. Proof of the Simulation Lemma. Let $T = (Q, \Sigma, \delta, q_0, F)$ be the given Turing machine with $t + u$ tapes, where the tapes $1, \dots, t$ are the external memory tapes and tapes $t + 1, \dots, t + u$ are the internal memory tapes. Let $m, n \in \mathbb{N}$ and $N = m \cdot (n + 1)$. Every tuple $\vec{v} = (v_1, \dots, v_m) \in I^m$ corresponds to an input string $\bar{v} := v_1 \# v_2 \# \dots \# v_m \#$ of length N . Let $r := r(N)$ and $s := s(N)$.

By Lemma 3, there is a constant $c_1 = c_1(u, |Q|, |\Sigma|)$, which does not depend on r, m, n, t , such that throughout the entire computation on an input \bar{v} (for any $\vec{v} \in I^m$), each of the external memory tapes $1, \dots, t$ of T has length at most

$$\ell(N) := N \cdot 2^{c_1 \cdot r \cdot (t+s)}. \quad (3)$$

Step 1: Definition of a superset \tilde{A} of M 's state set A .

Let \hat{Q} be the set of potential configurations of tapes $t+1, \dots, t+u$, together with the current state of T , that is,

$$\hat{Q} := \left\{ (q, p_{t+1}, \dots, p_{t+u}, w_{t+1}, \dots, w_{t+u}) \mid q \in Q, \right. \\ \left. p_{t+i} \in \{1, \dots, s\}, w_{t+i} \in \Sigma^{\leq s} \text{ (for all } i \in \{1, \dots, u\}) \right\}. \quad (4)$$

Then for a suitable constant $c_2 = c_2(u, |Q|, |\Sigma|)$ we have

$$|\hat{Q}| \leq 2^{c_2 \cdot s}. \quad (5)$$

We let

$$\tilde{A} := \left\{ (\hat{q}, \vec{p}_1, \dots, \vec{p}_t) \mid \hat{q} \in \hat{Q}, \right. \\ \text{and for each } j \in \{1, \dots, t\}, \\ \vec{p}_j = (p_j^{\llbracket}, p_j^{\uparrow}, p_j^{\rrbracket}, \text{head-direction}_j) \\ \text{with} \\ p_j^{\uparrow} \in \{1, \dots, \ell(N)\}, \\ \text{head-direction}_j \in \{+1, -1\}, \text{ and} \\ \text{either } p_j^{\llbracket} = p_j^{\rrbracket} = \ominus, \\ \text{or } p_j^{\llbracket}, p_j^{\rrbracket} \in \{1, \dots, \ell(N)\} \text{ with} \\ \left. (p_j^{\llbracket} = p_j^{\uparrow} \leq p_j^{\rrbracket} \text{ or } p_j^{\llbracket} \leq p_j^{\uparrow} = p_j^{\rrbracket}) \right\}$$

Here, \ominus is a symbol for indicating that p_j^{\llbracket} and p_j^{\rrbracket} are "undefined", that is, that they cannot be interpreted as positions on one of the Turing machine T 's tapes $1, \dots, t$.

Later, at the end of Step 3, we will specify, *which* particular subset of \tilde{A} will be designated as M 's state set A . With *any* choice of A as a subset of \tilde{A} we will have

$$|A| \leq |\tilde{A}| \leq |\hat{Q}| \cdot (\ell(N) + 1)^{3t} \cdot 2^t \\ \leq 2^{c_2 \cdot s} \cdot (N \cdot 2^{c_1 \cdot r \cdot (t+s)} + 1)^{3t} \cdot 2^t \leq 2^{d \cdot t^2 \cdot r \cdot s}$$

for a suitable constant $d = d(u, |Q|, |\Sigma|)$. This completes Step 1. \dashv

Step 2: Definition of M 's initial state a_0 and M 's set B of final states.

Let

$$\hat{q}_0 := (q_0, \underbrace{1, \dots, 1}_u, \underbrace{\square^s, \dots, \square^s}_u)$$

be the part of T 's *initial configuration* that describes the (start) state q_0 of T and the head positions and initial (empty) content of the tapes $t+1, \dots, t+u$ (that is, the tapes that represent internal memory).

Let

$$\vec{p}_1 := (p_1^{\llbracket}, p_1^{\uparrow}, p_1^{\rrbracket}, \text{head-direction}_1) \\ := \begin{cases} (1, 1, n+1, +1) & \text{if } m > 1, \\ (1, 1, m \cdot (n+1), +1) & \text{otherwise.} \end{cases}$$

As start state of the LM M we choose

$$a_0 := (\hat{q}_0, \vec{p}_1, \vec{p}_2, \dots, \vec{p}_t)$$

As M 's set of *final* states we choose $B := \tilde{B} \cap A$, with

$$\tilde{B} := \left\{ (\hat{q}, \vec{p}_1, \vec{p}_2, \dots, \vec{p}_t) \in A \mid \hat{q} \text{ is of the form } (q, \vec{p}, \vec{y}) \in \hat{Q} \right. \\ \left. \text{for some } q \in F \right\}$$

that is, a state of M is *final* if, and only if, the associated state of the Turing machine T is final. This completes Step 2. \dashv

Step 3: Definition of M 's transition function α .

We let

$$\text{Conf}_T := \left\{ (q, p_1, \dots, p_{t+u}, w_1, \dots, w_{t+u}) \mid q \in Q, \right. \\ \text{and for all } j \in \{1, \dots, t+u\} \\ p_j \in \mathbb{N} \\ \text{and for all } j \in \{1, \dots, t\} \\ w_j \in \{\otimes\}^* \Sigma^* \{\otimes\}^* \text{ with } w_{j,p_j} \in \Sigma \\ \text{and for all } j \in \{1, \dots, u\} \\ \left. w_{t+j} \in \Sigma^* \right\},$$

where \otimes is a symbol *not* in Σ , and w_{j,p_j} denotes the p_j -th letter in the string w_j .

Intended meaning: The symbol \otimes is used as a *wildcard* symbol that may be interpreted by any symbol in Σ . An element in Conf_T gives (potentially) incomplete information on a configuration of T , where the contents of tapes $1, \dots, t$ might be described only in some part (namely, in the part containing no \otimes -symbols).

We let $\tilde{A} := I \cup \tilde{A} \cup \{ \langle \cdot, \cdot \rangle \}$. By induction on i we fix, for $i \geq 0$,

- a set $A_i \subseteq \tilde{A}$
- a set $K_i \subseteq (\tilde{A} \setminus \tilde{B}) \times (\tilde{A}^*)^t$,

– a set $L_i \subseteq \tilde{A}^*$, letting

$$L_i := \{a\langle y_1 \rangle \cdots \langle y_t \rangle : (a, y_1, \dots, y_t) \in K_i\} \quad (6)$$

– a function

$$\text{config}_i : K_i \rightarrow \text{Conf}_T \cup \{\perp\}$$

Intended meaning: When the LM M is in a situation $\kappa \in K_i$, then $\text{config}_i(\kappa)$ is the TM's configuration at the beginning of M 's current step. If $\text{config}_i(\kappa) = \perp$, then κ does not represent a configuration of the TM.

– the transition function α of M , restricted to K_i , that is,

$$\alpha|_{K_i} : K_i \rightarrow \tilde{A} \times \text{Movement}^t$$

– for every tape $j \in \{1, \dots, t\}$, a function

$$\text{tape-config}_{j,i} : L_i \rightarrow \left\{ \begin{array}{l} (w, p^{\llbracket}, p^{\rrbracket}) \mid \\ \text{either } 1 \leq p^{\llbracket} \leq p^{\rrbracket} \leq \ell(N) \text{ and} \\ w \in \{\otimes\}^{p^{\llbracket}-1} \Sigma^{p^{\rrbracket}-p^{\llbracket}+1} \{\otimes\}^{\ell(N)-p^{\rrbracket}} \\ \text{or } p^{\llbracket} > p^{\rrbracket} \text{ and } w = \varepsilon \end{array} \right\}$$

Intended meaning: When the LM M is in a situation $\kappa = (a, \langle y_1 \rangle, \dots, \langle y_t \rangle) \in K_i$, then $\text{tape-config}_{j,i}(a\langle y_1 \rangle \cdots \langle y_t \rangle)$ gives information on the inscription from tape cell p^{\llbracket} up to tape cell p^{\rrbracket} of the j -th tape of the TM's configuration at the end of M 's current step.

Induction base ($i = 0$): We start with M 's start state a_0 and choose

$$A_0 := \{a_0\}.$$

If a_0 is *final*, then we let $K_0 := \emptyset$ and $A := A_0$. This then gives us an LM M which accepts its input without performing a single step. This is fine, since a_0 is final if, and only if, the Turing machine T 's start state q_0 is final, that is, T accepts its input without performing a single step.

For the case that a_0 is *not final*, we let

$$K_0 := \left\{ (a_0, y_1, \dots, y_t) \mid y_1 \in \{\langle v \rangle : v \in I\} \text{ and } y_2 = \dots = y_t = \langle \rangle \right\}.$$

The set L_0 is defined via equation (6).

The function config_0 is defined as follows: For every

$$\kappa = (a_0, y_1, \dots, y_t) \in K_0$$

with $y_1 = \langle v \rangle$ (for some $v \in I$), let

$$\text{config}_0(\kappa) := (q_0, \overbrace{1, \dots, 1}^{t+u}, v \# \otimes^{\ell(N)-(n+1)}, \underbrace{\square^{\ell(N)}, \dots, \square^{\ell(N)}}_{t-1}, \underbrace{\square^s, \dots, \square^s}_u).$$

Let

$$(\hat{q}_0, \vec{p}_1, \dots, \vec{p}_t) := a_0$$

with $\vec{p}_j = (p_j^{\llbracket}, p_j^{\uparrow}, p_j^{\rrbracket}, \text{head-direction}_j)$, for all $j \in \{1, \dots, t\}$.

For $j \in \{1, \dots, t\}$ we define

$$(\hat{p}_j^{\llbracket}, p_j^{\uparrow}, \hat{p}_j^{\rrbracket}) := (p_j^{\llbracket}, p_j^{\uparrow}, p_j^{\rrbracket}).$$

For defining $\alpha|_{K_0}(\kappa)$ and $\text{tape-config}_{j,0}(a\langle y_1 \rangle \cdots \langle y_t \rangle)$, consider the following: Let us start the Turing machine T with a configuration γ_0 that fits to $\text{config}_0(\kappa)$, that is, that can be obtained from $\text{config}_0(\kappa)$

by replacing each occurrence of the wildcard symbol \otimes by an arbitrary symbol in Σ . Let $\gamma_0, \gamma_1, \gamma_2, \dots$ be the successive configurations of T when started in γ_0 , that is, $\gamma_{v+1} = S(\gamma_v)$, for all $v \geq 0$.

Using this notation, the definition of $\alpha|_{K_0}(\kappa)$ and

$$\text{tape-config}_{j,0}(a\langle y_1 \rangle \cdots \langle y_t \rangle)$$

can be taken verbatim from the definition of $\alpha|_{K_{i+1}}(\kappa)$ and

$$\text{tape-config}_{j,i+1}(a\langle y_1 \rangle \cdots \langle y_t \rangle),$$

given below. This completes the induction base ($i = 0$).

Induction step ($i \rightarrow i+1$): We let

$$A_{i+1} := \{b \in \tilde{A} \mid \text{there is some } \kappa \in K_i \text{ such that}$$

$$\alpha|_{K_i}(\kappa) = (b, e_1, \dots, e_t)$$

$$\text{(for suitable } (e_1, \dots, e_t) \in \text{Movement}^t)\}$$

and

$$K_{i+1} := \{(a, y_1, \dots, y_t) \mid a \in A_{i+1} \setminus \tilde{B},$$

$$y_1 \in \{\langle v \rangle : v \in I\} \cup \bigcup_{i' \leq i} L_{i'}, \text{ and}$$

$$y_j \in \{\langle \rangle\} \cup \bigcup_{i' \leq i} L_{i'}, \text{ for all } j \in \{2, \dots, t\}\}$$

The set L_{i+1} is defined via equation (6).

The function config_{i+1} is defined as follows: Let $\kappa = (a, y_1, \dots, y_t) \in K_{i+1}$. Let

$$(\hat{q}, \vec{p}_1, \dots, \vec{p}_t) := a$$

with $\vec{p}_j = (p_j^{\llbracket}, p_j^{\uparrow}, p_j^{\rrbracket}, \text{head-direction}_j)$, for all $j \in \{1, \dots, t\}$, and

$$\hat{q} = (q, p_{t+1}, \dots, p_{t+u}, w_{t+1}, \dots, w_{t+u}).$$

Let $j \in \{1, \dots, t\}$.

If $y_j \in L_{i'}$ for some $i' \leq i$, then let

$$(w_j^{\llbracket}, p_j^{\llbracket}, p_j^{\rrbracket}) := \text{tape-config}_{j,i'}(y_j).$$

We choose $w_j := w_j^{\llbracket}$. (This is well-defined, because $\text{tape-config}_{j,i'}$ and $\text{tape-config}_{j,i''}$ operate identical on all elements in $L_{i'} \cap L_{i''}$, for all $i', i'' \leq i$).

Furthermore, we let $(\hat{p}_j^{\llbracket}, \hat{p}_j^{\rrbracket})$ be defined as follows:

$$(\hat{p}_j^{\llbracket}, \hat{p}_j^{\rrbracket}) := \begin{cases} (p_j^{\uparrow}, p_j^{\rrbracket}) & \text{if } p_j^{\llbracket} = p_j^{\rrbracket} = \ominus \text{ and } \text{head-direction}_j = +1 \\ (p_j^{\llbracket}, p_j^{\uparrow}) & \text{if } p_j^{\llbracket} = p_j^{\rrbracket} = \ominus \text{ and } \text{head-direction}_j = -1 \\ (p_j^{\llbracket}, p_j^{\rrbracket}) & \text{otherwise.} \end{cases}$$

If $y_j \notin L_{i'}$, for every $i' \leq i$, then we make a case distinction on j : In case that $j \in \{2, \dots, t\}$, we have $y_j = \langle \rangle$ and $\text{head-direction}_j = +1$. We define $(\hat{p}_j^{\llbracket}, \hat{p}_j^{\rrbracket})$ as follows:

$$(\hat{p}_j^{\llbracket}, \hat{p}_j^{\rrbracket}) := (p_j^{\uparrow}, \ell(N)),$$

and choose

$$w_j := \otimes^{\uparrow_{i'}-1} \square^{\ell(N)-(\uparrow_{i'}-1)}.$$

In case that $j = 1$, we know that y_j must be of the form $\langle v \rangle$, for some $v \in I$, and $\text{head-direction}_j = +1$. If v is *not* the m -th input item, that is, there is some $\mu \in \{1, \dots, m-1\}$ such that $(\mu-1) \cdot (n+1) < p_1^{\uparrow} \leq \mu \cdot (n+1)$, then we define

$$(\hat{p}_1^{\llbracket}, \hat{p}_1^{\rrbracket}) := (p_1^{\uparrow}, \mu \cdot (n+1)),$$

and choose

$$w_1 := \otimes^{(\mu-1)\cdot(n+1)} \nu \# \otimes^{\ell(N)-\mu\cdot(n+1)}.$$

Otherwise, ν must be the m -th input item, that is,

$$p_1^\uparrow > (m-1)\cdot(n+1).$$

In this case we define

$$(\hat{p}_1^\uparrow, \hat{p}_1^\uparrow) := (p_1^\uparrow, \ell(N))$$

and choose

$$w_1 := \otimes^{(m-1)(n+1)} \nu \# \square^{\ell(N)-m\cdot(n+1)}.$$

If, for some $j_0 \in \{1, \dots, t\}$, $w_{j_0} = \varepsilon$, then we define

$$\begin{aligned} \text{config}_{i+1}(\kappa) &:= \perp, \\ \text{tape-config}_{j,i+1}(a\langle y_1 \rangle \cdots \langle y_t \rangle) &:= (\varepsilon, 2, 1), \end{aligned}$$

and $\alpha_{|K_{i+1}}(\kappa) := (a, e''_1, \dots, e''_t)$, where for all $j \in \{1, \dots, t\}$,

$$e''_j := \begin{cases} (\text{head-direction}_j, \text{true}) & \text{if } w_j = \varepsilon \\ (\text{head-direction}_j, \text{false}) & \text{otherwise.} \end{cases}$$

In what follows, we consider the case where $w_j \neq \varepsilon$, for all $j \in \{1, \dots, t\}$. We define

$$\begin{aligned} \text{config}_{i+1}(\kappa) &:= (q, p_1, \dots, p_t, p_{t+1}, \dots, p_{t+u}, \\ &\quad w_1, \dots, w_t, w_{t+1}, \dots, w_{t+u}), \end{aligned}$$

where q and $p_{t+1}, \dots, p_{t+u}, w_{t+1}, \dots, w_{t+u}$ are obtained from \hat{q} ,

p_1, \dots, p_t are obtained from a via $p_j := p_j^\uparrow$, for all $j \in \{1, \dots, t\}$, and w_1, \dots, w_t chosen as above.

Altogether, the description of the definition of $\text{config}_{i+1}(\kappa)$ is complete.

For defining $\alpha_{|K_{i+1}}(\kappa)$ and $\text{tape-config}_{j,i+1}(a\langle y_1 \rangle \cdots \langle y_t \rangle)$, consider the following: Let us start the Turing machine T with a configuration γ_0 that fits to $\text{config}_{i+1}(\kappa)$, that is, that can be obtained from $\text{config}_{i+1}(\kappa)$ by replacing each occurrence of the wildcard symbol \otimes by a symbol in Σ . Let $\gamma_0, \gamma_1, \gamma_2, \dots$ be the successive configurations of T when started in γ_0 , that is, $\gamma_{v+1} = S(\gamma_v)$, for all $v \geq 0$.

Then, there is a minimal $v > 0$ for which there exists a $j_0 \in \{1, \dots, t\}$ such that throughout the run $\gamma_0 \cdots \gamma_{v-1}$,

- (1) none of the heads $1, \dots, t$ changes its direction, and
- (2) none of the heads $j \in \{1, \dots, t\}$ crosses a border \hat{p}_j^\uparrow or \hat{p}_j^\downarrow ,

and one of the following cases applies:

Case 1: In the transition from γ_{v-1} to γ_v , head j_0 crosses one of the borders $\hat{p}_{j_0}^\uparrow$ or $\hat{p}_{j_0}^\downarrow$. That is, in γ_v , the j_0 -th head is either at position $\hat{p}_{j_0}^\uparrow - 1$ or at position $\hat{p}_{j_0}^\downarrow + 1$. (And none of the heads $j \in \{1, \dots, t\} \setminus \{j_0\}$ crosses a border or changes its direction.³)

Case 2: In the transition from γ_{v-1} to γ_v , head j_0 changes its direction, but does not cross one of the borders $\hat{p}_{j_0}^\uparrow$ or $\hat{p}_{j_0}^\downarrow$. (And none of the heads $j \in \{1, \dots, t\} \setminus \{j_0\}$ crosses a border or changes its direction.)

Case 3: γ_v is final, and $j_0 := t$.

³Recall that w.l.o.g. we assume that the Turing machine is normalised, cf. Definition 1.

In all three cases we let

$$(q'', p''_1, \dots, p''_{t+u}, w''_1, \dots, w''_{t+u}) := \gamma_v.$$

We choose

$$\hat{q}'' := (q'', p''_{t+1}, \dots, p''_{t+u}, w''_{t+1}, \dots, w''_{t+u})$$

and define

$$b := (\hat{q}'', \vec{p}''_1, \dots, \vec{p}''_t),$$

where

$$\vec{p}''_j = (p''_{j, \parallel}, p''_{j, \uparrow}, p''_{j, \downarrow}, \text{head-direction}''_j)$$

will be specified below.

Finally, we define

$$\alpha_{|K_{i+1}}(\kappa) := (b, e''_1, \dots, e''_t),$$

where, for every $j \in \{1, \dots, t\}$,

$$e''_j := (\text{head-direction}''_j, \text{move}''_j)$$

will be specified below.

Recall that $\kappa = (a, y_1, \dots, y_t) \in K_{i+1}$. For every $j \in \{1, \dots, t\}$ we define

$$\begin{aligned} \text{tape-config}_{j,i+1}(a\langle y_1 \rangle \cdots \langle y_t \rangle) &:= \begin{cases} (\otimes^{p_j^\parallel-1} w''_{j, p_j^\parallel} \cdots w''_{j, p_j^\parallel} \otimes^{\ell(N)-p_j^\parallel+1}, p_j^\parallel, p_j^\parallel) & \text{if } p_j^\parallel \leq p_j^\parallel \\ (\varepsilon, p_j^\parallel, p_j^\parallel) & \text{otherwise} \end{cases} \end{aligned}$$

where p_j^\parallel and p_j^\parallel are specified below.

For all $j \in \{1, \dots, t\} \setminus \{j_0\}$ we know (by the choice of ν and j_0) that throughout the TM computation $\gamma_0, \dots, \gamma_v$, head j neither changes its direction nor crosses one of the borders $\hat{p}_j^\uparrow, \hat{p}_j^\downarrow$. Consequently, we choose

$$\begin{aligned} \text{head-direction}''_j &:= \text{head-direction}_j \\ \text{move}''_j &:= \text{false} \\ p''_{j, \uparrow} &:= p''_{j, \uparrow} \\ p''_{j, \parallel} &:= \begin{cases} p''_{j, \uparrow} & \text{if } \text{head-direction}_j = +1 \\ \hat{p}_j^\parallel & \text{if } \text{head-direction}_j = -1 \end{cases} \\ p''_{j, \downarrow} &:= \begin{cases} \hat{p}_j^\parallel & \text{if } \text{head-direction}_j = +1 \\ p''_{j, \uparrow} & \text{if } \text{head-direction}_j = -1 \end{cases} \\ p_j^\parallel &:= \begin{cases} \hat{p}_j^\parallel & \text{if } \text{head-direction}_j = +1 \\ p''_{j, \uparrow} + 1 & \text{if } \text{head-direction}_j = -1 \end{cases} \\ p_j^\parallel &:= \begin{cases} p''_{j, \uparrow} - 1 & \text{if } \text{head-direction}_j = +1 \\ p''_{j, \downarrow} & \text{if } \text{head-direction}_j = -1 \end{cases} \end{aligned}$$

To specify

$$\text{head-direction}''_{j_0}, \text{move}''_{j_0}, p''_{j_0, \parallel}, p''_{j_0, \uparrow}, p''_{j_0, \downarrow}, p_{j_0}^\parallel, \text{ and } p_{j_0}^\parallel,$$

we distinguish between the three cases listed above:

ad Case 1: In this case, head j_0 crosses one of the borders $\hat{p}_{j_0}^\uparrow$ or $\hat{p}_{j_0}^\downarrow$ in the transition from γ_{v-1} to γ_v (that is, p''_{j_0} is either $\hat{p}_{j_0}^\uparrow + 1$ or

$\hat{p}_{j_0}^{\parallel} - 1$). We choose

$$\begin{aligned} (p''_{j_0}, p''_{j_0}, p''_{j_0}) &:= (\ominus, p''_{j_0}, \ominus) \\ (p''_{j_0}, p''_{j_0}) &:= (\hat{p}_{j_0}^{\parallel}, \hat{p}_{j_0}^{\parallel}) \\ \text{move}_{j_0}'' &:= \text{true} \\ \text{head-direction}_{j_0}'' &:= \begin{cases} +1 & \text{if } p''_{j_0} = \hat{p}_{j_0}^{\parallel} + 1 \\ -1 & \text{otherwise.} \end{cases} \end{aligned}$$

ad Case 2: In this case, head j_0 changes its direction, but does not cross one of the borders $\hat{p}_{j_0}^{\parallel}$ or $\hat{p}_{j_0}^{\parallel}$. We only consider the case where the direction of head j_0 changes from $+1$ to -1 (the other case is symmetric).

We choose

$$\begin{aligned} (\text{head-direction}_{j_0}'', \text{move}_{j_0}'') &:= (-1, \text{false}) \\ (p''_{j_0}, p''_{j_0}, p''_{j_0}) &:= (\hat{p}_{j_0}^{\parallel}, p''_{j_0}, p''_{j_0} + 1) \\ (p''_{j_0}, p''_{j_0}) &:= (p''_{j_0} + 2, \hat{p}_{j_0}^{\parallel}) \end{aligned}$$

Note that here we might have $p''_{j_0} + 1 = \hat{p}_{j_0}^{\parallel}$. In this case, by the above definition, we obtain $p''_{j_0} = \hat{p}_{j_0}^{\parallel} + 1$.

ad Case 3: In this case we have $j_0 = t$ and γ_V is *final*, that is, q'' is a final state of the TM T . Therefore, b is a *final* state of the LMM. As M immediately stops in such a state, we may choose p''_t, p''_t, p''_t arbitrarily and may let $\text{head-direction}_t'' := \text{head-direction}_t$. We let $\text{move}_t'' := \text{true}$ — due to this final head movement on M 's output list t , the information on M 's current state a and the symbols currently read on the t lists is saved to the current cell of list t . This needs to be done in order to ensure that M 's output (which is obtained as explained in Step 4 below) is indeed exactly the output that the Turing machine would generate.

We choose

$$(p''_{j_0}, p''_{j_0}) := (\hat{p}_{j_0}^{\parallel}, \hat{p}_{j_0}^{\parallel}).$$

This completes the induction step.

Finally, we are ready to fix M 's state set A and transition function α as follows:

$$\begin{aligned} A &:= \bigcup_{i \geq 0} A_i \\ K &:= \bigcup_{i \geq 0} K_i \\ \alpha &:= \bigcup_{i \geq 0} \alpha_{K_i} \end{aligned}$$

Note that

1. α is well-defined, because α_{K_i} and $\alpha_{K_{i'}}$ operate identical on all elements in $K_i \cap K_{i'}$ (for all $i, i' \geq 0$).
2. K consists of all situations $(a, y_1, \dots, y_t) \in (A \setminus B) \times (\mathbb{A}^*)^t$ that may occur in runs of M .
3. α remains undefined on elements (a, y_1, \dots, y_t) in $(A \setminus B) \times (\mathbb{A}^*)^t$ that do *not* belong to K . This is fine, because such a situation (a, y_1, \dots, y_t) can never occur in an actual run of M .

This completes Step 3. \dashv

Step 4: Definition of M 's output function $\omega : \mathbb{A}^* \rightarrow (\Sigma \setminus \{\square\})^*$.

We define $\omega(\langle \rangle) := \varepsilon$.

For every $v \in I$ we let $\omega(\langle v \rangle) := v$.

For each string y that belongs to L_i , for some $i \geq 0$, we let

$$(w', p^{\parallel}, p^{\parallel}) := \text{tape-config}_{t,i}(y),$$

choose w'' to be the string obtained from w' by deleting all \otimes -symbols and all \square -symbols, and let

$$\omega(y) := w''.$$

(This is well-defined, because $\text{tape-config}_{j,i}$ and $\text{tape-config}_{j,i'}$ operate identical on all elements in $L_i \cap L_{i'}$.)

On all strings y which, for every $i \geq 0$, do not belong to L_i , the output function can remain undefined (or set to any arbitrary value, e.g., $\omega(y) := \varepsilon$), because such strings will never occur as list-entries throughout a computation of M .

This completes Step 4. \dashv

Step 5: Given arbitrary $(v_1, \dots, v_m) \in I^m$, we have

$$f_M(v_1, \dots, v_m) = f_T(v_1 \# \dots \# v_m).$$

Proof: By a straightforward (but tedious) induction along the definition of M 's transition function and output function given in Steps 3 and 4. This completes Step 5. \dashv

Altogether, the proof of Lemma 14 is complete. \square

6. A Lower Bound for Sorting on List Machines

Let $I \subseteq \{0, 1\}^*$. We say that an LM $M = (t, m, I, A, a_0, \alpha, B, \omega)$ solves the φ -sorting problem for m inputs from I if for all inputs $(v_{\varphi(1)}, \dots, v_{\varphi(m)})$, where $v_1 < \dots < v_m \in I$, the output of M is $v_1 \# \dots \# v_m \#$.

Lemma 15 (Lower Bound for Sorting on LM).

Let $k, m, n, r, t \in \mathbb{N}$ such that m is a power of 2 and

$$t \geq 2, \quad m \geq 4 \cdot t^{2r}, \quad k \geq m + 2, \quad n > 3m^3 \cdot \log k.$$

Then there is no r -bounded LM M with t lists and at most k states that solves the sorting problem for m inputs from $\{0, 1\}^n$.

Proof: Let $M = (t, m, I, A, a_0, \alpha, B, \omega)$ be an r -bounded LM with $|A| \leq k$. Let $I = \{0, 1\}^n$, and suppose for contradiction that M solves the sorting problem for m inputs from I .

The *total list length* of a configuration of M is the sum of the lengths of all lists in that configuration. Observe that the total list length never decreases during a computation.

Claim 1 For all $(v_1, \dots, v_m) \in I^m$, the total list length of all configurations of the run of M on input (v_1, \dots, v_m) is bounded by

$$(t+1)^r \cdot m.$$

Proof: Let γ be a configuration of total list length ℓ . Then the total list length of the successor configuration of γ is at most $\ell + t$ if a head moves or changes its direction in the transition from γ to its successor, and it remains ℓ otherwise. Now suppose γ' is a configuration that can be reached from γ without changing the direction of any head. Then γ' is reached from γ with at most $\ell - t$ head movements, because a head can move at most $i - 1$ times on a list of length i . Thus the total list length of γ' is at most

$$\ell + t \cdot (\ell - t) \tag{7}$$

The total list length of the initial configuration is $m+t-1$. A simple induction based on (7) shows that the total list length of a configuration that occurs before the i th change of a head direction is at most

$$(t+1)^i \cdot m.$$

This proves Claim 1. \dashv

The *cell size* of a configuration is the maximum length of the entries of the cells occurring in the configuration (remember that the cell entries are strings over \mathbb{A}).

Claim 2 For all $(v_1, \dots, v_m) \in I^m$, the cell size of all configurations of the run of M on input (v_1, \dots, v_m) is bounded by

$$8 \cdot t^r.$$

Proof: Let γ be a configuration of cell size c . Then the cell size of all configurations that can be reached from γ without changing the direction of any head is at most

$$1+t \cdot (c+2)$$

The cell size of the initial configuration is 3. A simple induction shows that the total cell size of any configuration that occurs before the i th change of a head direction is at most

$$5t^i + \sum_{j=1}^{i-1} 3t^j + 1 \leq 8 \cdot t^i.$$

This proves Claim 2. \dashv

Recall that for a sequence $(v_1, \dots, v_m) \in I^m$ the number $\sigma(\vec{v})$ is the length of the longest (increasingly or decreasingly) sorted subsequence of \vec{v} .

Let $\gamma = (a, \vec{p}, \vec{d}, X)$ be a configuration of M , where

$$X = (\vec{x}_1, \dots, \vec{x}_t)^\top \text{ with } \vec{x}_i = (x_{i,1}, \dots, x_{i,m_i}) \in (\mathbb{A}^*)^{m_i},$$

for some $m_i \geq 1$. We say that a sequence $(v_1, \dots, v_\ell) \in I^\ell$ occurs in γ if there is an $i \leq t$ and $1 \leq j_1 \leq \dots \leq j_\ell \leq m_i$ such that for $1 \leq p \leq \ell$ the number v_p occurs in the string x_{i,j_p} . We let $\sigma(\gamma)$ be maximum ℓ such that there is a sequence $(v_1, \dots, v_\ell) \in I^\ell$ with $v_1 < \dots < v_\ell$ or $v_1 > \dots > v_\ell$ that occurs in γ .

Claim 3 Let $\vec{v} = (v_1, \dots, v_m) \in I^m$. Then for all configurations γ of the run of M on input (v_1, \dots, v_m) we have

$$\sigma(\gamma) \leq t^r \cdot \sigma(\vec{v}).$$

Proof: Let γ be a configuration with $\sigma(\gamma) = \ell$. Then for all configurations γ' that can be reached from γ without changing the direction of any head we have

$$\sigma(\gamma') \leq t \cdot \ell. \quad (8)$$

To see this, suppose for contradiction that a sequence

$$\vec{v} = (v_1, \dots, v_{\ell+t+1}) \in I^{\ell+t+1}$$

with $v_1 < \dots < v_{\ell+t+1}$ or $v_1 > \dots > v_{\ell+t+1}$ occurs in γ' . Each occurrence of a v_i in γ' can be traced back to the occurrence v_i on some list in γ . Thus there is a list j such that at least $\ell+1$ of these occurrences of v_i s in γ are on list j , say, in positions $p_1 \leq \dots \leq p_{\ell+1}$. But this means that $\sigma(\gamma) \geq \ell+1$, which is a contradiction.

A simple induction based on (8) proves the statement of the claim. \dashv

We subdivide $I = \{0, \dots, 2^n - 1\}$ into m consecutive intervals I_1, \dots, I_m each of length $2^n/m$. Remember that 2^n is divisible by m because m is a power of 2. From now on, we only consider inputs

$$\vec{v} = (v_1, \dots, v_m) \in \mathcal{I} := I_{\varphi(1)} \times \dots \times I_{\varphi(m)}.$$

Note that

$$|\mathcal{I}| = \left(\frac{2^n}{m}\right)^m.$$

Let $\vec{v} = (v_1, \dots, v_m) \in \mathcal{I}$, and let $\gamma = (a, \vec{p}, \vec{d}, X)$ be the final configuration of the run of M on input \vec{v} . Let $\vec{x}_t = (x_{t,1}, \dots, x_{t,m_t}) \in (\Sigma^*)^{m_t}$ for some $m_t \geq 1$ be the content of the last list in γ . To determine the output, the function ω is applied to all $x_{t,j}$. The output is

$$v_{\varphi^{-1}(1)} \# v_{\varphi^{-1}(2)} \# \dots \# v_{\varphi^{-1}(m)} \#.$$

For $1 \leq i \leq m$, let $e_i \in \{1, \dots, m_t\}$ be the position where ω produces the i th '#', that is, where the output of $v_{\varphi^{-1}(i)}$ is completed. Let $e_0 := 1$ and, for $1 \leq i \leq m$,

$$E_i := \{e_{i-1}, \dots, e_i\}.$$

We call E_i the *output interval* for $v_{\varphi^{-1}(i)}$ or, for short, the output interval for i .

Claim 4 There is some index $i_0 \in \{1, \dots, m\}$ such that $v_{\varphi^{-1}(i_0)}$ does not occur in $x_{t,j}$ for any $j \in E_{i_0}$.

Proof: By Claim 3, there are at most $t^r \cdot (2\sqrt{m} - 1)$ indices $i \in \{1, \dots, m\}$ such that $v_{\varphi^{-1}(i)}$ occurs in $x_{t,j}$ for some $j \in E_i$. Since $\sqrt{m} \geq 2t^r$, we have $t^r \cdot (2\sqrt{m} - 1) < m$, and the claim follows. \dashv

Let $j \leq m_t$. Note that all $v \in I$ that appear as letters in the strings $x_{t,j}$ are contained in $\{v_1, \dots, v_m\}$. The *index string* of $x_{t,j}$ is the string $\text{ind}(x_{t,j})$ obtained from $x_{t,j}$ by replacing each occurrence of v_i by the index i , for $1 \leq i \leq m$. The *skeleton generated by* $\vec{v} = (v_1, \dots, v_m)$ is the tuple

$$\text{skel}(\vec{v}) = (\text{ind}(x_{t,1}), \dots, \text{ind}(x_{t,m_t}), e_1, \dots, e_m).$$

Note that the index strings are strings over the alphabet $\{1, \dots, m\} \cup A \cup \{\#\}$. By Claim 2, the length of these strings is at most $8t^r$. Thus there are at most

$$(m+k+2)^{8t^r}$$

different index strings. Since the e_i are bounded by the length m_t of the last list and since $m_t \leq (t+1)^r \cdot m$ by Claim 1, there are at most

$$\begin{aligned} & (m+k+2)^{8t^r \cdot m_t} \cdot m_t^m \\ & \leq (m+k+2)^{8t^r \cdot (t+1)^r m} \cdot ((t+1)^r \cdot m)^m \\ & \leq (2 \cdot k)^{m^3} \cdot m^{2m} \end{aligned}$$

$$\text{(because } k \geq m+2, m \geq 4t^{2r}, \text{ and } t \geq 2)$$

skeletons.

Let K be a skeleton that is generated by a maximum number of inputs $\vec{v} \in \mathcal{I}$, and let \mathcal{I}_1 be the set of all inputs that generate K , that is,

$$\mathcal{I}_1 = \{\vec{v} \in \mathcal{I} \mid \text{skel}(\vec{v}) = K\}.$$

Then

$$|\mathcal{I}_1| \geq \frac{|\mathcal{I}|}{(2 \cdot k)^{m^3} \cdot m^{2m}}.$$

Since the output intervals only depend on the skeleton, by Claim 4 there is some index $i_0 \in \{1, \dots, m\}$ such that for all $\vec{v} \in \mathcal{I}_1$ the number $v_{\varphi^{-1}(i_0)}$ does not occur in $x_{t,j}$ for any $j \in E_{i_0}$. Without loss of generality we may assume that $i_0 = m$. Choose $v_1 \in I_1, \dots, v_{m-1} \in I_{m-1}$ such that the tuple (v_1, \dots, v_{m-1}) has the maximum number of extensions

$$(v_1, \dots, v_m) \in \mathcal{I}_1.$$

Then the number of v_m such that $(v_1, \dots, v_m) \in \mathcal{S}_1$ is at least

$$\begin{aligned} \frac{|\mathcal{S}_1|}{\left(\frac{2^n}{m}\right)^{(m-1)}} &\geq \frac{\left(\frac{2^n}{m}\right)^m}{(2 \cdot k)^{m^3} \cdot m^{2m} \cdot \left(\frac{2^n}{m}\right)^{(m-1)}} \\ &= \frac{2^n}{(2 \cdot k)^{m^3} \cdot m^{2m+1}} \\ &\geq 2. \end{aligned}$$

Thus there are $v_m \neq v'_m$ such that both $\vec{v} = (v_1, \dots, v_m) \in \mathcal{S}_1$ and $\vec{v}' = (v_1, \dots, v_{m-1}, v'_m) \in \mathcal{S}_1$ have the same skeleton K . Moreover, if $\vec{x}_t = (x_{t,1}, \dots, x_{t,m_t})$ and $\vec{x}'_t = (x'_{t,1}, \dots, x'_{t,m_t})$ are the last lists of the final configuration of M on input \vec{v} and \vec{v}' , respectively, then for all $j \in E_m$ we have $x_{t,j} = x'_{t,j}$. Thus ω generates the same output on both inputs, which is impossible (since we assume that M correctly sorts its input). \square

7. Proof of the Theorem 7

Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ such that

$$r(N) = o(\log N) \quad \text{and} \quad s(N) = o\left(\sqrt[5]{N}/r(N)\right),$$

and let $t \in \mathbb{N}$. Without loss of generality we may assume that $t \geq 2$. Suppose for contradiction that T is an (r, s, t) -bounded Turing machine for φ -SORT.

Let d be the constant introduced in Lemma 14 (the Simulation Lemma). Let m be a sufficiently large power of 2 such that

$$m \geq 4t^{2r(m^5+m)} \quad \text{and} \quad (9)$$

$$m^4 \geq 3m^3 \cdot d \cdot t^2 \cdot (r(m^5+m) \cdot s(m^5+m) + \log(m^5+m)). \quad (10)$$

Such an m exists because

$$r(N) = o(\log N) \quad (\text{for (9)}) \quad \text{and} \quad r(N) \cdot s(N) = o\left(\sqrt[5]{N}\right) \quad (\text{for (10)}).$$

Let $n := m^4$ and $I := \{0, \dots, 2^n - 1\}$. By Lemma 14, there exists an $(m \cdot (n+1))$ -bounded LM

$$M = (t, m, I, A, a_0, \alpha, B, \omega)$$

with

$$k = 2^{d \cdot t^2 \cdot r(m \cdot (n+1)) \cdot s(m \cdot (n+1)) + 3t \cdot \log(m \cdot (n+1))}$$

states that simulates T on inputs from I^m and thus solves the sorting problem for such inputs.

We clearly have $k \geq m + 2$. By (9), we have

$$m \geq 4t^{2r(m^5+m)} = 4t^{2r(m \cdot (n+1))}.$$

By (10), we have

$$\begin{aligned} n &= m^4 \\ &\geq 3m^3 \cdot d \cdot t^2 \cdot (r(m^5+m) \cdot s(m^5+m) + \log(m^5+m)) \\ &= 3m^3 \cdot d \cdot t^2 \cdot (r(m(n+1)) \cdot s(m(n+1)) + \log(m(n+1))) \\ &= 3m^3 \cdot \log k. \end{aligned}$$

Thus by Lemma 15, the machine M cannot solve the sorting problem for inputs from I^m . This is a contradiction.

Altogether, the proof of Theorem 7 is complete. \square

8. References

- [1] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 540–549, 2004.
- [2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.
- [3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems*, pages 1–16, 2002.
- [4] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. Springer-Verlag, 2nd edition, 1995.
- [5] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the memory requirements of XPath evaluation over XML streams. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems*, pages 177–188, 2004.
- [6] J. Chen and C.-K. Yap. Reversal complexity. *SIAM Journal on Computing*, 20(4):622–638, 1991.
- [7] M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. Manuscript, 2005.
- [8] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *External memory algorithms*, volume 50, pages 107–118. DIMACS Series In Discrete Mathematics And Theoretical Computer Science, 1999.
- [9] C. Koch. Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree automata-based approach. In *Proceedings of 29th Conference on Very Large Data Bases*, pages 249–260, 2003.
- [10] U. Meyer, P. Sanders, and J. F. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2832 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.
- [11] J. J. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
- [12] S. Muthukrishnan. Data streams: algorithms and applications. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 413–413, 2003.
- [13] J. F. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33:209–271, 2001.
- [14] K. Wagner and G. Wechsung. *Computational Complexity*. VEB Deutscher Verlag der Wissenschaften, 1986.