SIAM J. COMPUT.
Vol. 21, No. 4, pp. 697–712, August 1992

THE POWER OF THE QUEUE*

MING LI†, LUC LONGPRɇ, AND PAUL VITÁNYI§

Abstract. Queues, stacks, and tapes are basic concepts that have direct applications in compiler design and the general design of algorithms. Whereas stacks (pushdown store or last-in–first-out storage) have been thoroughly investigated and are well understood, this is much less the case for queues (first-in–first-out storage). In this paper a comprehensive study comparing queues to stacks and tapes (off-line and with a one-way input tape) is presented. The techniques used rely on Kolmogorov complexity. In particular, one queue and one tape (or stack) are incomparable:

(1) Simulating one stack (and hence one tape) by one queue requires $\Omega(n^{4/3}/\log n)$ time in both the deterministic and the nondeterministic cases. A corollary of this lower bound states that for this model of one-queue machines, nondeterministic linear time is not closed under complement.

(2) Simulating one queue by one tape requires $\Omega(n^2)$ time in the deterministic case and requires $\Omega(n^{4/3}/(\log n)^{2/3})$ in the nondeterministic case.

The paper further compares the relative power between different numbers of queues:

(3) Simulating two queues (or two tapes) by one queue requires $\Omega(n^2)$ time in the deterministic case, and $\Omega(n^2/(\log^2 n \log\log n))$ in the nondeterministic case. The deterministic bound is tight. The nondeterministic one is almost tight. The upper bounds for queues are also obtained.

Key words. abstract storage unit, multi-queue machines, multi-tape machines, on-line simulation, lower bounds, upper bounds, Kolmogorov complexity

AMS(MOS) subject classifications. 68Q05, 68Q30

1. Introduction. It has been known for over 20 years that all multi-tape Turing machines can be simulated on line by two-tape Turing machines in time $O(n \log n)$ [HS66] and by one-tape Turing machines in time $O(n^2)$. Since then, many other models of computation have been introduced and compared [Aan74], [DGPR84], [HS65], [HS66], [HU79], [KOS79], [LS81], [MSS87], [PSS81], [Pau82], [Vit85]. In addition to different storage mechanisms, real-time, on-line, and off-line machines have been studied. An on-line simulation essentially simulates step-by-step each move of the simulated machine. In this paper we consider off-line machines, for which an answer is given only after the entire input has been read. There is no need to simulate the moves of the machine; it only matters that the right answer is given. We also use the one-way input convention, which states that the machine has a one-way input tape. As usual, the machines have a finite control and access to some storage.

The relative power of stacks and tapes is more or less well known.[1] For example, for the nondeterministic case, we know that 1 stack < 1 tape < 2 stacks < 3 stacks = k

*Received by the editors July 7, 1989; accepted for publication (in revised form) July 24, 1991. Part of this paper appeared in a preliminary version in [LL V86].

†Present address, Department of Computer Science, University of Waterloo, Ontario, Canada N2L 361. Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3. This research was performed while the author was at Ohio State University and Harvard University. This work was supported by the National Science Foundation under grant DCR-8606366 by the Office of Naval Research under grant N00014-85-K-0445.

‡College of Computer Science, Northeastern University, Boston, Massachusetts 02115. Part of this research was performed while the author was a visiting faculty member in the Computer Science Department at the University of Washington.

§Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, the Netherlands. The work was performed in part at the Laboratory for Computer Science, Massachusetts Institute of Technology, and supported in part by the Office of Naval Research under contract N00014-85-K-0168, by the U.S. Army Research Office under contract DAAG29-84-K-0058, by the National Science Foundation under grant DCR-83-02391, and by the Defense Advanced Research Projects Agency under contract N00014-83-K-0125.

[1]Throughout the paper, *stack* and *pushdown store* are used synonymously. The basic operations are *push* and *pop*, and only the top of the store is accessible to the machine.

stacks = k tapes, where $A < B$ means that B can simulate A in linear time, but A cannot simulate B in linear time. In most of the cases, close lower and upper bounds are known for the simulation [Maa85], [Li85b], [Li88], [LV88], [Vit84b].

In this paper we give a complete characterization of (off-line, one-way input) queue machines. The main theorems show that one-queue machines are incomparable to one-stack or one-tape machines, both deterministically and nondeterministically. One corollary of our nondeterministic lower bound is that for our model of one-queue machines, nondeterministic linear time is not closed under complement. We also compare the relative power of machines having different numbers of queues. The current knowledge of upper and lower bounds for the simulation between queues and tapes is roughly summarized in Figs. 1, 2, and 3. Figure 1 contains results that were previously known. The results of Fig. 2 are covered in §2. Notice that all the bounds in Fig. 2 are valid also for simulating one stack or two stacks. The results of Fig. 3 are covered in §3.

	deterministic	nondeterministic
upper bound	$O(n^2)$ (in [HS65])	$O(n^{3/2}\sqrt{\log n})$ (in [Li88])
lower bound	$\Omega(n^2)$ (in [LV88])	$\Omega(n^{4/3}/\log^{2/3} n)$ (in [LV88] or [Li85a])

FIG. 1. *Simulating one queue by one tape.*

	deterministic	nondeterministic
upper bound	$O(n^2)$	$O(n^2)$
lower bound	$\Omega(n^{4/3}/\log n)$	$\Omega(n^{4/3}/\log n)$

FIG. 2. *Simulating one tape, one stack, or two stacks, by one queue.*

	deterministic	nondeterministic
upper bound	$O(n^2)$	$O(n^2)$
lower bound	$\Omega(n^2)$	$\Omega(n^2/\log^2 n \log\log n)$

FIG. 3. *Simulating two queues by one queue.*

We use Kolmogorov complexity techniques [Sol64], [Kol65], [Cha77], together with some new techniques to enable us to deal with queues to prove the theorems. The Kolmogorov complexity $K(x)$ of a string x is the length of the shortest program printing the string x. By a simple counting argument, we know that for at least half of the strings x of each length, $K(x) \geq |x|$. These strings are called *incompressible* or *K random*. For completeness, we recall the notions of Kolmogorov complexity of binary strings and those of self-delimiting descriptions (see, e.g., [PSS81], [LV88]). Fix an effective coding C of all Turing machines as binary strings, such that no code is a prefix of any other code. Denote the code of Turing machine M by $C(M)$. The Kolmogorov complexity with respect to C of a binary string x, denoted $K_C(x)$, is the length of the smallest binary string $C(T)y$

such that T started on input y halts with output x. The crucial fact one uses is that for any fixed effective enumerations C and D, for all x $|K_C(x) - K_D(x)| < c$, with c a constant depending only on C and D (but not on x). Thus, up to an additive constant, the Kolmogorov complexity is independent of the particular effective enumeration chosen, which allows us to drop the subscript. With some abuse of notation, the sequel equalities and inequalities involving Kolmogorov complexity will always be assumed to hold up to an additive constant only. To be able to differentiate between parts of y such that T is able to use different parts for different purposes (can compute an r-ary function), we need the notion of self-delimiting descriptions. If $a = a_1 a_2 \cdots a_n$ is a string of 0's and 1's, then $a_1 0 a_2 0 \cdots 0 a_n 1$ is a self-delimiting description of twice the original length. More efficiently, if $b = b_1 \cdots b_m$ is the length of a in binary, then the self-delimiting description of b concatenated with a is also a self-delimiting description of a, this time of length $n + 2\log n$ instead of $2n$. For example, 1000011101 is the self-delimiting version of 1101.

2. The queue machine model. We will first describe more formally the model and the notation we use for queue machines.

A queue machine has a one-way input tape with the input head initially positioned at the beginning of the input string. For storage it uses a queue. The rear of the queue contains the first symbols pushed (and not popped). The front contains the last symbols pushed. The machine can access only one symbol at the rear of the queue.

One step of the queue machine consists of all the following. According to the old state and the contents of the cells scanned on the input and on the queue, the machine

1. reads an empty or nonempty symbol from the input,
2. pops an empty or nonempty symbol from the queue,
3. pushes an empty or nonempty symbol on the queue,
4. changes state.

Let h_{in} be the read-only head on the one-way input tape. We identify the queue with a tape with two heads h_r and h_w. The queue machine is implemented as follows on the tape representation. The initial state and the state transitions are the same. The head h_r is a read-only, one-way head on the tape. The head h_w is a write-only, one-way head on the tape. One step of the queue machine is implemented as follows:

1. the input head h_{in} behaves the same way as on the original queue machine;
2. if a nonempty symbol is written (pushed) on the queue, then h_w writes the symbol in the currently scanned cell and moves to the right adjacent cell (if an empty symbol is written, then h_w does not move);
3. if a nonempty symbol is read (popped) from the queue, then h_r moves to the right adjacent cell (if an empty symbol is read, then h_r does not move);
4. the change of state occurs as in the original machine.

Without loss of generality, we assume that the machine uses a binary alphabet on the queue and accepts by empty queue.

Let $h_k(t)$ denote the position of head $k \in \{in, r, w\}$ at time t on its respective tape. Let $c_1, c_2, \cdots, c_n$ be the individual cells on the input tape. Let $d_1, d_2, \cdots$ be the individual cells on the queue. We sometimes use $h_k(t)$ to denote the cell at that position.

The contents of the tape from $h_r(t)$ through $h_w(t) - 1$ inclusive is called the *actual queue* at time t, or *Queue*(t). The length of *Queue*(t), denoted $|$*Queue*$(t)|$, is $h_w(t) - h_r(t)$. We say that cells d_i and d_j are *contiguous* on *Queue*(t) if $h_r(t) < j < h_w(t)$ and $j = i+1$, or if $i + 1 = h_w(t)$ and $j = h_r(t)$ (that is, the cells at opposite ends of the queue are also considered contiguous).

3. Simulating one tape by one queue.

3.1. Upper bound. Our upper bound is straightforward. It is for simulating any fixed number of stacks, but since two stacks can simulate one tape in real time, our upper bound applies to tapes as well.

THEOREM 3.1. *For any fixed k, one queue can simulate k stacks in $O(n^2)$ time for both deterministic and nondeterministic machines.*

Proof. Simulate the k stacks by coding them sequentially onto the queue such that the top of each stack comes first. In front of each stack top, put a marker to indicate the separation between the stacks.

Each operation (push or pop on one stack) can be done in $O(n)$ time by scanning the entire queue and performing the local transformation after the appropriate marker. Scanning is done by successively transferring the symbols from one end of the queue to the other end. The total time is then in $O(n^2)$. This simulation can be made for deterministic or nondeterministic machines. □

3.2. Lower bound. In this section, we show that it takes $\Omega(n^{4/3}/\log n)$ time for a nondeterministic one-queue machine with a one-way input to recognize the language $L = \{w\#w^R : w \in \{0,1\}^*\}$. The proof also provides the same lower bound for the set of palindromes.

Because L can be recognized in linear time by a deterministic one-stack machine (a deterministic pushdown automaton), we can conclude that it takes $\Omega(n^{4/3}/\log n)$ time for a nondeterministic one-queue machine to simulate a deterministic one-stack machine.

The intuition behind the proof is that while the queue machine reads w, it has to store all the information in some sequential way on the queue. It turns out to be impossible to check the stored form of w for correspondence with w^R while the latter string is read from the input tape, so w^R must be stored in some sequential way as well. Using crossing sequence arguments, we show that whatever way the information is stored, the machine is forced to scan the queue many times. This repeated scanning then implies the lower bound on simulation time.

THEOREM 3.2. *A nondeterministic one-queue machine with a one-way input tape requires $\Omega(n^{4/3}/\log n)$ time to accept the language $L = \{w\#w^R : w \in \{0,1\}^*\}$.*[2]

Remark. This holds both for the worst-case time and the average time, when the average is taken over all strings in L. Notice that the straightforward algorithm to accept L with a queue has a linear average time when the average is taken over all strings, since most strings can be discovered not to be in the language quickly.

Proof. Let Q be a one-queue machine that accepts L. We show that Q will make $\Omega(n^{4/3}/\log n)$ steps before accepting any string $x\#x^R$ for incompressible strings x of size n. Since the size of the input is $2n+1$, this will provide the wanted lower bound for L. Since at least half the strings of each length are incompressible, this also provides the claimed average time lower bound.

Let x be an incompressible string of length n. We separate x into two blocks: $x = x_0\tilde{x}$, with $|x_0| = \lfloor n/2 \rfloor$. Let $m = \lfloor n^{1/3}/4 \rfloor$ and $p = \lfloor n/2m \rfloor$. We further separate $\tilde{x}$ into m blocks of size p or $p+1$: $\tilde{x} = x_1 x_2 \cdots x_m$.

---

[2]Here we use the stronger version of Ω where $T(n) \in \Omega(f(n))$ if there are positive constants c and n_0 such that for all $n \geq n_0, T(n) \geq cf(n)$. Notice that there is no string of even length in the language. To be strict, we show that the time is in $\Omega(n^{4/3}/\log n \, : \, n \text{ is odd})$. With a slightly modified language, $\{x\#x^R\} \cup \{x\#\#x^R\}$, we could prove it for all n.

We look at any fixed accepting computation of the machine on input $x\#x^R$. Let t_j be the time step when h_{in} enters the block x_j. Let t'_j be the time step when h_{in} enters the block $x_j{}^R$. If z is a substring of x, then z' denotes the corresponding substring of x^R ($= x'$).

CLAIM 3.3. *If* $t_1 \le t \le t'_0$, *then* $|Queue(t)| \ge n/2 - O(\log n)$.

Proof. Let $t_1 \le t \le t'_0$. Let $|Queue(t)| = s$. The string x can be reconstructed by using the following information: a description of this discussion and of Q in $O(1)$ bits, the string $Queue(t)$ of length s, the string $\tilde{x}$ of length $\lceil n/2 \rceil$, the state $q(t)$ of the machine in $O(1)$ bits, and $h_{in}(t)$ in $\le \log n + 2$ bits. All items are encoded as self-delimiting strings. The total number of bits required for this description is $s + n/2 + O(\log n)$.

To reconstruct x from this information, run Q with all possible candidate strings y substituted for x_0. Single out the strings y for which there is a time step for which $Queue(t)$, $h_{in}(t)$, and $q(t)$ correspond. Among those y, the machine should accept only if $y = x_0$; otherwise, it would accept the string $x_0\tilde{x}\#\tilde{x}^R y^R \notin L$ by behaving like the computation on $x\#x^R$ up to time t and like the computation on $y\tilde{x}\#\tilde{x}^R y^R$ after time t.

Because x is incompressible, we know that $K(x) \ge n$, so it must be that our program reconstructing x has size $\ge n$. Thus, we have $s + n/2 + O(\log n) \ge n$, from which the claim follows. □

The machine Q needs to remember what it reads on the input and code it in some way on the queue or compare it with what is already on the queue. What can be written on the queue is determined by the current state, the input, and the rear of the queue. The input can be compared with the rear of the queue. These intuitive ideas motivate the following definitions of *influence*.

DEFINITION 3.4. An input cell c_i *directly influences* a cell d_j if h_{in} scans c_i while h_w writes in d_j (that is, $h_w(t) = j$, $h_w(t+1) = j+1$, and $h_{in}(t) = i$).

DEFINITION 3.5. A cell d_i *backward influences* a cell d_j if h_w is or moves onto d_i when h_r moves onto d_j (that is, $h_r(t-1) = j-1$, $h_r(t) = j$ and $h_w(t) = i$).

DEFINITION 3.6. A cell d_i *forward influences* a cell d_j if h_r scans d_i while h_w writes in d_j (that is, $h_w(t) = j$, $h_w(t+1) = j+1$ and $h_r(t) = i$).

(See Fig. 4 for an example of direct influence and Fig. 5 for an example of backward and forward influence.)

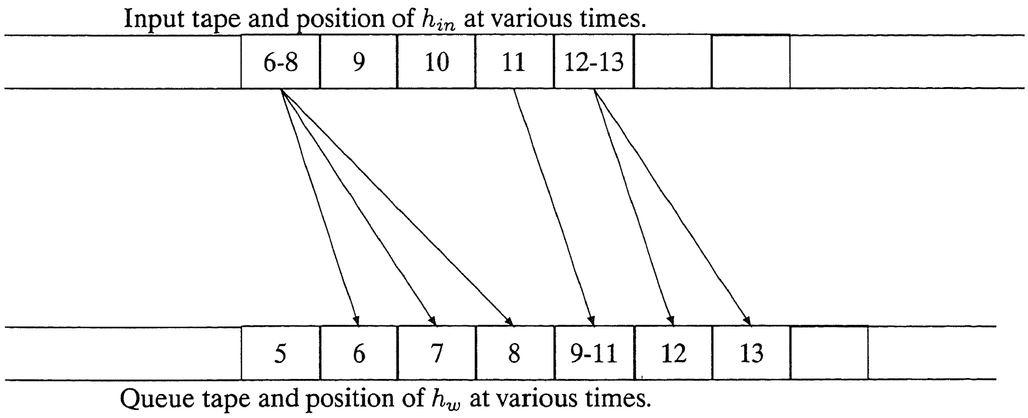

FIG. 4. *Direct influence relation.*

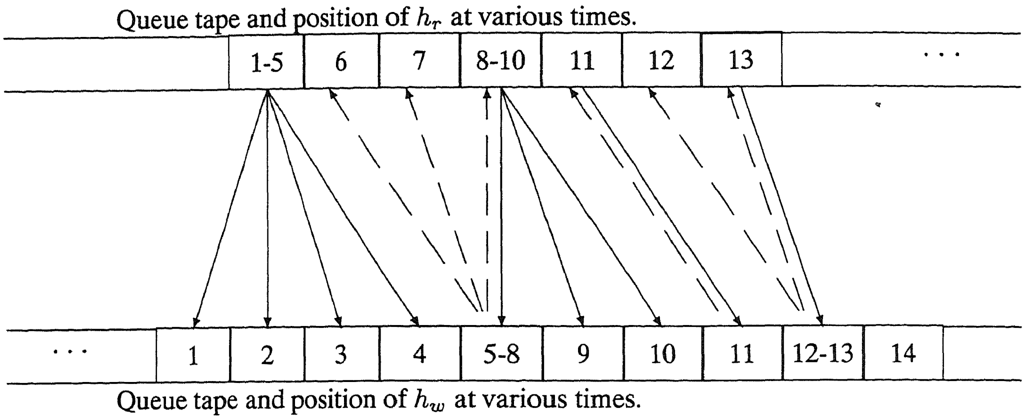

FIG. 5. *Forward (⟶) and backward (— — ⇢) influence relation.*

DEFINITION 3.7. The *influence relation* among the tape cells is the transitive closure of the forward influence relation union the transitive closure of the backward influence relation. In other words, a cell d_i influences a cell d_j if there is a chain of forward influences or a chain of backward influences from d_i to d_j.

An input cell c_i influences a cell d_j if c_i directly influences a tape cell that influences d_j.

A block of cells influences a cell if and only if at least one of the cells in the block influences it. A block of cells is influenced by a block of cells if at least one cell of the first block is influenced by the second block. Figure 6 illustrates the concept. The influence relation will allow us to talk about where information can be stored on the queue or which information from the queue can be compared with the input.

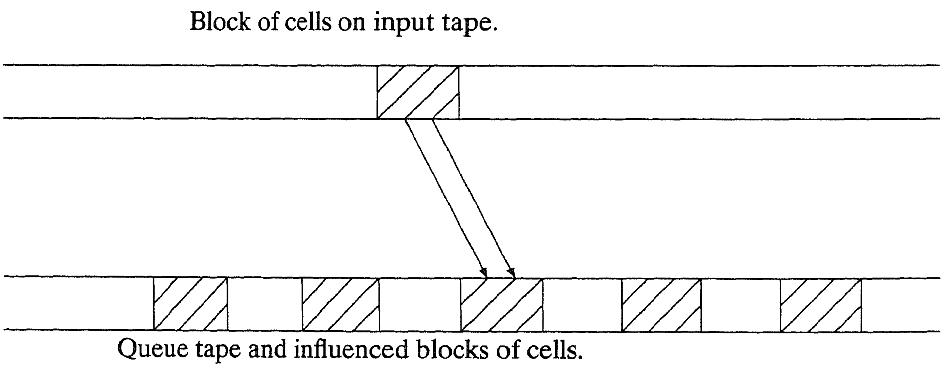

FIG. 6. *Blocks on the queue influenced by a block on the input.*

It is worth stating a few facts about the influence relations. Each tape cell is directly influenced by exactly one input cell. It is also forward and backward influenced by exactly one tape cell. The cells directly influenced by a contiguous block of input cells form a contiguous block. This holds also for forward and backward influence.

The sequence of blocks influenced by a block of input cells will be used with the crossing sequence around the blocks. Crossing sequences for queue machines need a special definition.

DEFINITION 3.8. A *partial configuration* of the machine at some time t is the state of the machine at that time, the position of all the heads on their respective tape, the contents of the cells $h_r(t)$, $h_{in}(t)$, and the contents of the cells immediately preceding those two cells.

DEFINITION 3.9. The *crossing sequence* (c.s.) associated with a cell d_i is the partial configuration at the time t when h_r goes from cell d_i to cell d_{i+1} (that is, $h_r(t-1) = d_i$ and $h_r(t) = d_{i+1}$) plus the partial configuration at the time when h_w goes from d_i to d_{i+1}. Since using more than n^2 tape cells would take too much time, we may assume that each head position can be described in $O(\log n)$ bits.

The *crossing sequence* around a region $d_i \cdots d_j$ is the c.s. associated with d_{i-1} concatenated with the one associated with d_j.

The *crossing sequence* around a list of regions is the concatenation of the c.s. around each of the regions.

Intuitively, for a deterministic computation, changing a block of input will change only the influenced regions, provided that the change does not alter the crossing sequence around the influenced regions. For a nondeterministic computation, the situation is a little more delicate, but the idea is the same. We need the backward influence to be able to deal with nondeterministic computations. A nondeterministic machine can guess the input on the queue and start the computation before the input head even moves once. A change in an input block will have "backward effects" on that computation.

For every computation path, there is a backward computation path consisting of all the configurations in reverse order. Moreover, there is a queue machine Q' that has as accepting computation paths all the backward accepting computations of Q. Just exchange the role of the read and write heads: $h'_w(t) = h_r(t)$ and $h'_r(t) = h_w(t)$. For the computation, the time and the heads go backwards. The influence definition was designed such that the forward influence on the tape for Q corresponds to the backward influence for Q' and vice versa. The region influenced by a block of *tape* cells will be the same for Q and Q'. The blocks of cells influenced by a block of *input* cells will differ slightly, because the direct influence will be directed at a different part of the tape. However, this does not affect the proof.

In the following, a *cycle* $\sigma(t)$ is a half-open interval (of time) $[t, \hat{t})$ such that $h_r(\hat{t}) = h_w(t)$ if $\hat{t} > t$ or such that $h_r(t) = h_w(\hat{t})$ if $\hat{t} < t$ (backward cycle). Given a time τ_1, we will be interested in nonoverlapping contiguous cycles $\sigma_1(\tau_1), \sigma_2(\tau_2), \cdots$ starting at time τ_1, such that $\sigma_1(\tau_1) = [\tau_1, \tau_2)$, $\sigma_2(\tau_2) = [\tau_2, \tau_3)$, and so on. In what follows, whenever we count cycles, the start time τ_1 either will be specified or will be clear from context and we will count the successive nonoverlapping contiguous cycles, as induced by the computation of Q. Backward cycles could alternatively be defined by using backward computations. Notice that the blocks of cells influenced by a block of input cells form a sequence of blocks, one block for each cycle.

CLAIM 3.10. *For any t, if $\hat{t} > t$ is fewer than s cycles away from t, then each cell in Queue($\hat{t}$) is influenced by at most s input cells in $\tilde{x}\#\tilde{x}^R$.*

Proof. Let the chain of cycles starting from $\tau_1 = t$ be $\sigma_1(\tau_1)$, $\sigma_2(\tau_2), \cdots$. The proof is by induction on the indices s. No cell in *Queue*(τ_1) is influenced by any input cell in $\tilde{x}\#\tilde{x}^R$. During σ_1, each cell written is influenced by exactly one input cell. Suppose the claim is true for cycles σ_1 through σ_{s-1}. During the cycle $\sigma_s(\tau_s)$, each cell written is influenced by one new input cell (possibly) and by each input cell that influences the cell scanned by h_r. This adds up to at most s input cells. □

DEFINITION 3.11. For each i, we say that x_i is a *valid* block if *Queue*(t'_0) contains a cell that is influenced by neither x_i nor x_i'.

Informally, x_i is valid if each of x_i and $x_i{}'$ is read within one cycle. Indeed, if x_i is not read within one cycle, then x_i directly influences all of *Queue*(t_i) and hence influences every cell of the tape by transitivity, including every cell of *Queue*(t_0'), where t_0' is the time when h_{in} leaves x_1'.

Next, we need to show that valid blocks exist. We need the existence of only one valid block, but, in fact, the majority of blocks are valid.

CLAIM 3.12. *If there is no valid block, then Q takes* $\Omega(n^{4/3})$ *time.*

Proof. Pick a cell d on *Queue*(t_0'). Suppose there is no valid block. This means that for all i, d is influenced by either x_i or $x_i{}^R$. It means that d is influenced by at least m different cells. By Claim 3.10, we know that then the machine makes at least $m-1$ cycles from t_1 to t_0'. By Claim 3.3, the queue has length at least $n/2 - O(\log n)$ for each cycle, so the algorithm will take at least $(m-1)(n/2 - O(\log n)) \in \Omega(n^{4/3})$. □

In the following, we may assume there is at least one valid block. The next two claims explain why a valid block is a part of the input that has been coded sequentially on the queue.

CLAIM 3.13. *For each valid block* x_j, *any two cells in* x_j *influence disjoint sets of cells on the queue. Moreover, cells in* x_j' *also influence disjoint sets of cells on the queue. However, some cells on the queue can be influenced by both a cell of* x_j *and a cell of* x_j'.

Proof. If x_i is a valid block, each of x_i and x_i' must be read within one cycle. Within one cycle, each cell written into is influenced by at most one cell of x_i. This property will be preserved by transitivity throughout the successive cycles, either backward or forward. The same situation arises for x_i'. □

CLAIM 3.14. *For any time t, the regions influenced by the sequence of cells of a valid block* x_j *form a contiguous ordered sequence on Queue*(t). (*The same statement holds for* x_j'.)

Proof. This can be seen with a similar argument as in the previous claim. □

For our valid block x_i, both x_i and $x_i{}'$ have been coded sequentially on the queue. Now we have to show that it takes $\Omega(n^{4/3}/\log n)$ time to check $x_i{}' = x_i{}^R$. Intuitively, we can check only a constant number of bits of $x_i{}'$ at each cycle. Each cycle takes as much time as the size of the queue at that time. The strategy is to show that the size of the queue cannot decrease too much at each cycle, for each of the forward and backward computations. Then, showing that many cycles are required will provide the lower bound.

CLAIM 3.15. *If* $\hat{t} > t_{i-1}'$ *is fewer than s cycles away from* t_{i-1}' *and* $t < t_i$ *is fewer than s cycles before* t_i, *then* $|Queue(t)| + |Queue(\hat{t})| \geq n^{2/3} - O(s \log n)$.

Proof. Let x_i be a valid block, $i > 0$. Let $x_i = uv$, where u and v are strings of equal size (± 1).

If there is a time τ such that $h_{in}(\tau) \in v'$ and $h_r(\tau)$ is influenced by v, then choose $y = u$, otherwise, choose $y = v$. In both cases, for all t, if $h_{in}(t) \in y'$, $h_r(t)$ is not influenced by y. This is immediate from Claim 3.14 for the case $y = v$. For the case $y = u$, let τ be such that $h_{in}(\tau) \in v'$ and $h_r(\tau)$ is influenced by v. Let d be a cell on *Queue*(τ) not influenced by x_i or x_i'. By Claim 3.14, the region influenced by $y = u$ is after d and the region influenced by $y' = u'$ is before d (refer to Figs. 7 and 8). The regions cannot intersect.

As a consequence of our choice of y, we have that the regions influenced by y and by y' are disjoint.

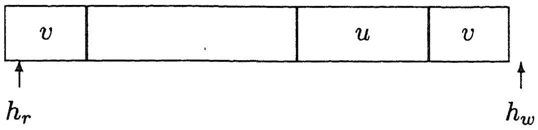

FIG. 7. *Influence of* $x_i = uv$ *on* $Queue(\tau)$.

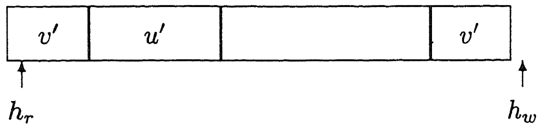

FIG. 8. *Influence of* $x_i' = v'u'$ *on* $Queue(\tau)$.

Let t and $\hat{t}$ be as in the statement of the claim. Let $\bar{x}$ be the string x for which y is deleted. The size of y is about $n^{2/3}$:

$$|y| \geq p/2 - 1 = \lfloor n/2m \rfloor/2 - 1 = \frac{\left\lfloor \frac{n}{2\lfloor n^{1/3}/4 \rfloor} \right\rfloor}{2} - 1 \in n^{2/3} - O(1).$$

The size of $\bar{x} = n - |y| \in n - n^{2/3} + O(1)$.

Let S be the set of cells influenced by y. We show below that x can be computed from $\bar{x}$, t, $\hat{t}$, the position of y in $\bar{x}$, the crossing sequence around S from time t to time $\hat{t}$, $Queue(t)$, and $Queue(\hat{t})$. If each item is encoded as self-delimiting, this description takes $n - n^{2/3} + O(s\log(n)) + |Queue(t)| + |Queue(\hat{t})|$ bits. Because $K(x) \geq n$, it then follows that $|Queue(t)| + |Queue(\hat{t})| \geq n^{2/3} - O(s\log(n))$.

We compute y with the information provided in the following way. For all binary strings z of equal length as y, let x_z be the string x for which z has been substituted for y. Run Q on all strings $x_z\#x_z{}^R$ until one that matches the description is found. By construction, $z = y$ matches the description. Leading to a contradiction, suppose $z \neq y$ matches the description as well. Let C_y be the accepting computation on $x\#x^R$ and C_z the accepting computation on $x_z\#x_z^R$ that matches the description. Then, by cutting and pasting the two computations we can construct a legal computation of Q on $x_z\#x^R$. Let S_z be the set of cells influenced by z in C_z. Because the crossing sequence includes the position of all heads, the regions in S and in S_z occupy the same absolute positions on the queue. Let t_y be the time when h_{in} leaves y. We can compose the accepting computation as follows. Use any of the two computations up to time t. At this time, we will have $Queue(t)$.

From time t to t_y, we are dealing with backward influence. If h_w is scanning a cell of S, then h_r is scanning either a cell of S or a cell immediately before it. If h_w is scanning a cell not in S, then h_r is also scanning a cell not in S or immediately before it. Since the cell before the region has been included in the c.s., it is possible to follow C_y when h_w is in S and C_z when h_w is out of S. Notice that h_{in} cannot scan a cell of y while h_w is writing a cell out of S because of the direct influence. Moreover, h_in cannot scan a cell of z while h_w writes a cell of S because that cell would be influenced by both y and z, which cannot happen by our choice of y.

From time t_y to time $\hat{t}$,we are dealing with forward influence. Follow C_z when h_r is not in S and follow C_y when h_r is in S.

At time $\hat{t}$, the queue will correspond with the queue in both computations. Just complete the computation following any of C_y or C_z. This gives an accepting computation for a string $\notin L$, which is a contradiction. □

CLAIM 3.16. *The machine makes* $\Omega(n^{4/3}/\log n)$ *steps before* t_i *or after* t'_{i-1}.

Proof. Let T be the time Q accepts. Both $Queue(0)$ and $Queue(T)$ are of length 0. By the previous claim, $|Queue(0)| + |Queue(T)| \geq n^{2/3} - O(s \log n)$. Let $|Queue(0)| + |Queue(T)| \geq n^{2/3} - cs \log n$. This means Q makes at least $n^{2/3}/(c \log n)$ cycles for some constant c. At least $n^{2/3}/(2c \log n)$ of those cycles will have a queue of size $\Omega(n^{2/3})$, by the previous claim. This makes a total of $\Omega(n^{4/3}/\log n)$ steps. □

COROLLARY 3.17. *For off-line one-way-input one-queue machines, nondeterministic linear time is not closed under complement.*

Proof. The complement of the palindrome language used in the proof of Theorem 3.2 can be accepted in nondeterministic linear time. This can be seen as follows. If the string is of the form $w_1\#w_2$, where $|w_1| = |w_2|$, nondeterministically go and read position i of w_1 for which there is a discrepancy. While doing that, push i symbols on the queue. Then nondeterministically go and read the corresponding position of w_2. Verify the position by using the number of symbols pushed on the queue.

Other cases can be checked in deterministic linear time. Finding which case applies can be made by a nondeterministic initial move. This concludes the proof of Theorem 3.2. □

4. More queues versus fewer queues. In this section we study the power of queue machines with different numbers of queues. We first provide some straightforward upper bounds: Two queues work as well as k queues in the nondeterministic case. This motivates our research focusing on small numbers of queues. One queue can simulate k queues in quadratic time, deterministically or nondeterministically. We then provide tight, or almost tight, lower bounds for our simulations mentioned above.

4.1. Upper bounds.

THEOREM 4.1. *Two stacks can simulate one queue in linear time, for both deterministic and nondeterministic machines.*

Proof. We design a machine P with two stacks pd1, pd2. To simulate a queue, every time a symbol is pushed into the queue, P pushes the same symbol into pd1. If a symbol is taken from the queue, then P pops a symbol from pd2 if pd2 is not empty. If pd2 is empty, then P first unloads the entire contents of pd1 into pd2 and then pops the top symbol from pd2. At the end of the input, P accepts if and only if the one-queue machine accepts. □

THEOREM 4.2. *Two queues can nondeterministically simulate* k *queues for any fixed* k *in linear time.*

Proof. This theorem follows from the method used by Book and Greibach [BG70] to nondeterministically simulate k tapes by two tapes in linear time. For the sake of completeness, we will describe the idea. The two-queue machine guesses the computation of the k-queue machine and puts this guess on one queue in the form $\mathrm{ID}_1, \mathrm{ID}_2, \cdots$, where ID_i contains the state of the k queue machine and the $k+1$ queue symbols scanned by the k queue heads and the input head at step i. First, check that the state in each ID is consistent with the previous ID and check the correctness of the guessed input symbol in each ID_i by scanning the ID's and moving the input head when necessary. Then, scan the ID's again k times, each time simulating one of the k queues of the simulated machine on the other queue. This simulation takes $O((k+1)n) = O(n)$ time. □

THEOREM 4.3. *Three stacks can nondeterministically simulate* k *queues in linear time.*

Proof. Combine the ideas from the above two theorems; i.e., guess the computation of the k-queue machine as before, and put the guess into one stack. Save this guess also to another stack (but put a marker on the top). Then simulate a queue and check the correctness of the guess. (The simulation needs two stacks; one of the stacks has the guessed computation saved in the bottom.) After simulating one queue, retrieve the guessed contents; again put it into two stacks. Repeat this process for each queue. □

Remark. It is a folklore fact, and easily verified, that one-queue machines accept precisely the r.e. languages. In contrast, one-stack machines accept only CFL's. Hence, one queue is better than one stack. However, when we have more stacks, more stacks seem to be better than queues because they are more efficient. It was proved in [HM81] that four stacks can simulate a queue in real time.

THEOREM 4.4. *One queue can simulate k queues in quadratic time, both deterministically and nondeterministically.*

Proof. This is similar to the simulation of k tapes by one tape by Hartmanis and Stearns [HS65] (see [HU79, p. 292]). □

This also relates to the interesting problem of whether two heads (on one tape) are better than two tapes (each with one single head). Vitányi [Vit84a] showed that two tapes cannot simulate a queue in real time if at least one of the tape heads is within $O(n)$ cells from the start cell at all times. We saw that two stacks can simulate a queue in linear time and four stacks can do this in real time. It would be interesting to know whether two or three stacks can do this in real time. The question of how to deterministically simulate k queues by two queues in $O(n^2)$ time, like the Hennie–Stearns simulation in the tape case [HS66], remains open.

4.2. Lower bounds. We now prove optimal lower bounds for the above simulations. Let L be the following language.

$$
\begin{aligned}
L = \{\, a \,\&\, & b_0^1 b_1^1 \cdots b_k^1 \# b_0^2 b_0^3 b_1^2 b_2^2 b_1^3 b_3^2 \cdots b_{2i}^2 b_i^3 b_{2i+1}^2 \cdots b_{k-1}^2 b_{(k-1)/2}^3 b_k^2 \\
& b_0^4 b_{(k+1)/2}^3 b_1^4 b_2^4 b_{(k+3)/2}^3 b_3^4 \cdots b_{2i \bmod (k+1)}^4 b_i^3 b_{(2i+1) \bmod (k+1)}^4 \cdots b_{k-1}^4 b_k^3 b_k^4 \,\&\, a : \\
& b_i^1 = b_i^2 = b_i^3 = b_i^4 \text{ for } i = 0, \cdots, k \\
& \quad \text{all } b_i^j \text{ have format } \$x\$, \text{ where } x \in \{0,1\}^* \\
& k \text{ is odd, and } a \in \{0,1\}^* \,\}.
\end{aligned}
$$

When we prove the lower bound, all the b_i^j will have the same length. The string between the first & and second & can be obtained by copying $b_0 b_1 \cdots b_k$ three times:

$$b_0 b_1 \cdots b_k \ \# b_0 b_1 \cdots b_k \ b_0 b_1 \cdots b_k,$$

and then adding one more copy of $b_0 b_1 \cdots b_k$ by inserting block b_i after $2i$ blocks, starting from $\# b_0$ in above. The superscripts on the b_i's are used only to facilitate later discussions. L can be considered as a modified version of a language used in [Maa85]. We have added a string a on both ends. The purpose of a is to prevent the queue from shrinking, since if we choose a to be a long K-random string, then before the second a is read the size of the queue has to be at least about $|a|$. We have to prevent the queue from shrinking because otherwise the crossing sequence argument would not work. In addition to the techniques in [Maa85], and [LV88], we will need the techniques introduced in this paper to treat queues.

An alternative way to describe the language L is as follows. Let y and z be sequences of blocks in which each block is of form $\$u\$$, where $u \in \{0,1\}^*$. Define $intermingle(y) =$

z if (1) the blocks of z in positions $i \equiv 2 \pmod 3$ form the string y ($z_2 z_5 z_8 \cdots = y_1 y_2 y_3 \cdots$) and (2) the remaining blocks of z form the string yy.

Then, $L = \{a\&y\#intermingle(y)\&a : y$ contains an even number of blocks$\}$.

THEOREM 4.5. *Simulating a deterministic two-queue machine with a one-way input tape by a nondeterministic one-queue machine with a one-way input tape requires* $\Omega(n^2/\log^2 n \log\log n)$ *time.*

Proof. We will show that the L just defined requires $\Omega(n^2/\log^2 n \log\log n)$ time on a nondeterministic one-queue machine. Since L can be trivially accepted by a deterministic two-queue machine in linear time, the theorem will follow.

Now, aiming at a contradiction, assume that a one-queue machine M accepts L in time $T(n)$, which is not in $\Omega(n^2/\log^2 n \log\log n)$. Without loss of generality, we assume that M has a binary queue alphabet and that M accepts with a final state and an empty queue. We use the same notation and definitions as in the previous section, e.g., $Queue$, $|Queue(t)|$, h_{in}, h_r, h_w, cycles, and crossing sequence.

Choose a large n and a large enough C such that $C >> |M| + c$ and all the subsequent formulas make sense, where $|M|$ is the number of bits needed to describe M and c is a constant given in Claim 4.9, which follows. Choose an incompressible string $X \in \{0,1\}^{2n}$, $K(X) \geq |X|$. Let $X = X'X''$, where $|X'| = |X''| = n$. Divide X'' into $k+1 = n/(C\log\log n)$ equal parts, $X'' = x_0 x_1 \cdots x_k$, where each x_i is $C\log\log n$ long. Consider a word $w \in L$, where $a = X'$, $b_i^j = x_i$ for $1 \leq j \leq 4$, and $0 \leq i \leq k$. Fix a shortest accepting path P of M on w. We will show that M takes $\Omega(n^2/\log^2 n \log\log n)$ time on P. Since n is linearly related to the size of the input, this will provide the lower bound in the theorem.[3]

Consider only the path P. Let $g(n) = C^5 \log^2 n \log\log n$. Let $t_{\&}$ be the time when h_{in} reaches the first &, $t'_{\&}$ be the time h_{in} reaches the second &, and $t_{\#}$ be the time when h_{in} reaches #.

CLAIM 4.6. $|Queue(t)| \geq n - O(\log n)$ *for every* $t_{\&} \leq t \leq t'_{\&}$.

Proof. The proof of this claim is the same as that of Claim 3.3 and is omitted. □

CLAIM 4.7. *The number of cycles from time* $t_{\&}$ *to* $t'_{\&}$ *is less than* $n/g(n)$.

Proof. This follows directly from the previous claim. Each cycle is of length $\Omega(n)$ and hence takes $\Omega(n)$ time. If M requires at least $n/g(n)$ cycles from $t_{\&}$ to $t'_{\&}$, then M used $\Omega(n^2/\log^2 n \log\log n)$ time, which is a contradiction. □

For each time t, we say that a substring s of the input w is *mapped into* a set S of cells on $Queue(t)$ if all the cells influenced by s on $Queue(t)$ are in S.

CLAIM 4.8. *Let* $k' = k/2 - n/g(n)$. *At time* $t_{\#}$, $Queue(t_{\#})$ *can be partitioned into two segments,* $S_1(t_{\#})$ *and* $S_2(t_{\#})$, *such that* k' b_i^1*'s, say* $b_{i_1}^1, \cdots, b_{i_{k'}}^1$, *are mapped into* $S_1(t_{\#})$ *and* k' *other* b_i^1*'s, say* $b_{j_1}^1, \cdots, b_{j_{k'}}^1$, *are mapped into* $S_2(t_{\#})$.

Proof. Consider any cell c_0 on the $Queue(t_{\#})$. By the nature of the queue and Claim 4.7, at most $m = n/g(n)$ b_i^1's can influence c_0 at $t_{\#}$ because M made no more than m cycles on the queue from $t_{\&}$ to $t_{\#}$. Hence, for any partition of $Queue(t_{\#})$ into two parts, $S_1(t_{\#})$ and $S_2(t_{\#})$, there can be at most $2m$ b_i^1 blocks, each influencing both $S_1(t_{\#})$ and $S_2(t_{\#})$. Each of the rest of the $k+1-2m$ b_i^1 blocks either influences only $S_1(t_{\#})$ or influences only $S_2(t_{\#})$. It is now trivial to build S_1 and S_2 by moving the border cell by cell until the claim is satisfied. □

Now, let $S_1(t_{\#})$ and $S_2(t_{\#})$ be as specified in the previous claim. At any time t, let

[3]Here, as in the previous section, the language does not have a string of each length. The proof provides an input that causes the machine to take a long time for each length that has at least one string in the language. To produce a hard string for each length, just add a finite padding in the definition of the language; for example, allow markers to repeat up to four or five times.

$S_1(t)$ be the part of *Queue*(t) influenced by $S_1(t_\#)$ and let $S_2(t)$ be the complementary region on *Queue*(t). Let S_1 be the set of all cells on the tape influenced by $S_1(t_\#)$ and S_2 be the other cells.

The next claim is a simple generalization of a theorem proved in [Maa85, Thm. 3.1]. The proof of the claim is a simple reworking of the Maass proof and is hence omitted.

CLAIM 4.9. *Let S be a sequence of numbers from $0, \cdots, k$, where $k = 2^l$ for some l. Assume that every number $b \in \{0, \cdots, k\}$ is somewhere in S adjacent to the numbers $2b \pmod{k+1}$ and $2b \pmod{k+1} + 1$. Then, for every partition of $\{0, \cdots, k\}$ into two sets G and R such that $|G|, |R| > k/4$, there are at least $k/(c \log k)$ (for some fixed c) elements of G that occur somewhere in S adjacent to a number from R.* □

A $k/\sqrt{\log k}$ upper bound corresponding to the lower bound in this claim is contained in [Li88]. A more general, but weaker, upper bound can be found in [Kla84].

Remark 4.1. For each word $w \in L$, the sequence of the subscripts of the substrings (in the order they appear) in w between the # sign and the second & satisfies the requirements in Claim 4.9. For example, given k, such a sequence is formed by inserting i after $2i$th number, $i = 0, 1, \cdots, k$, in the following sequence:

$$0, 1, 2, \cdots, k, 0, 1, 2, \cdots, k.$$

Therefore, each number i is adjacent to $2i \pmod{k+1}$, and $2i + 1 \pmod{k+1}$. In what follows we will also say that a pair of b_i blocks are adjacent if their subscripts are adjacent in the above sequence.

CLAIM 4.10. *At time $t'_\&$, the b_i's between # and the second & are mapped into $Queue(t'_\&)$ in the following way: either*

1. *a set, $\bar{S}_1$, of $k/(3c \log k)$ b_j's, which belong to $\{b^1_{j_1}, \cdots, b^1_{j_{k'}}\}$, are mapped into $S_1(t'_\&)$; or*
2. *a set, $\bar{S}_2$, of $k/(3c \log k)$ b_i's, which belong to $\{b^1_{i_1}, \cdots, b^1_{i_{k'}}\}$, are mapped into $S_2(t'_\&)$, where $c << C$ is the small constant in Claim 4.9.*

Proof. By Claim 4.7, from time $t_\#$ to $t'_\&$, M makes fewer than $n/g(n)$ cycles. Hence, h_w can alternate between S_1 and S_2 fewer than $2n/g(n)$ times. Each time h_w alternates between S_1 and S_2, h_w can map at most one adjacent pair of b^j_i blocks into both $S_1(t'_\&)$ and $S_2(t'_\&)$. All other pairs are each mapped totally into $S_1(t'_\&)$ or totally into $S_2(t'_\&)$. There are $\theta(k)$ such pairs in L.

Combining Claim 4.8, Claim 4.9, and Remark 4.1, we know that there are at least $k/c \log k - n/(C^5 \log^2 n \log \log n)$ pairs of b^j_i blocks such that each of these pairs contains a component belonging to $G = \{b^1_{i_1}, \cdots, b^1_{i_{k'}}\}$ and another component belonging to $R = \{b^1_{j_1}, \cdots, b^1_{j_{k'}}\}$. Most of these pairs, except $n/g(n)$ of them by the previous paragraph, are mapped either totally into $S_1(t'_\&)$ or totally into $S_2(t'_\&)$. Hence, either (1) or (2) must be true. □

Without loss of generality, assume that (1) of Claim 4.10 is true.

CLAIM 4.11. *Let t_{end} be the time M accepts. $|Queue(t_{end})| = 0$. Then there exists a time $t'_\& \le t_1 \le t_{end}$ such that $|Queue(t_1)| \le n/(C^5 \log n)$ and from $t'_\&$ to t_1 M made fewer than $n/(C^5 \log n \log \log n)$ cycles.*

Proof. Otherwise M spends $\Omega(n^2/(\log^2 n \log \log n))$ time, a contradiction. □

CLAIM 4.12. *There also exists a time $t_0 \le t_\&$ such that $|Queue(t_0)| \le n/(C^5 \log n)$ and from t_0 to $t_\&$ M made fewer than $n/(C^5 \log n \log \log n)$ cycles.*

Proof. Note that by Claim 4.6 $|Queue(t_\&)| \ge n - O(\log n)$. Thus, we can choose t_0 to be the last time step before $t_\&$ such that $|Queue(t_0)| \le n/(C^5 \log n)$. Hence, if the claim is not true, M would spend $\Omega(n^2/(\log^2 n \log \log n))$ time, a contradiction. □

By Claim 4.7 the number of cycles M made from $t_{\&}$ to $t'_{\&}$ is less than $n/g(n)$. By Claims 4.11 and 4.12 M made at most $n/(C^5 \log n \log\log n)$ cycles from time $t'_{\&}$ to t_1 and from time t_0 to $t_{\&}$. Hence, the length of the crossing sequence at the boundary of S_1 and S_2 from $t_{\&}$ to t_1 is shorter than $n/C^4 \log n \log\log n$. For every j, if a $b_j^k \in \bar{S}_1$ for some k, then b_j^1 is mapped into S_2 by Claim 4.10.

Now we describe a program that reconstructs X with less than $|X|$ information. The program uses $Queue(t_0)$, $Queue(t_1)$, the crossing sequence around S_1, the string X where the b_j^k blocks have been deleted, and the relative position of those b_j^k blocks.

Consider every Y such that $|Y| = |X|$ and $Y = a\, y_0 \cdots y_k$ for some $y_0 \cdots y_k$.

1. Check if Y is the same as X at positions other than those places occupied by $b_j^k \in \bar{S}_1$.
2. If (1) is true, then construct the input w_Y the same way w was constructed except with x_i replaced by y_i for $i = 0, 1, \cdots, k$.
3. Copy the contents of $Queue(t_0)$ on the queue. Then simulate M from t_0 to t_1 such that h_r never goes into S_2. Whenever h_r reaches the border of S_2 it compares the current ID with the corresponding one in the crossing sequence. If they match, then M *jumps* over S_2 and, starting from the next ID on the other side of S_2, M continues until time t_1. At time t_1, compare the actual queue with what it is supposed to be. Accept Y if everything worked correctly.
4. This computation will accept if and only if $Y = X$. If it is not the case, we could compose an accepting computation on M for the string where the b_j^1 blocks correspond to those in Y and the other b_j blocks correspond to those in X. This can be done in a way very similar to what was done in Claim 3.15. The details are omitted here.

The information we used in this program is only the following:

1. $X - \bar{S}_1$, plus the information to describe the relative locations of $b_j^k \in \bar{S}_1$ in X. This would require at most

$$\begin{aligned} |X| - |\bar{S}_1||b_j^k| + O(|\bar{S}_1|\log(k/|\bar{S}_1|)) &\le 2n - |\bar{S}_1| C \log\log n + O(|\bar{S}_1|\log\log n) \\ &\le 2n - (|\bar{S}_1| C \log\log n)/2 \\ &\le 2n - n/C^2 \log n, \end{aligned}$$

where in the first line the second term is for the b_j's in $\bar{S}_1$, the third term is for the information to describe the relative positions of $b_j \in \bar{S}_1$: To represent $|\bar{S}_1|$ elements of $\{0, 1, \cdots, k\}$, sort the elements, determine the sequence of their differences, and use a self-delimiting encoding of the natural numbers to write each difference. The final encoding has approximately $O(|\bar{S}_1|\log(k/|\bar{S}_1|))$ bits (see, for example, [LV88], [Lou84], [Eli75]).
2. Description of the crossing sequence, of length less than $n/(C^4 \log n \log\log n)$, around S_2. Again by the above efficient encoding method, this requires at most $n/(C^3 \log n)$ bits. The detail of this encoding can be found in [LV88]. The idea is as follows: Each item in the c.s. is (state of M, h_{in}'s position). Trivial encoding of $n/(C^4 \log n \log\log n)$ long c.s. needs $n/(C^4 \log\log n)$ bits. However, we can use the above method and encode only the differences of h_{in}'s positions and thus use fewer than $n/(C^3 \log n)$ bits.
3. Description of the contents of S_2 at times t_0 and t_1. But, for $i = 0, 1$ $|Queue(t_i)| \le n/(C^5 \log n)$.
4. Extra $O(\log n)$ bits to describe the program discussed above.

The total is less than $2n - n/(C \log n)$. Therefore, $K(X) < |X|$, a contradiction. □

COROLLARY. Simulating two deterministic tapes by one nondeterministic queue requires $\Omega(n^2/\log^2 n \log\log n)$.

Proof. L can also be accepted by a two-tape Turing machine in linear time. □

THEOREM 4.13. *To simulate two deterministic queues by one deterministic queue requires* $\Omega(n^2)$ *time.*

Proof idea. Define a language L_1 as follows $(a, x_i, y_i \in \{0,1\}^*)$.

$$\begin{aligned} L_1 &= \{a \,\&\, x_1\$x_2\$\cdots\$x_k\#y_1\$\cdots\$y_l\#(1^{i_1},1^{j_1})(1^{i_2},1^{j_2})\cdots(1^{i_s},1^{j_s})\,\&\, a \mid \\ x_p &= y_q;\ \ (p = i_1+\cdots+i_t,\ q = j_1+\cdots+j_t) \text{ and } 1 \le t \le s\}. \end{aligned}$$

L_1 can be accepted by a deterministic two-queue machine in linear time. Using the techniques in the above theorem and in [LV88], where it is proved that one deterministic Turing machine tape requires square time for this language, it can be shown that L_1 requires $\Omega(n^2)$ for a one-queue deterministic machine. We omit the proof. □

Acknowledgment. We are grateful to the referee for his careful analysis and extensive comments on the first version of this paper. We would also like to thank Andy Klapper and Roy Rubinstein for their helpful comments.

REFERENCES

[Aan74] S. O. AANDERAA, *On k-tape versus (k − 1)-tape real-time computations*, in R. M. Karp, ed., Complexity of Computation, SIAM-AMS Proceedings, Vol. 7, American Mathematical Society, Providence, RI, 1974, pp. 75–96.

[BG70] R. BOOK AND S. GREIBACH, *Quasi real-time languages*, Math. Systems Theory, 4 (1970), pp. 97–111.

[BGW70] R. BOOK, S. GREIBACH, AND B. WEGBREIT, *Time- and tape-bound Turing acceptors and afl's*, J. Comput. System Sci., 4 (1970), pp. 606–621.

[Cha77] G. CHAITIN, *Algorithmic information theory*, IBM J. Res. Develop., 21 (1977), pp. 350–359.

[DGPR84] P. DURIS, Z. GALIL, W. PAUL, AND R. REISCHUK, *Two nonlinear lower bounds for on-line computations*, Inform. and Control, 60 (1984), pp. 159–173.

[Eli75] P. ELIAS, *Universal codeword sets and representation of integers*, IEEE Trans. Inform. Theory, IT-21 (1975), pp. 194–203.

[GKS86] Z. GALIL, R. KANNAN, AND E. SZEMERÉDI, *On nontrivial separators for k-page graphs and simulations by nondeterministic one-tape Turing machines*, in Proc. 18th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1986, pp. 39–49.

[HM81] R. HOOD AND R. MELVILLE, *Real-time queue operations in pure LISP*, Inform. Process. Lett., 13 (1981), pp. 50–54.

[HS65] J. HARTMANIS AND R. STEARNS, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc., 117 (1965), pp. 285–306.

[HS66] F. HENNIE AND R. STEARNS, *Two tape simulation of multitape Turing machines*, J. Assoc. Comput. Mach., 13 (1966), pp. 533–546.

[HU79] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[Kla84] M. KLAWE, *Limitations on explicit construction of expanding graphs*, SIAM J. Comput., 13 (1984), pp. 156–166.

[Kol65] A. KOLMOGOROV, *Three approaches for defining the concept of information quantity*, Problems Inform. Transmission, 1 (1965), pp. 1–7.

[Kos79] S. KOSARAJU, *Real time simulation of concatenable double-ended queues by double-ended queues*, in Proc. 11th ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1979, pp. 346–351.

[Li85a] M. LI, *Lower bounds by Kolmogorov complexity*, in 12th ICALP, Lecture Notes in Computer Science, No. 194, Marcel Dekker, New York, 1985, pp. 383–393.

[Li85b] ———, *Lower bounds in computational complexity*, Ph.D. thesis, Tech. Report TR85-663, Computer Science Department, Cornell University, 1985.

[Li88] ———, *Simulating two pushdowns by one tape in* $O(n^{1.5}(\log n)^{0.5})$ *time*, in Proc. 26th IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1985, pp. 56–64.

[Lou84] M. C. Loui, *The complexity of sorting on distributed systems*, Inform. and Control, 60 (1984), pp. 70–85.

[LS81] B. L. Leong and J. I. Seiferas, *New real-time simulations of multihead tape units*, J. Assoc. Comput. Mach., 28 (1981), pp. 166–180.

[LV88] M. Li and P. M. B. Vitányi, *Tape versus queue and stacks: the lower bounds*, Inform. and Comput., 78 (1988), pp. 56–85.

[LLV86] L. Longpré, M. Li, and P. M. B. Vitányi, *The power of the queue*, in Structure in Complexity Theory, Lecture Notes in Computer Science, Springer-Verlag, 223, 1986, pp. 219–233.

[Maa85] W. Maass, *Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines*, Trans. Amer. Math. Soc., 292 (1985), pp. 675–693.

[MSS87] W. Maass, G. Schnitger, and E. Szemeredi, *Two tapes are better than one for off-line Turing machines*, in Proc. 19th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1987, pp. 94–100.

[Pau82] W. Paul, *On-line simulation of* $k+1$ *tapes by* k *tapes requires nonlinear time*, Inform. and Control, 53 (1982), pp. 1–8.

[PSS81] W. Paul, J. Seiferas, and J. Simon, *An information-theoretic approach to time bounds for on-line computation*, J. Comput. System Sci., 23 (1981), pp. 108–126.

[Sol64] R. Solomonoff, *A formal theory of inductive inference*, Part 1 and Part 2, Inform. and Control, 7 (1964), pp. 1–22, 224–254.

[Vit84a] P. M. B. Vitányi, *On two-tape real-time computation and queues*, J. Comput. System Sci., 29 (1984), pp. 1303–1311.

[Vit84b] ———, *One queue or two pushdown stores take square time on a one-head tape unit*, Tech. Report CS-R8406, Computer Science, CWI, Amsterdam, 1984.

[Vit85] ———, *An N**1.618 lower bound on the time to simulate one queue or two pushdown stores by one tape*, Inform. Process. Lett., 21 (1985), pp. 147–152.