The Complexity of Sorting with Networks of Stacks and Queues

Stefan Felsner * Martin Pergel †‡

Abstract

We consider a sorting problem on networks whose nodes are storage elements of type stack or queue. A railway switchyard could be an instance of such a network. Given is an input node where a permutation of items 1 to n is delivered and an output node where they are expected in sorted order. How many moves, where an item is transfered from one node to an adjacent node, are needed in the worst case for the sorting? Among others we have the following results: A characterization of networks where the sorting complexity is $\Theta(n \log n)$. A lower bound of $\Omega(n^{2-\epsilon})$ for the network consisting of only two stacks that can exchange items.

1 Introduction

In 1972 Tarjan published the article "Sorting Using Networks of Queues and Stacks" [8]. Tarjan's model consists of an acyclic directed graph, alternatively called *network* or *switchyard*, with a designated input node s and output node t and additional nodes representing storage buffers of type Q (queue) or S (stack). Suppose a permutation $i_1, i_2, \ldots, i_n$ of items $1, \ldots, n$ is entered at the source node of the network, the question is whether they can be sorted, i.e., whether there is a sequence of moves such that the items arrive at the output node in the correct order. A *move* consists of choosing an edge $e = (i, j)$ and transferring the item that can be extracted at i through e and insert it into the storage at j.

*Institut für Mathematik, Technische Universität Berlin. felsner@math.tu-berlin.de

†Department of Applied Mathematics (KAM), Charles University Prague. perm@kam.mff.cuni.cz

‡Support of grant GAUK 154907 is gladly acknowledged.

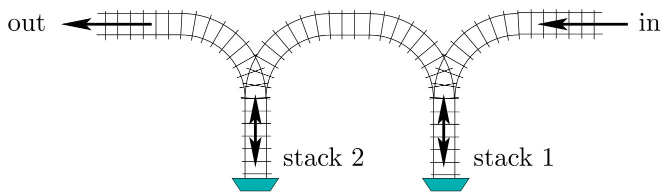


Figure 1: If the track between the stacks is directed we have 2 stacks in series, the permutation 2435761 is unsortable, making the track bidirectional allows to sort every input.

The question could be answered for some special types of networks. The first result being Knuth's characterization of permutations that can be sorted with a single stack as those avoiding the pattern 231, see [5, Exercises 2.2.1.2–6]. This line of research leads to the study of permutation classes, c.f. Bóna [3, 4]. A related line of research deals with token passing, in this model the nodes of the network are allowed to hold only a single item, again characterizations of sortable permutations are a central topic, e.g. Atkinson et al. [2]. Amato et al. [1] study the problem of reversing a train with a spur line just large enough to hold a single car. They have results for several cost models.

In this paper we shift the focus from existence to complexity. Quoting Tarjan: *"A circuit in the switchyard will allow us to sort any sequence"*, thus when looking at 'cyclic' networks we may ask the questions:

> How many moves are needed in the worst case to sort a permutation of n items in a given network?

We feel that the question is well motivated from a practical point of view, after all switchyards are cyclic in general and a specific order of the wagons of a train may be requested. Figure 1 shows a network with two stacks. The ability to use the track connecting the two stacks in two directions transforms the classical 2-stack problem into a 'cyclic' 2-stack problem as investigated in Theorem 3.

We were motivated to investigate the problem by discussions with König and Lübbecke [6]. They ask for approximation algorithms for a problem in steel processing where steel slabs are moved into a warehouse where they have to be placed on a fixed number of stacks. The aim is to allow extraction of the slabs in a prescribed order such that the amount of rearrangement (stack to stack transfers) is minimized. The hardness of the optimization problem is shown in [7].

In Section 2 we consider networks consisting of k communicating stacks, i.e., there is a directed edge between each ordered pair of stacks. We determine the asymptotic worst case for the number of moves required to sort in such a network for different choices of k. For $k \geq 3$ constant the complexity is $\Theta(n \log n)$, for $k \sim \log n$ it is $\Theta(\frac{n \log n}{\log \log n})$ and for $k = \sqrt{n}$ it is exactly $3n - \sqrt{n}$. In the case $k = 2$ we complement the trivial upper bound of $O(n^2)$ with a lower bound of $\Omega(n^{2-\epsilon})$ for all $\epsilon > 0$.

Section 3 deals with general networks. We identify two simple substructures of networks that allow sorting with $O(n \log n)$ moves. Networks avoiding these substructures are 'almost acyclic'. We show that the sorting complexity on such networks depends on the length of certain paths. In the conclusion we have collected some open problems.

2 Communicating Stacks

In this section we analyze the sorting complexity for networks of k communicating stacks. Such a network consists of input and output nodes s and t and k additional nodes each representing an unbounded stack, i.e., a storage of last-in-first-out type. There are edges (s, i), (i, j) and (j, t) for all $i, j \neq s, t$. Recall that we assume that the input to the sorting network consists of some permutation π of items numbered $1, 2, \ldots, n$.

2.1 The upper bound

Let us begin considering the case $k = 3$ and let S_0, S_1 and S_2 be the three stacks. The idea for the algorithm is to sort by recursive splitting. First the input is distributed on the stacks such that S_0 contains a block consisting of the $n/3$ smallest items, the block on S_1 consists the middle third of the items and the block on S_2 consists of the largest third of the items. The three blocks will be sorted and sent to the output one after the other. To begin with the block B_0 from S_0 is extracted. The smaller half of the items, i.e., those with a number smaller than the median of B_0, are moved to S_2 and the larger half the items, i.e., the remaining ones, are moved to S_1. This makes new blocks B_2' and B_1' on top of the blocks B_2 and B_1. Recursively first sort B_2' and then B_1'. If a block to be sorted in the recursive process is of size 1, then this item is sent to the output.

For a more formal description we need to enhance each block B with information about the items in it, in particular we need $\mathsf{min}(B)$ and $\mathsf{max}(B)$ to be the smallest and largest numbers of items in B and $\mathsf{stack}(B)$ to be the index of the stack of B, we use arithmetic modulo 3 on these indices. Here is a code describing the recursion.

```
sort(B)
    if |B| = 1 then output this item
    else
        i ← stack(B)
        m ← ⌊(min(B)+max(B))/2⌋
        create a new block B_− on S_{i−1}
            min(B_−) ← min(B) and max(B_−) ← m
        create a new block B_+ on S_{i+1}
            min(B_+) ← m + 1 and max(B_+) ← max(B)
        for b ∈ B do
            if b ≤ m then move b to B_− else move b to B_+
        sort(B_−)
        sort(B_+)
```

Correctness of the procedure follows from the fact that the algorithm always acts on the block containing the smallest items that have not yet been moved to the output. For the complexity note that when an element is moved, then it is transfered from a block B to a block whose size is only half of the size of B. Hence after $\log_2 n$ move operations the element is in a block of size 1 and will be output.

The procedure is easily generalized to the case of networks with $k > 3$ communicating stacks. In that case a block can be split into $k-1$ parts and the total number of moves between stacks is bounded by $n \log_{k-1}(n)$. Note that in this analysis we have already included the cost of moving items from the input to their initial blocks. Adding one unit per item for the move to the output we obtain a bound of $n(\log_{k-1}(n) + 1)$ for the total number of moves. The following simple observation allows to improve this slightly. Whenever, $\mathsf{sort}(B)$ is called the smallest element of the block can be moved directly to the output. Doing this saves at least one move for every recursive call. The number of recursive calls equals the number of inner nodes of a full $(k-1)$-ary tree with n leaves, i.e., it is $\frac{n-1}{k-2}$.

Theorem 1. *Every permutation π of $1, \ldots, n$ can be sorted in a network of $k \geq 3$ communicating stacks with at most $n \log_{k-1}(n) + n - \frac{n-1}{k-2}$ moves.*

Let us look at two particular values.

If $k = \log(n) + 1$ the cost per item is $a = \log_{\log(n)}(n) = \frac{\log n}{\log \log n}$.

If $k = \sqrt{n} + 1$ the cost per item is $a = \log_{\sqrt{n}}(n)$, i.e., $(\sqrt{n})^a = n$ and $a = 2$. From the theorem we get the upper bound $3n - \sqrt{n}$ for the number of moves. This number of moves is also enough if we have one stack less, i.e., for $k = \sqrt{n}$: Split the elements into $\sqrt{n}$ blocks of size $\sqrt{n}$, when processing a block the smallest goes to the output and the others are intermediately placed in the $\sqrt{n} - 1$ other stacks. Hence $\sqrt{n}$ items are moved only twice and all others three times. Note that an additional smallest element 0 can be processed with two additional moves. When it arrives it is immediately moved to the output.

For completeness some word about the case $k = 2$. In this case sorting can be accomplished by keeping all items together in one block which is moved hence and forth between the two stacks. In each transfer of the block the smallest remaining element is directly moved to the output. Hence the size of the block is decreasing and the overall complexity is at most $\binom{n+1}{2}$.

2.2 Lower Bounds

For the lower bound we only consider permutations with the least element last. This property implies that all elements have to be inserted into the stacks before the first element can be moved to the output. This restriction is natural when the stacks model the store at some transportation hub. Following König and Lübbecke we refer to the restriction as the *midnight constraint*.

The idea for the lower bounds is to define an encoding for a sorting procedure. Different input permutations shall require differently encoded sorting procedures. Hence the number of different encodings of sorting procedures for n items must be at least $n!$. In the computations we use Stirling's formula $n! \approx (\frac{n}{e})^n \sqrt{2\pi n}$ to approximate $n!$.

A move of an item from stack i to stack j will be encoded as a pair (i, j). A move from the input to stack i or from this stack to the output will be encoded as (i, i). A sorting procedure is a sequence of moves, hence, a list of such pairs. Since we have k^2 pairs there are k^{2t} possible sequences that potentially encode a sorting with t moves. The inequality $k^{2t} \geq n! > (\frac{n}{e})^n$

yields $2t \geq n \log_k(n) - O(n)$, i.e., for n large t has to get arbitrarily close to $\frac{n\log(n)}{2\log(k)}$.

For $k \geq 3$ we thus obtain upper and lower bounds differing only by the small factor $\frac{2\log(k)}{\log(k-1)}$, that is by a factor ≤ 3.2.

Theorem 2. *The worst case complexity for sorting n elements in a network of $k \geq 3$ communicating stacks is at least $\frac{n}{2}\log_k(n) - O(n)$.*

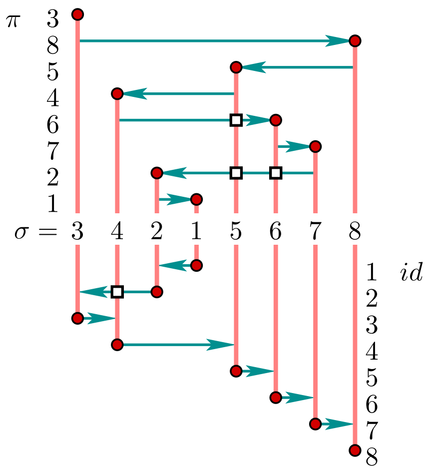
For $k = \sqrt{n}$ we can point to a specific permutation that maximizes the number of moves required. Consider the permutation $\pi = 1, 2, 3, \ldots, n, 0$. The element 0 at the end enforces the midnight constraint. Consider the position of the n items in the stacks right after inserting element 0. For a consecutive pair a below b on any of these stacks we either have $a < b$ and b has to be displaced before a can be output or we have $a > b$ and if $b \neq 0$ it has been moved after the arrival of a. Hence, there is a stack to stack move for all elements except 0 and the lowest of each stack. This gives a total of at least $3(n+1) - \sqrt{n} - 1 = 3n - \sqrt{n} + 2$ moves. Together with the sorting described in the previous subsection we have the proposition.

Proposition 1. *The worst case complexity for sorting $n+1$ elements in a network of $\sqrt{n}$ communicating stacks is precisely $3n - \sqrt{n} + 2$.*

2.3 Two Communicating Stacks

Again we assume the midnight constraint, i.e., the largest element comes last. Consider the position of the n elements in the two stacks at *midnight*, i.e., right before the first element is moved out. Imagine the two stacks horizontally sticked together top to top, this shows a permutation of all the elements, this is the midnight permutation σ of the process. Remarkably the pair (π, σ) uniquely describes a sorting $\pi \to id$ on the two stacks network. A good way of visualizing the process is to keep the stacks sticked together linearly from the beginning and to think of an operating head moving left and right over this linear structure, push and pop operations always take place at the position of the head. Figure 2 shows an example.

To encode the sorting process we describe the movement of the head between consecutive in- resp. out-moves. Such a movement is readily described by a direction $a \in \{\ell, r\}$ indicating whether the head moves left or right and a distance b for the move, clearly with $0 \leq b \leq n-1$. In total we have $2n$ such pairs (a_i, b_i), $i = 1, .., 2n$. Actually, there are only $2n-2$



in() push_L
in() push_R
in() push_R
in() push_L
pop_R push_L in() push_L
in() push_R
pop_L push_R pop_L push_R in() push_L
in() push_L

pop_L out()
pop_L out() pop_L push_R
pop_L out()
pop_R out()
pop_R out()
pop_R out()
pop_R out()
pop_R out()

Figure 2: Sorting the input permutation $\pi = 38546721$ via $\sigma = 34215678$. Time corresponds to the vertical axis, movements of the head are horizontal arrows. The four extra moves where an element has to switch between stacks are indicated by squares. On the right we give the sequence of stack operations for sorting π via σ.

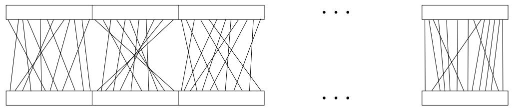


Figure 3: A typical pair of α-close permutations, each of the $n^{1-\alpha}$ blocks of length n^{α} is permuted independently.

movements of the head but such details disappear in the asymptotic analysis, hence, we will continue ignoring them. The total complexity of the sorting is $t = 2n + \sum_i b_i$. For a fixed t we may consider $(b_i)_i$ as a composition of the number $t - 2n$ with $2n$ parts. Therefore, there are at most $2^{2n}\binom{t}{2n}$ choices of $2n$ pairs (a_i, b_i) respecting the sum constraint. Sorting codes of different permutations have to be different, therefore, t has to be large enough for $2^{2n}\binom{t}{2n} \geq n!$.

If t satisfies $2^{2n}\binom{t}{2n} \geq n!$ then $2^{2n}\frac{t^{2n}}{(2n)!} \geq n!$, hence, $2^{2n}t^{2n} \geq \left(\frac{n}{e}\right)^n\left(\frac{2n}{e}\right)^{2n}$ and $t^{2n} \geq \left(\frac{n}{e}\right)^{3n}$. Taking the $2n$th root yields $t \geq c\, n^{3/2}$. This is already well above the $\Theta(n\log(n))$ complexity obtained for $k \geq 3$ stacks. With an additional idea we will squeeze more out of this approach.

The idea is that if π is effectively sortable via the midnight permutation σ and σ' is close to σ, then the cost of sorting π via σ' will not be much higher, hence, an effectively sortable π has many effective sortings. To make this precise we begin with a notion of closeness: Given a parameter $0 < \alpha < 1$ we say that σ and σ' are *α-close* if they have the same elements in the interval between positions $\lfloor p\, n^{\alpha}\rfloor$ and $\lfloor (p+1)\, n^{\alpha}\rfloor - 1$ for each $p \geq 0$. The concept is illustrated in Figure 3. For later use we note that the equivalence classes of α-closeness are of size $(n^{\alpha})!^{\,n/n^{\alpha}} = (n^{\alpha})!^{\,n^{1-\alpha}} \geq \left(\frac{n^{\alpha}}{e}\right)^n$.

Lemma 1. *If sorting π with midnight permutation σ requires at most $c\, n^{1+\alpha}$ moves and σ' is α-close to σ, then the sorting of π with midnight permutation σ' requires at most $(c+4)\, n^{1+\alpha}$ moves.*

Proof. By exchanging σ and σ' the distance of each of the $2n$ movements of the head can increase by no more than $2n^{\alpha}$. This adds up to no more than $4n^{1+\alpha}$. □

Being interested mainly in the exponent $1 + \alpha$ of the sorting complexity we may thus assume that every permutation has $(n^{\alpha})!^{\,n^{1-\alpha}}$ different sortings of this complexity. For the required number t of moves we need $2^{2n}\binom{t}{2n}/(n^\alpha)!^{\,n^{1-\alpha}} \geq n!$. This allows to estimate t as follows:

$$2^{2n}\frac{t^{2n}}{(2n)!} \geq n!\left(\frac{n^\alpha}{e}\right)^n \quad\Longrightarrow\quad 2^{2n}\,t^{2n} \geq \left(\frac{n}{e}\right)^n\left(\frac{2n}{e}\right)^{2n}\left(\frac{n^\alpha}{e}\right)^n$$

$$\Longrightarrow\quad t^{2n} \geq \left(\frac{n^{3+\alpha}}{e^4}\right)^n \quad\Longrightarrow\quad t \geq \frac{1}{e^2}\,n^{\frac{3+\alpha}{2}}$$

Any t satisfying the last inequality is in $\Omega(n^{1+\alpha})$, i.e., the additional moves for replacing a midnight permutation σ by an α-close σ' can be afforded. The bound for t holds for every $\alpha < 1$ we thus obtain:

Theorem 3. *The worst case complexity for sorting n elements in a network of two communicating stacks is at least $\Omega(n^{2-\epsilon})$ for all $\epsilon > 0$.*

3 General Networks

To avoid trivialities we assume that every node of a given network is contained in some directed $s-t$ path and that a stack-node never has a loop. In the first part of this section We identify two simple substructures $\mathcal{S}_1$, and $\mathcal{S}_2$ of networks that allow sorting with $O(n\log n)$ moves.

Networks avoiding the substructures $\mathcal{S}_1$ and $\mathcal{S}_2$ will be called *almost acyclic*. They have strongly connected components of very restricted type only. For such networks with a path containing r components that do not consist of a single node without loop we prove that sorting is possible in $O(n^{1+\frac{1}{r}})$ moves. If every $s-t$ path intersects at most r strong components, then there is a lower bound of $\Omega(n^{1+\frac{1}{2r}})$.

3.1 Strong Substructures

We first describe the two substructures allowing fast sorting. They are:

($\mathcal{S}_1$) Three stacks S_1, S_2 and S_3 and paths $p_1 : S_1 \to S_2$, $p_2 : S_2 \to S_3$ and $p_3 : S_3 \to S_1$.

($\mathcal{S}_2$) A queue Q, an a second node T, either stack or queue, with paths $p_1 : Q \to T$, $p_2 : T \to Q$ and in the case where T is a queue an additional path $q : Q \to Q$ that avoids T, q may be a loop. In the case where T is a stack the concatenation of p_1 and p_2 can replace q.

The analysis for case $\mathcal{S}_1$ is an obvious reduction to the situation with three communicating stacks analyzed in Section 2. Move all items from s to one of the stacks and then use the splitting scheme from Subsection 2.1. When an item has to be moved from a block on stack S_i to a block on stack S_j we move them along an appropriate concatenation of the paths p_1, p_2, p_3. This yields a sorting with $cn \log n$ moves, where c depends on the length of the paths p_i.

A sorting strategy for case $\mathcal{S}_2$ also uses blocks and splitting. The block B that has to be processed will be in the front of Q. Small elements from B are moved via q to the back of Q where the block B_- is created. Large elements are parked in the block B_+ on T. When the processing of B is complete the content of T is also moved to the back of Q. The start is with a single block consisting of all elements on B. A *round* is a period of time in which every element is moved into a new block. The size of the blocks is essentially halved in each round. When blocks have size 1 we have a completely sorted list on Q and are done. The complexity is $c\,n$ times the number of rounds, i.e., $c\,n \log n$.

Note that the argument preceding Theorem 2 applies to arbitrary networks with a constant number k of nodes and thus yields a general lower bound of order $\Omega(n \log n)$ for the worst case sorting complexity.

Proposition 2. *The sorting complexity on networks with a constant number k of nodes that contain a substructure of type $\mathcal{S}_1$ or $\mathcal{S}_2$ is $\Theta(n \log n)$.*

3.2 Almost Acyclic Networks

Networks avoiding the substructures $\mathcal{S}_1$ and $\mathcal{S}_2$ are called *almost acyclic*. Their strong components are either trivial, i.e., consisting of a single node without loop, or they consist of a simple cycle of queues, this may also be a single queue with a loop, or they consist of two communicating stacks.

Let us consider a network with an $s - t$ path containing k nontrivial strong components. Let $C_1, \ldots, C_k$ be the order of the components on the path. In the following description of a sorting procedure we again use the terminology of blocks. At the beginning all items form a single block on C_1. When a component is empty it may receive a new block from the preceding component. A block sent from C_i to C_{i+1} always consists of (approximately) a_i items, where $a_i = n^{1-\frac{i}{k}}$. When component C_i is non-empty and C_{i+1} is allowed to receive a block, then C_i looks at all items it holds and sends the a_i smallest. The numbers are set up such that C_k will send singletons to the output. Since at every moment of time, when two components C_i and C_j with $i < j$ are non-empty, all items on C_i are larger than any item on

C_j it follows that the process yields the sorted sequence at the output.

Every block received by C_i is of size $a_{i-1} = n^{\frac{1}{k}}\, a_i$ and every block sent by C_i is of size a_i. Therefore, each element is 'looked at' at most $n^{\frac{1}{k}}$ times within the component. The number of moves of an element in a component is proportional to the number of looks at it. This makes a cost of $O(n^{1+\frac{1}{k}})$ per component. This makes a total of $O(k\, n^{1+\frac{1}{k}})$ moves. For k constant this is $O(n^{1+\frac{1}{k}})$ while for $k \sim \log n$ it is $O(n \log n)$.

Theorem 4. *Almost acyclic networks with an $s-t$ path containing k non-trivial strong components can sort with $O(k\, n^{1+\frac{1}{k}})$ moves.*

In Subsection 3.3 we deal with a special class of almost acyclic networks where the result of the theorem is best possible. First, however, we go for a general lower bound. Again, the method of choice is to use an appropriate encoding.

We start considering a network consisting of a linearly arranged sequence of k strong components. Collapsing the strong components this reduces to a single $s-t$ path with k nodes. To encode a sorting consisting of t moves we first break the sequence at *transition moves*, i.e. moves where an element is transfered from one component to the next. We assume that the sorting is normalized in the sense that between a transition move bringing an item from C_i to C_{i+1} and the next transition move there are only moves within C_i and C_{i+1}. From the structure of the strong components it follows that for each component all that matters is the direction of the movement and the number of elements moved. Hence, we can encode the transition move with the index i of the component and the action on C_i and C_{i+1} by two bits b, b' and two numbers x and y. There is a total of $k\,n$ transition moves, hence, we get a sequence of kn encoding tuples $(i_j, b_j, b'_j, x_j, y_j)$. For the bits and the leading indices there are at most $(4\,k)^{kn}$ choices. The numbers satisfy $\sum_j x_j + y_j < t$, hence, there are at most $\binom{t}{2kn}$ possibilities for them.

The computation that follows is similar to what we did in Subsection 2.2. Requiring that t is large enough such that all input permutations consisting of n items can be sorted implies an inequality:

$$(4\,k)^{kn} \binom{t}{2kn} \geq n! \implies (4\,k)^{kn} \frac{t^{2kn}}{(2kn)!} \geq n! \quad \implies$$

$$(2\sqrt{k}\,t)^{2kn} \geq \left(\frac{n}{e}\right)^{n} \left(\frac{2kn}{e}\right)^{2kn} \implies t^{2kn} \geq \left(\frac{n}{e}\right)^{(2k+1)n} k^{kn} \quad \implies$$

$$\implies t \geq c\, n^{\frac{(2k+1)}{2k}}.$$

Consider an arbitrary almost acyclic network with a constant number k of strong components. There is 'only' a constant, say k^k, number of $s-t$ path in the network. Therefore, in a sorting of n items there is a path taken by linearly many of the items. Applying the previous consideration to this path we conclude:

Theorem 5. *An almost acyclic network containing a constant number k of strong components requires $\Omega(n^{1+\frac{1}{2k}})$ moves for sorting.*

In the following subsection we consider two special cases of almost acyclic networks. In both cases we can improve upon the lower bound of the theorem.

3.3 Sequences of Looped Queues and Doublestacks

In this subsection we consider almost acyclic networks consisting of a single $s-t$ path of k strong components. We investigate two particular instances:

(1) Each strong component is a cycle of queues or a single queue with a loop.

(2) Each strong component is a doublestack, i.e., a pair of communicating stacks.

Let NQ_k be the first of these instances, and let Z_i, $i = 1, .., k$, denote the ith cycle of queues along the path. The input permutation is $\pi = n, n-1, \ldots 2, 1$. It will be shown that a sorting of π in NQ_k requires at least $\Omega(n^{1+\frac{1}{k}})$ moves.

Consider a sorting procedure and associate a vector $q(x) \in \mathbb{N}^k$ with every number $x \in \{1, .., n\}$. The component $q_i(x)$ of this vector records the number of rounds item x makes on Z_i during the sorting. To account for the move of the element from Z_i to Z_{i+1} the actual value of $q_i(x)$ is one more than the number of rounds. Hence, in the case where each Z_i consists of a single queue with a loop, the total number of moves of the sorting is exactly $\sum_{i,x} q_i(x)$.

Observe that during a sorting for every given pair of numbers $x < y$ there is an i such that y is overtaken by x in Z_i, i.e., x arrives later in Z_i but leaves earlier, in particular $q_i(x) < q_i(y)$. This implies that the vectors $q(x)$ are pairwise different.

Lemma 2. *There is a constant c_k depending only on k such that*

$$\sum_{x=1}^{n} \sum_{i=1}^{k} q_i(x) \geq c_k \, n^{1+\frac{1}{k}}$$

for every set of n pairwise different vectors $q(x) \in \mathbb{N}^k$.

Before proving the lemma we shall point to its consequence. We get a lower bound matching the upper bound from Theorem 4:

Proposition 3. *Sorting the reverse permutation π on the network NQ_k consisting of k cycles of queues along a path requires $\Omega(n^{1+\frac{1}{k}})$ moves.*

Proof of the lemma. A set $\mathcal{Q}$ of n different positive vectors minimizing the sum has to be packed in the sense that there is a k-dimensional simplex $\Delta_k^0(r)$ spanned by $\mathbf{0}$ and the k vectors $r\,\mathbf{e}_i$ such that all integral points in the open interior of $\Delta_k^0(r)$ belong to $\mathcal{Q}$ and no point outside of $\Delta_k^0(r)$ belongs to $\mathcal{Q}$.

Every $q \in \mathcal{Q}$ is the maximal corner of a unit-cube C_q contained in $\Delta_k^0(r)$, therefore,

$$\frac{r^k}{k!} = \mathsf{Vol}_k(\Delta_k^0(r)) \geq n \quad \Longrightarrow \quad r \geq \frac{k}{e} \, n^{\frac{1}{k}}.$$

¿From $\sum_i q_i \geq \int_{C_q} (\sum_i x_i)\, dx$ for every $q \in \mathcal{Q}$ we get

$$\sum_{q \in \mathcal{Q}} \sum_i q_i \geq \int_{\Delta_k^0(r)} (\sum_i x_i)\, dx = \int_{t=0}^{\frac{r}{\sqrt{k}}} t \cdot \mathsf{Vol}_{k-1}(\Delta_{k-1}(t))\, dt,$$

where $\Delta_{k-1}(t)$ is the $(k-1)$-dimensional simplex spanned by the k vectors $t\,\mathbf{e}_i$, i.e., $\Delta_{k-1}(t)$ is a regular simplex with sidelength $\sqrt{2}t$. The volume of the regular k-dimensional simplex with sidelength one is $\frac{\sqrt{k+1}}{\sqrt{2}^k k!}$, hence, $\mathsf{Vol}_{k-1}(\Delta_{k-1}(t)) = \frac{\sqrt{k}}{(k-1)!}\, t^{k-1}$. Using this in the above integral and substituting for r yields the final inequalities:

$$\sum_{q \in \mathcal{Q}} \sum_i q_i \geq \frac{k\sqrt{k}}{(k+1)!} \left(\frac{r}{\sqrt{k}}\right)^{k+1} \geq \frac{k\sqrt{k}}{(k+1)!} \frac{k^{k+1}}{(e\sqrt{k})^{k+1}} \, n^{\frac{k+1}{k}} = c_k \, n^{1+\frac{1}{k}}.$$

□

Let NS_k be an almost acyclic networks consisting of a single $s-t$ path of k doublestacks. A lower bound for the number of moves required to sort on NS_k was given in Theorem 5. To improve upon this bound we use terminology and the idea from the proof of Theorem 3. We assume that the input permutation has the least element last, i.e., the midnight constraint is enforced. Consider the arrangement of items on doublestack i at midnight, this is a sequence σ_i. The concatenation of these sequences is a permutation σ split into k pieces $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_k)$, in the following we refer to such a permutation as *splitted (midnight) permutation.* Two splitted permutations σ and σ' are α-close if for all i: σ_i and σ'_i contain the same items and they are α-close in the old sense, i.e., they contain the same items in their buckets of length n^α. Equivalence classes of α closeness are of size at least $(n^\alpha)!^{\,n/n^\alpha - k}$. Since k is a constant we may still estimate this size as $(\frac{n^\alpha}{e})^n$.

Lemma 3. *If sorting π on NS_k with splitted midnight permutation σ requires at most $c\,n^{1+\alpha}$ moves and σ' is α-close to σ, then there is a sorting of π with splitted midnight permutation σ' that requires at most $(c{+}4{+}k)\; n^{1+\alpha}$ moves.*

Proof. The sorting with σ' reproduces the original sorting with σ as close as possible, i.e, the sequence of moves where elements are sent to the next doublestack are identical, moreover, if x is an element belonging to piece σ_j, then then position of x in the sequence of each doublestack i with $i \neq j$ is exactly the same in both sortings. Hence, all additional moves that are associated with x occur when x is inserted or removed from doublestack j. These additional moves are of two types:

- The head is passing an element y that belongs to the block of x, i.e., to the interval of length n^α on σ_j containing x. There are at most $2n^\alpha$ such moves associated with the insertion of x into doublestack j and again $2n^\alpha$ moves associated with the removal.
- The head is passing an element z belonging to a piece σ_i with $i \neq j$. For this to happen it must be that in one of the two sortings x is left of z and in the other it is right of z, i.e., the position of z is in the range spanned by elements the block of x. This kind of move is assigned to z. While sitting on doublestack i element z can cause at most n^α such moves. Altogether there are at most $k\,n^\alpha$ such moves assigned to z.

Summing over all x and z we can bound the number of additional moves by $n\,(4+k)\,n^\alpha$. □

Given the lemma we can redo the computation preceding Theorem 5:

$$(4\,k)^{kn}\binom{t}{2kn} \geq n!(\frac{n^\alpha}{e})^n \implies t \geq c\,n^{1+\frac{1+\alpha}{2k}}.$$

The choice of α is restricted by the condition that there is additional work of order $n^{1+\alpha}$. Hence we need $1+\alpha \leq 1+\frac{1+\alpha}{2k}$, i.e., the best we can do is to choose $\alpha = \frac{1}{2k-1}$. This yields the proposition:

Proposition 4. *Sorting n elements on the network NS_k consisting of k doublestacks along a path requires at least $\Omega(n^{1+\frac{1}{2k-1}})$ moves in the worst case.*

Conclusion

We have analyzed the sorting complexity of networks of stacks and queues. In most cases we could prove upper and lower bounds that are at least reasonably close. Some questions are left open or raised by our results. To us the single most intriguing problem is the following:

- Is it possible to sort on two communicating stacks with $o(n^2)$ moves?

One of the aspects where networks of queues and networks of stacks differ is that in the former case we could get lower bounds by analyzing a specific input permutation while in the second case we had to rely on counting arguments. It would be interesting to get hand on explicit permutations that are hard to sort on a given network of stacks, e.g., on NS_k.

A related line of questions is opened if we fix an input permutation and ask for the optimal sorting on a given network. As mentioned in the introduction some instances of the problem have been shown to be computationally hard by König et al. [7]. Again the case of the network consisting of two stacks seems to be challenging.

- Is it hard to compute an optimal sorting for an input permutation π on a network of two communicating stacks?
- Is it possible to approximate the sorting complexity of π on a network of two communicating stacks in polynomial time?

References

[1] N. Amato, M. Blum, S. Irani and R. Rubinfeld, *Reversing trains: a turn of the century sorting problem*, J. Alg., 10 (1989), 413–428.

[2] M. D. Atkinson, M. J. Livesy, and D. Tulley, *Permutations generated by token passing in graphs*, Theor. Comput. Sci., 178 (1997), 103–118.

[3] M. Bóna, *A survey of stack-sorting disciplines*, Electr. J. Combin., 9(2) (2003), A1, 16 pages.

[4] M. Bóna, *Combinatorics of Permutations*, Chapman & Hall, 2004.

[5] D. E. Knuth, *The Art of Computer Programming, Vol. 1*, Addison-Wesley, 1968. Third ed., updated and revised 1997.

[6] F. G. König and M. E. Lübbecke, *Sorting with Complete Networks of Stacks*, TU Berlin, Mathematik, Preprint 036-2007.

[7] F. G. König, M. E. Lübbecke, R. H. Möhring, G. Schäfer, and I. Spenke, *Solutions to real-world instances of Pspace-complete stacking*, in Proceedings ESA '07, vol. 4698 of Lecture Notes Comput. Sci., Springer-Verlag, 2007, 729–740.

[8] R. Tarjan, *Sorting using networks of queues and stacks*, J. Assoc. Comput. Mach., 19 (1972), 341–346.