RAIRO-Theor. Inf. Appl. 58 (2024) 13
https://doi.org/10.1051/ita/2024010

RAIRO - Theoretical Informatics and Applications
www.rairo-ita.org

SORTING WITH A POPQUEUE

Lapo Cioni and Luca Ferrari[*]

Abstract. We introduce a new sorting device for permutations, which we call *popqueue*. It consists of a special queue, having the property that any time one wants to extract elements from the queue, actually all the elements currently in the queue are poured into the output. We illustrate two distinct optimal algorithms, called `Min` and `Cons`, to sort a permutation using such a device, which allow us also to characterize sortable permutations in terms of pattern avoidance. We next investigate what happens by making two passes through a popqueue, showing that the set of sortable permutations is not a class for `Min`, whereas it is for `Cons`. In the latter case we also explicitly find the basis of the class of sortable permutations. Finally, we study preimages under `Cons` (by means of an equivalent version of the algorithm), and find a characterization of the set of preimages of a given permutation. We also give some enumerative results concerning the number of permutations having k preimages, for $k = 1, 2, 3$, and we conclude by observing that there exist permutations having k preimages for *any* value of $k \geq 0$.

Mathematics Subject Classification. 05A05, 68R05, 68W32.

Received December 12, 2022. Accepted February 23, 2024.

1. Introduction

Sorting algorithms for permutations using several kinds of containers have been extensively studied in the last decades. In particular, containers like stacks and queues turn out to be of particular interest. Some classical papers dealing with these matters are, for instance, [1–3], whereas some more recent ones are [4–7]. This is, however, only a very small fraction of the literature dedicated to sorting permutations with stacks and queues, which at present amounts to several dozens of articles.

The most classical and fundamental of these sorting algorithms is `Stacksort` [8], which attempts to sort a permutation by making use of a stack. One of the many modifications of the `Stacksort` algorithm uses a popstack in place of a stack, and was first studied in [9], where the authors also considered the case of several popstacks in series. We will refer to such an algorithm as `Popstacksort`. However, it took many years until Pudwell and Smith [10], and successively Claesson and Guðmundsson [11], started to investigate the case of two or more passes from a popstack, and explicitly provide the (rational) generating functions for the corresponding sortable permutations.

By replacing the stack with a queue (and also allowing the bypass of the queue) one obtains another interesting (and much less studied) sorting algorithm, usually called `Queuesort`. Much information on `Queuesort` can be found in [12], whereas a detailed analysis of the preimages under the associated operator has been recently

Keywords and phrases: Sorting algorithm, popqueue, enumeration, pattern avoidance, permutation.

Dipartimento di Matematica e Informatica “U. Dini”, University of Firenze, Firenze, Italy.

[*] Corresponding author: luca.ferrari@unifi.it

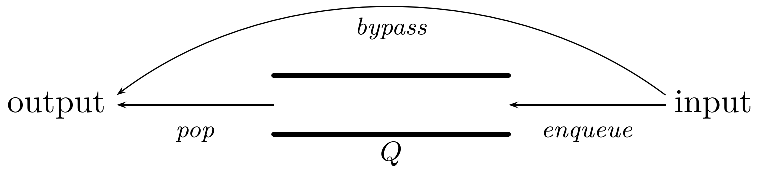

FIGURE 1. A popqueue.

pursued in [13, 14]. In the present paper we define a new sorting device that specializes `Queuesort` in the same way as `Popstacksort` specializes `Stacksort`. Our main results are a characterization and enumeration of permutations sortable with a popqueue, as well as the description of two optimal sorting algorithms, which are able to sort all sortable permutations but employ different strategies. We then investigate what happens by performing twice each of the two sorting algorithms, showing that the corresponding sets of sortable permutations are different (notably, in one case we get a class, whereas in the other case we do not). Finally, the last part of the paper is devoted to the study of preimages of one of our sorting algorithms, namely we provide a description of the set of all preimages of a given permutation as well as a few related enumeration results. We close the paper by giving some hints for further work.

2. Preliminary notions and results

A popqueue (see Fig. 1) is a data structure where we can insert and extract elements using a FIFO (*i.e.*, first in first out) policy, with the further requirement that the extraction operation consists of removing all the elements in the container. Thus a popqueue is a special type of queue, with the only difference that an extraction operation removes all the elements in the container instead of removing only the one in the front of the queue.

A popqueue can be used as a sorting device for permutations. Namely, the allowed operations are the following:

e: *enqueue*, insert the current element of the input permutation into the popqueue, in the last position;
p: *pop*, remove all the elements currently in the popqueue, from the front to the back, and send them into the output;
b: *bypass*, move the current element of the input permutation into the output.

Starting from an arbitrary input, several different permutations can be obtained, depending on the order in which the above operations are carried out. However, since we are interested in sorting a given input permutation, we will say that a permutation is *sortable* when there exists a sequence of operations that output the identity permutation. One of our main goals is to provide an *optimal sorting algorithm*, which is able to sort all sortable permutations. In designing such an algorithm, it will be important to define it also on nonsortable permutations; this will allow us to iterate it, in order to be able to study devices consisting of popqueues connected in series.

As it is usual in this framework, our sorting device can be conveniently described and studied by means of the notion of *pattern* of a permutation.

Given two permutations $\sigma = \sigma_1\sigma_2\cdots\sigma_k, \tau = \tau_1\tau_2\cdots\tau_n$, with $k \leq n$, we say that σ is a *pattern* of τ whenever there exist indices $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ such that σ is order-isomorphic to $\tau_{i_1}\tau_{i_2}\cdots\tau_{i_k}$ (*i.e.*, for all l, m, $\sigma_l < \sigma_m$ if and only if $\tau_{i_l} < \tau_{i_m}$). See Figure 2 for an example. If T is any set of permutations, then the set of all permutations avoiding each pattern of T is denoted $\mathrm{Av}(T)$, whereas $\mathrm{Av}_n(T)$ is the subset of $\mathrm{Av}(T)$ consisting of permutations of length n. The set S of all permutations of finite length can be partially ordered using the notion of pattern, by declaring $\sigma \leq \tau$ whenever σ is a pattern of τ. In the poset $(S, \leq)$, sets of the form $\mathrm{Av}(T)$, for some $T \subseteq S$, are called permutation *classes*, and it is easy to see that they are *downsets* (*i.e.* they are closed downwards). In what follows, we will also use the notation S_n to denote the set of all permutations of length n.

Thanks to the notion of pattern, we can provide a necessary condition for a permutation to be sortable.

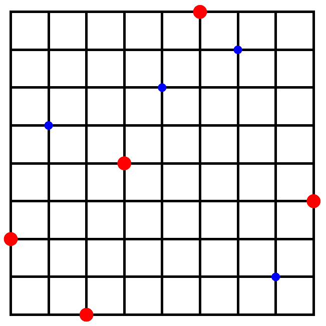

FIGURE 2. The permutation 361 579 824 contains the pattern 21 453 (in red).

Proposition 2.1. *If a permutation $\pi \in S$ contains an occurrence of the pattern* 321 *or of the pattern* 2 413 *(i.e., $\pi \notin \mathrm{Av}(321, 2\,413)$), then π is not sortable.*

Proof. Suppose that π contains an occurrence cba of the pattern 321, and suppose by contradiction that π is sortable. Then c must enter the popqueue, because otherwise it would reach the output before a. For the same reason, b has to enter the popqueue. As a consequence, c exits the popqueue before b, which is a contradiction. On the other hand, if the sortable permutation π contains an occurrence $bdac$ of the pattern 2413, when a is the current element in the sorting process, the elements b and d are both inside the popqueue (since a must reach the output before both of them). Thus b and d are popped from the popqueue at the same time, so they reach the output before c, which is again a contradiction (since π is assumed to be sortable). □

3. Two optimal sorting algorithms

We now provide two different sorting algorithms which, although rather similar, have a noticeably different behavior when applied twice. The first algorithm, called `Min`, takes as input a permutation $\pi = \pi_1\pi_2\cdots\pi_n$ and is described below (Alg. 3.1). In the description, Q denotes the popqueue, $Front(Q)$ is the current first element of the queue and $Back(Q)$ is the current last element of the queue.

Algorithm 3.1 `Min`
```
$Q := \emptyset$;
for $i = 1, \dots n$ do
    if Front($Q$) $< \pi_j$, for all $j \geq i$ then
        execute $p$;    execute $e$;
    else if Back($Q$) $< \pi_i$ then
        execute $e$;
    else if Front($Q$) $> \pi_i$ then
        execute $b$;
    else
        execute $p$;    execute $e$;
execute $p$;
```

The reason for the name `Min` comes from the fact that, whenever the first element of the popqueue is the next one to output (*i.e.*, it is the minimum element not already in the output), the algorithm pours the whole content of the popqueue into the output. If this is not the case, the current element of the input is enqueued, provided that it is larger than the element in the back of the popqueue; otherwise the smallest between the

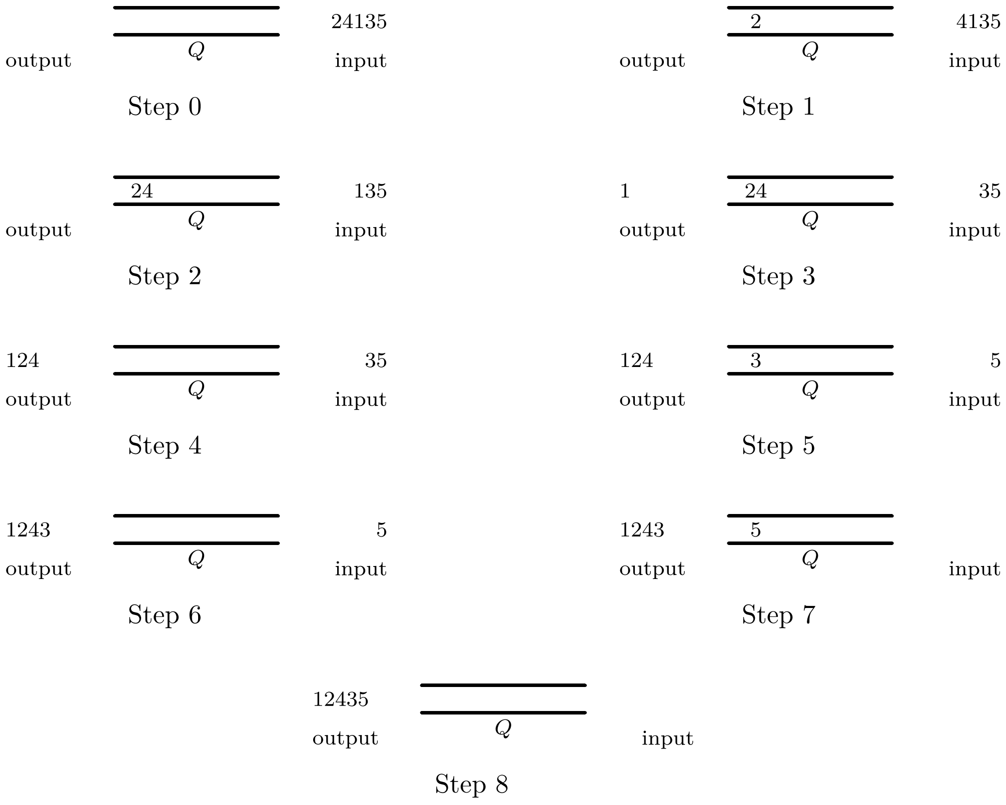

FIGURE 3. Performing the algorithm `Min` on the (unsortable) permutation 24 135.

current element and the element in the front of the popqueue is output (of course, in the latter case the whole content of the popqueue reaches the output). An example of the execution of `Min` is given in Figure 3.

`Min` is an optimal sorting algorithm, as it is shown in the next proposition.

Proposition 3.1. *Let $\pi \in S_n$ and denote with $Min(\pi)$ the output of Min with input π. Then $Min(\pi) \neq id_n$ if and only if $321 \leq \pi$ or $2\,413 \leq \pi$.*

Proof. By Proposition 2.1, we know that a permutation containing 321 or 2 413 is not sortable.

On the other hand, suppose that $\texttt{Min}(\pi) \neq id_n$, and consider the first configuration in which an incorrect element reaches the output (here "incorrect" means that it is not the minimum element not already in the output). If such an element comes to the output after executing a bypass operation, then we call it b and we observe that it must be smaller than the front element c of the queue. Moreover, there must be an element $a < b$ not yet in the output, otherwise b would be the correct element to output. Since (the content of) the queue is increasing by construction, then a must be in the input, hence π contains an occurrence cba of the pattern 321.

Otherwise, if the first incorrect element goes through the queue and reaches the output after executing a pop operation, then we have two possibilities. In the first case, the first element b of the queue is the smallest element not yet in the output, but the sorting procedure fails because there are elements d inside the queue and c in the input such that $d > c$. In such a case, the last element which has reached the output is $b - 1$, and this necessarily happened after a bypass operation when b and d were already in the queue (otherwise b would be already in the output). Therefore the elements b, d, $b-1$, c form an occurrence of the pattern 2 413 in π. In the second case, if the minimum element a not yet in the output is not the first element of the queue, then a must

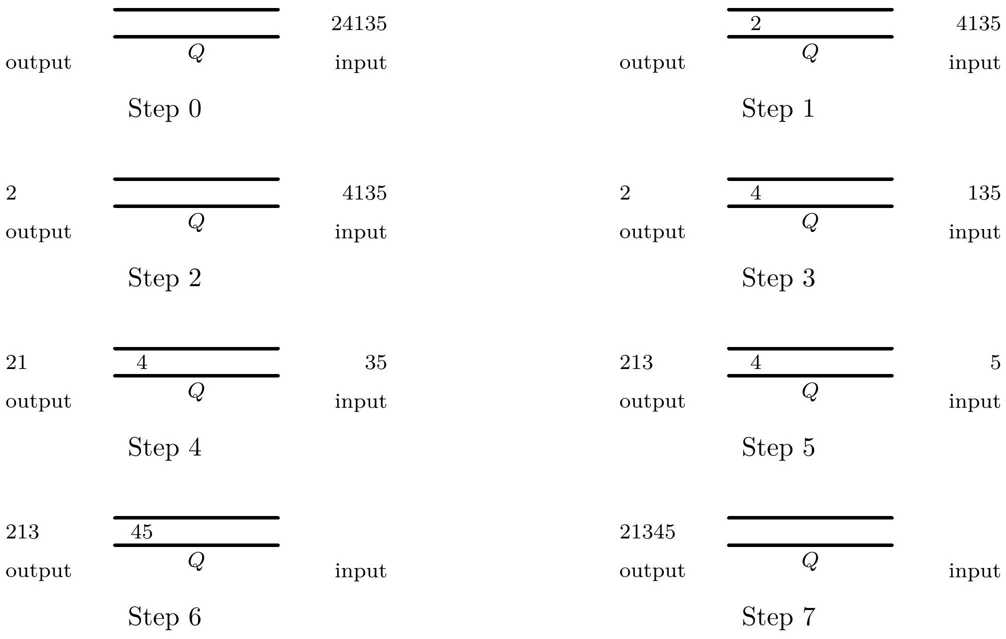

FIGURE 4. Performing the algorithm `Cons` on the (unsortable) permutation 24 135.

still be in the input, and the last element c of the queue must be larger than the current element b of the input (which is in turn larger than a). Therefore π contains an occurrence cba of the pattern 321. $\square$

The second algorithm we describe is `Cons`, which again takes as input a permutation $\pi = \pi_1\pi_2\cdots\pi_n$. The notations are the same as in the description of `Min`.

Algorithm 3.3 (`Cons`).

```
$Q := \emptyset$;
**for** $i = 1, \dots n$ **do**
    **if** $\pi_i = \mathrm{Back}(Q) + 1$ **then**
        execute $e$;
    **else if** $\mathrm{Front}(Q) > \pi_i$ **then**
        execute $b$;
    **else**
        execute $p$;    execute $e$;
execute $p$;
```

The name `Cons` comes from the first instruction, which forces the content of the queue to consist of consecutive elements during the whole execution. An example of the execution of `Cons` is given in Figure 4.

As it happened for `Min`, also `Cons` is an optimal sorting algorithm.

Proposition 3.2. *Let $\pi \in S_n$ and denote with* `Cons`(π) *the output of* `Cons` *with input π. Then* `Cons`$(\pi) \neq id_n$ *if and only if* $321 \leq \pi$ *or* $2\,413 \leq \pi$.

Proof. We already know, by Proposition 2.1, that a permutation containing either the pattern 321 or the pattern 2 413 is not sortable. On the other hand, suppose that `Cons`$(\pi) \neq id_n$, and consider the first configuration in which an "incorrect" element reaches the output. If the incorrect element b comes directly from the input, by performing a bypass operation, then an argument completely analogous to that used in Proposition 3.1 shows that π contains the pattern 321.

Otherwise, if the first incorrect element goes to the output after a pop operation, then let b be the last element of the queue, $d > b+1$ be the current element of the input and a be the smallest element not yet in the output. Then $b+1$ and a must still be in the input, because Q consists of consecutive elements and no incorrect element reached the output yet. So either π contains the occurrence $bda(b+1)$ of the pattern 2 413, or π contains the occurrence $d(b+1)a$ of the pattern 321. □

We thus have two different optimal sorting algorithms, which follow distinct heuristics to sort π. Although the set of sortable permutations is the same, the output of `Min` and `Cons` is different in general. For instance, `Min`$(2\,413) = 1\,243$ and `Cons`$(2\,413) = 2\,134$. We remark that both heuristics are somehow reasonable in the framework of permutation sorting. First of all, neither `Min` nor `Cons` create new inversions. Moreover, if the first element of the queue is the minimum element among those not yet in the output, then it is reasonable to pop the queue immediately, as `Min` does, because the same thing would certainly be done before performing any bypass operation. On the other hand, if we were to allow non-consecutive elements in the queue, then the output would certainly not be the identity permutation, so it is reasonable to impose that the elements inside the queue are consecutive, as `Cons` does. Both ideas give us optimal algorithms, whose outputs differs only for nonsortable permutations, although the order in which the single operations are executed may be different even for sortable permutations. One could be tempted to see what happens when using both heuristics, by designing an algorithm that only allows consecutive elements inside the queue, and (at the same time) pops the queue whenever its front element is the next one to be output. However, the resulting algorithm would actually be equivalent to `Cons`, because the output would be the same, although operations would be performed in a different order in general, by prioritizing pop operations over enqueue ones.

As a consequence of the results of this section, we thus have that the set of sortable permutations of length n is $\mathrm{Av}_n(321, 2\,413)$. The enumeration of such a class is known [15]: this is the sequence of even-indexed Fibonacci numbers (sequence A001519 in [16]), whose first terms are $1, 1, 2, 5, 13, 34, \ldots$.

We close this section by proving some properties of `Cons` that will be useful later. Given a permutation $\pi = \pi_1 \cdots \pi_n$, an element π_i is called a *left-to-right maximum* (briefly, *LTR maximum*) when it is greater than π_j, for all $j < i$.

Lemma 3.3. *Let $\pi = \pi_1 \cdots \pi_n \in S_n$. When performing* `Cons` *on π, π_i enters the queue if and only if it is a LTR maximum. Moreover, the relative order of the non-LTR maxima of π is preserved in* `Cons`(π).

Proof. The proof is by induction on the number of steps during the execution of `Cons`. The first element π_1 is always a LTR maximum, and is in fact enqueued. Now suppose by induction that, at some point, all the elements currently enqueued are LTR maxima, and none of the LTR maxima has not been enqueued. We will prove that this remains true after `Cons` has executed its next instruction. Let π_i be the current element of the input. If $\pi_i = Back(Q) + 1$, then π_i is greater than $Back(Q)$ and thus it is greater than all the previous LTR maxima, because they all entered the queue in increasing order by the induction hypothesis. Therefore π_i is a LTR maxima as well, and is in fact enqueued by `Cons`. If instead π_i is smaller than $Front(Q)$, then it is not a LTR maximum by definition, and it bypasses the queue. Finally, if $\pi_i > Back(Q) + 1$, then π_i is a LTR maximum by the same argument as for the case $\pi_i = Back(Q)$, and it is in fact enqueued (after having popped the queue).

Finally, if a and b are not LTR maxima of π, they do not enter the queue, hence they keep their relative order in the output. □

This lemma does not hold for `Min`. Indeed, although every LTR maximum does enter the queue, some non-LTR maxima may also enter it (for example, the element 4 in 25 143 enters the queue). It is interesting to note that the optimal sorting algorithm that uses a classical queue (known as `Queuesort`, see [13]) has the same properties described in the above lemma.

4. Two passes through a popqueue

As we have already observed, `Min` and `Cons` are both optimal sorting algorithms, even if they exploit different strategies. In particular, the images of *unsortable* permutations are different, and this is clearly relevant when we run each of the two algorithms multiple times.

In this section we investigate permutations which are sortable by running each of the previous algorithms twice. This has been done for `Stacksort` by West [3], who found that the set of sortable permutations is not a class; nevertheless it can be described in terms of the avoidance of a pattern and a barred pattern.

We start by defining the sets $Sort_M^{(k)}$ and $Sort_C^{(k)}$ of permutations sortable by k applications of `Min` and `Cons`, respectively, that is $Sort_M^{(k)} = \{\pi \in S \mid \texttt{Min}^k(\pi)) = id\}$ and $Sort_C^{(k)} = \{\pi \in S \mid \texttt{Cons}^k(\pi)) = id\}$. In the previous section we have shown that $Sort_M^{(1)} = Sort_C^{(1)} = \mathrm{Av}(321, 2\,413)$. It is interesting to notice that $Sort_M^{(2)}$ and $Sort_C^{(2)}$ are instead unrelated from the point of view of set containment. Indeed, consider the permutations $2\,431$ and $35\,214$. We have $\texttt{Min}(\texttt{Min}(2\,431)) = 1\,243$ and $\texttt{Cons}(\texttt{Cons}(2431)) = 1\,234$, hence $2\,431 \in Sort_C^{(2)} \setminus Sort_M^{(2)}$. On the other hand, $\texttt{Min}(\texttt{Min}(35\,214)) = 12\,345$ and $\texttt{Cons}(\texttt{Cons}(35\,214)) = 21\,345$, hence $35\,214 \in Sort_M^{(2)} \setminus Sort_C^{(2)}$. This shows that $\texttt{Min}^2$ is capable of sorting permutations that $\texttt{Cons}^2$ is unable to sort, and *vice versa*.

What we show next is that $Sort_M^{(2)}$ and $Sort_C^{(2)}$ are not just distinct sets; they also have different features from the point of view of pattern containment. More precisely, $Sort_C^{(2)}$ is a permutation class, whereas $Sort_M^{(2)}$ is not.

Proposition 4.1. *The set $Sort_M^{(2)}$ is not a permutation class.*

Proof. Given the permutation $241\,653$, we have that $\texttt{Min}(\texttt{Min}(241\,653)) = 123\,456$, however $\texttt{Min}(\texttt{Min}(2\,431)) = 1\,243$, and clearly $2\,413 \leq 241\,653$. □

Proposition 4.2. *Let $\pi \in S_n$ be such that π contains one of the following patterns:*

- $4\,321$;
- $35\,241$;
- $35\,214$;
- $52\,413$;
- $25\,413$;
- $246\,153$;
- $246\,135$;
- $426\,153$;
- $426\,135$.

Then $\texttt{Cons}(\texttt{Cons}(\pi)) \neq id_n$.

Proof. By using the facts that

- the relative order of non-LTR maxima of π is preserved in $\texttt{Cons}(\pi)$ (see Lem. 3.3), and
- non-inversions in π remain non-inversions in $\texttt{Cons}(\pi)$,

as well as the definition of `Cons`, which requires the elements in the queue to be consecutive at all times, it is not difficult to show that, if π contains one of the patterns $4\,321$, $35\,241$, $35\,214$, $52\,413$, $25\,413$, $246\,153$ or $246\,135$, then $\texttt{Cons}(\pi)$ contains the pattern 321 or the pattern $2\,413$, therefore $\pi \notin Sort_C^{(2)}$.

We now consider the remaining patterns. Suppose that $dbfaec$ is an occurrence of the pattern $426\,153$ in π. We start by observing that d precedes a in $\texttt{Cons}(\pi)$ since, when f is processed, d must either be already in the output or exit the queue (because e is still in the input and $d < e < f$). Now, recalling the above facts, if d precedes b in $\texttt{Cons}(\pi)$ then dba is an occurrence of 321 in $\texttt{Cons}(\pi)$, otherwise $bdac$ is an occurrence of $2\,413$ in $\texttt{Cons}(\pi)$. Therefore in both cases $\pi \notin Sort_C^{(2)}$. An occurrence of the pattern $426\,135$ can be dealt with in a similar way. □

Proposition 4.3. *Let $\pi \in S_n$ be such that* $\texttt{Cons}(\texttt{Cons}(\pi)) \neq id_n$. *Then π contains one of the following patterns:*

- $4\,321$;
- $35\,241$;
- $35\,214$;
- $52\,413$;
- $25\,413$;
- $246\,153$;
- $246\,135$;
- $426\,153$;
- $426\,135$.

Proof. Since $\texttt{Cons}(\texttt{Cons}(\pi)) \neq id_n$, we know that $321 \leq \texttt{Cons}(\pi)$ or $2\,413 \leq \texttt{Cons}(\pi)$. We consider the two cases separately.

If $321 \leq \texttt{Cons}(\pi)$, then $321 \leq \pi$, since $\texttt{Cons}$ does not produce new inversions. Let c, b, a be elements forming an occurrence of 321 in π. If c is not a LTR maximum, then of course π contains an occurrence of $4\,321$. Otherwise, the LTR maximum c enters the queue, but it has to reach the output before b. This happens precisely when the current element e of the input is larger than $Back(Q)+1$ (and so also $e > c$). Now observe that the element $d = Back(Q)+1$ cannot appear before c in π (since c is a LTR maximum), nor between c and e in π (otherwise, when e is the current element of the input, the last element of the queue could not be $d-1$). Therefore, all possible ways in which the elements a, b, c, d, e can occur in π are $cedba$, $cebda$ e $cebad$, which form occurrences of the patterns $4\,321$ (ignoring c), $35\,241$ and $35\,214$, respectively.

Otherwise, suppose that $\texttt{Cons}(\pi)$ contains the pattern $2\,413$, and the elements b, d, a, c form such an occurrence. Since $\texttt{Cons}$ does not create new inversions, in the permutation π the element d appears before a and c, which are then not LTR maxima. Therefore π contains the elements d, a, c precisely in this order. Since b has to precede a in π, we are thus left with two possible configurations, which are $dbac$ and $bdac$.

If π contains the subword $dbac$, then necessarily d is a LTR maximum of π (otherwise b could not reach the output before d), and so in particular d enters the queue at some point. Moreover, we need a to reach the output after both d and b. In order to understand how this can be possible, we focus on the instant in which d (and so the entire content of the queue) is popped. In such a situation, b has to be already in the output, d is in the queue, and the current element of the input, call it f, is larger than $Back(Q)+1$ (and so also than d). Now observe that $e = f-1$ is necessarily in the input at this moment: in fact it cannot be inside the queue (since the content of the queue is increasing) and it cannot be already in the output, otherwise it would appear before d in π and so d would not be a LTR maximum. Summing up, we have that π must contain one among the subwords $dbfeac$, $dbfaec$, $dbface$, corresponding to the patterns $25\,413$ (ignoring d), $426\,153$, $426\,135$.

If instead π contains the subword $bdac$, then d may or may not be a LTR maximum. In case it is, then we can use an argument completely analogous to the one of the previous case, getting that π has to contain one among the patterns $25\,413$, $246\,153$, $246\,135$. On the other hand, if d is not a LTR maximum of π, then there is an element $e > d$ preceding d in π, hence π contains either $bedac$ or $ebdac$, corresponding to patterns $25\,413$ or $52\,413$. □

The first terms of the sequence counting $Sort_C^{(2)}$ with respect to the length are $1, 1, 2, 6, 23, 99, 445, 2\,029, 9\,292, 42\,608, 195\,445, \ldots$, and do not appear in [16]. We also observe that analogous numbers can also be computed for $Sort_M^{(2)}$, and the first values are $1, 1, 2, 6, 22, 89, 379, 1\,660, 7\,380, 33\,113, 149\,059, \ldots$. From these data, it appears reasonable to conjecture that, for any $n \geq 3$, the number of permutations of length n sortable by $\texttt{Min}^2$ is smaller than the number of permutations of length n sortable by $\texttt{Cons}^2$.

5. Preimages

In this section we will study the preimages of a generic permutation under $\texttt{Cons}$. Note that we might also study the preimages under $\texttt{Min}$, and that they are (in general) different. We choose $\texttt{Cons}$ since it has the property that all LTR maxima, and no other element, enter the queue during the sorting process. This property is shared with `Queuesort`, and is crucial to describe a simple procedure to find all the preimage of a generic permutation. For brevity, in the sequel we just call "preimages" the preimages under `Cons`.

We start by giving a different description of `Cons`, which highlights how it behaves on the LTR maxima of a permutation. Given two elements of a permutation, we say that they are *consecutive* when their values differ by 1, whereas we say that they are *adjacent* when their positions differ by 1. Such a notation will be kept for the rest of the paper.

Let $\pi = \pi_1 \cdots \pi_n \in S_n$. Mark π_1, and repeat the following steps until there are no marked elements:

- if there are no elements to the right of the (necessarily unique) block of adjacent marked elements, then unmark all marked elements;
- otherwise, compare the rightmost element μ of the block of adjacent marked elements with the element α to its right:
  - if $\mu > \alpha$, then swap α with the entire block of marked elements;
  - if $\mu = \alpha - 1$, then mark α;
  - if $\mu < \alpha - 1$, then mark α, and unmark all other elements of π.

As an example, we illustrate how this procedure operates on the permutation $\pi = 3\,241\,687$. The marked elements are indicated in **bold**.

$$\mathbf{3}\,241\,576 \to 2\,\mathbf{3}41\,576 \to 2\,\mathbf{34}1\,576 \to 2\,1\mathbf{34}\,576 \to 2\,1\mathbf{34}\,\mathbf{5}76 \to 2\,134\,5\mathbf{7}6 \to 2\,134\,56\mathbf{7} \to 2\,134\,567$$

Notice that, since an element is marked if and only if it is greater than the previous marked element, this procedure marks precisely the LTR maxima of the permutation (*i.e.* the elements that enter the queue). During the execution they are moved to the right until they reach the next LTR maximum; when this happens, they are glued together and continue moving to the right if and only if they are consecutive. This means that the block of marked elements consists of elements that are both adjacent and consecutive.

Notice that our marking mimics precisely what happens in the queue during the execution of `Cons`: the block of marked elements consists of the elements in the queue, the elements on its left are in the output and those on its right are still in the input. Moreover, with the notations used in the above procedure, the case $\mu > \alpha$ refers to a bypass operation, the case $\mu = \alpha - 1$ refers to an enqueue operation, and the case $\mu < \alpha - 1$ refers to a pop operation.

From now on, when we refer to the execution of `Cons` on a permutation, we consider this alternative description.

Proposition 5.1. *Let σ be a permutation and set $\pi = \mathtt{Cons}(\sigma)$. Then the last element of π is n. Also, denoting with $LTR(\tau)$ the set of the LTR maxima of a permutation τ, we have $LTR(\sigma) \subseteq LTR(\pi)$.*

Proof. During the execution of `Cons` on σ, n is going to be marked, because it is a LTR maximum, and it reaches the end of the permutation, since there are no greater elements that can block it.

Now, let μ be a LTR maximum of σ and suppose, by contradiction, that μ is not a LTR maximum of π. This means that there exists $\mu' > \mu$ to the left of μ in π. Since `Cons` does not create new inversions, we have that μ' is to the left of μ also in σ, which contradicts the fact that μ is a LTR maximum of σ. $\square$

In view of the previous proposition, we can look for the preimages of a given permutation π by looking at all the subsets of $LTR(\pi)$ and, for each of them, listing all the possible preimages with the prescribed set of LTR maxima. Notice that it is possible for a permutation to have no preimage (as well as many preimages) for a given subset. For example, there are no preimages of 213 whose LTR maxima are both 2 and 3, while $3\,421$ and $3\,214$ are both preimages of $2\,134$ whose LTR maxima are 3 and 4. To describe preimages we introduce the notion of a *mix* of two sequences.

Definition 5.2. Let $L = l_1 \cdots l_p$ and $A = a_1 \cdots a_r$ be two sequences of positive integers, such that the l_i's and a_j's are all different. We say that a sequence $m_1 \cdots m_{p+r}$ is a *mix* of L and A if it contains both L and A as subsequences, and $m_1 = l_1$. Define $Mix(L, A)$ as the set of all the mixes of L and A.

Essentially, a mix of two sequences L and A is a special shuffle of the two sequences, in which the first element belongs to L.

For example, the mix of 245 and 13 is the set $\{24\,513, 24\,153, 24\,135, 21\,453, 21\,435, 21\,345\}$.

We are now ready to describe the set $\texttt{Cons}^{-1}(\pi)$ of the permutations whose output under $\texttt{Cons}$ is π.

Proposition 5.3. *Let $\pi \in S_n$ be a permutation ending with n. Let $B \subseteq LTR(\pi)$ such that $n \in B$.*

If B contains two consecutive integers that are not adjacent in π, then there are no preimages of π whose set of LTR maxima is B.

Otherwise, we can write π as $\pi = A_1L_1A_2L_2 \cdots A_kL_k$, where the blocks L_i are maximal sequences of consecutive elements of B. The blocks A_i contain the remaining elements of π, and may be empty. Then, all the preimages of π whose set of LTR maxima is B are those of the form $\rho = \rho_1\rho_2 \cdots \rho_k$, with $\rho_i \in Mix(L_i, A_i)$, for every $i = 1, \dots, k$.

Proof. Let π be a permutation ending with n and $B \subseteq LTR(\pi)$ such that $n \in B$.

Suppose that $\mu, \mu + 1 \in B$ are not adjacent in π. Suppose that there exists a preimage σ of π such that $LTR(\sigma) = B$. We show that $\texttt{Cons}(\sigma) \neq \pi$, thus obtaining a contradiction. During the execution of $\texttt{Cons}$, μ is marked and moves to the right. Since $\mu + 1$ is also a LTR maximum of σ, μ will reach it and $\mu + 1$ will be marked while μ remains marked. Therefore both μ and $\mu + 1$ will be unmarked at the same time, and will be adjacent in the output. Since μ and $\mu + 1$ are not adjacent in π, $\texttt{Cons}(\sigma) \neq \pi$, which is a contradiction. Thus π has no preimages whose set of LTR maxima is B.

On the other hand, if B contains no consecutive elements which are not adjacent in π, let ρ, L_i and A_j be as in the statement of the proposition, and let λ_i be the first element of L_i, for every $i = 1, \dots, k$. We want to prove that ρ is a preimage of π. We start by proving that $\texttt{Cons}(\rho_i) = A_iL_i$. This is in fact true, because the elements of L_i are LTR maxima of π, and are therefore greater than the elements of A_i. By definition of mix, the first element of ρ_i is λ_i, therefore the elements of A_i are not LTR maxima of ρ_i. Finally, the elements of L_i are consecutive, therefore $\texttt{Cons}$ will move all of them together to the end of the string, thus obtaining A_iL_i. Now we just need to notice that the elements of different L_i are not consecutive to obtain that, during the execution of $\texttt{Cons}$ on ρ, all the elements of each L_i will be unmarked before marking the elements of L_{i+1}, hence $\texttt{Cons}(\rho) = A_1L_1 \cdots A_kL_k = \pi$.

To conclude, we need to prove that every preimage σ of π such that $LTR(\sigma) = B$ is of the form $\rho_1 \cdots \rho_k$, for some mixes ρ_i of L_i and A_i. First of all, the elements of σ that are not LTR maxima must be in the same relative order as they are in π, since $\texttt{Cons}$ does not change their relative positions. So, reading such elements in σ from left to right, we get the string $A_1 \cdots A_k$. The same argument can be used for the LTR maxima of σ, thus getting that σ contains the subsequence $L_1 \cdots L_k$. The first LTR maximum is λ_1, and it must be the first element of σ. Furthermore, all the elements of A_1 must be to the left of λ_2 in σ. Indeed, since all the elements of L_1 are consecutive, and λ_2 is strictly larger than the last element of L_1 plus 1, then during the execution of $\texttt{Cons}$ the elements of L_1 are marked and move to the right, but they will be unmarked as soon as λ_2 is reached. Therefore the elements of L_1 cannot shift to the right of λ_2 and all the elements of A_1 must be to their right, otherwise $\texttt{Cons}(\sigma)$ would not be the permutation $\pi = A_1L_1 \cdots A_kL_k$. Finally, all the elements of A_2 must be to the right of λ_2, because otherwise the last element of L_1 would shift to the right of them and the image of σ would not have all the elements of A_2 to the right of L_1. Therefore σ can be written as ρ_1S, where ρ_1 a mix of L_1 and A_1.

Iterating this argument, we obtain that a mix ρ_2 of L_2 and A_2 must follow ρ_1 in σ, and so on, until we obtain $\sigma = \rho_1 \cdots \rho_k$. □

By Proposition 5.1, we know that the LTR maxima of any preimage of π must be a subset of $LTR(\pi)$ containing n. Therefore the previous proposition describes all the preimages of a generic permutation.

For example, given $\pi = 3\,245\,617$, if we select the subset $B = \{4, 5, 7\}$, we have that the corresponding preimages are $4\,532\,761$, $4\,352\,761$, $4\,325\,761$. If we look at all the subsets of $LTR(\pi) = \{3, 4, 5, 6, 7\}$ (containing 7 and without consecutive elements not adjacent in π) we obtain the following permutations, which are all the preimages of π:

$\{7\}$: $7\,324\,561$,
$\{3, 7\}$: $3\,724\,561$,
$\{4, 7\}$: $4\,327\,561$,
$\{5, 7\}$: $5\,324\,761$,
$\{3, 5, 7\}$: $3\,524\,761$,
$\{4, 5, 7\}$: $4\,532\,761$, $4\,352\,761$, $4\,325\,761$.

5.1. Enumerative results

The correspondence between subsets of $LTR(\pi)$ and preimages of π would be particularly simple for permutations with no consecutive LTR maxima. Indeed, in that case there would be only one preimage for every legal subset. Unfortunately this never happens, because a permutation has preimages if and only if it ends with its maximum n, hence $n-1$ is always a LTR maximum. Nonetheless we still have a simple case, for which we are able to count preimages.

Proposition 5.4. *Let $\pi \in S_n$ be a permutation ending with n, such that the only consecutive elements in $LTR(\pi)$ are $n-1$ and n. Suppose that $n-1$ and n are not adjacent in π, and let $k = |LTR(\pi)|$. Then $|\texttt{Cons}^{-1}(\pi)| = 2^{k-2}$ and there is a bijection between $\texttt{Cons}^{-1}(\pi)$ and the subsets of $LTR(\pi)$ containing n but not $n-1$.*

Proof. Referring to Proposition 5.3, let B be a subset of $LTR(\pi)$ containing n. If $n-1 \in B$, then there are no preimages of π whose LTR maxima are the elements of B, since $n-1$ and n are not adjacent in π. Otherwise, if $n-1 \notin B$, then we express $\pi = A_1L_1 \cdots A_kL_k$ as in Proposition 5.3, noting that the blocks L_i consist of just one element (call it λ_i). Therefore there exists precisely one mix between L_i and A_i, for every i, which provides exactly one preimage. Thus, we have a correspondence mapping each subset of $LTR(\pi)$ containing n and not containing $n-1$ into a preimage of π. Since the preimages are all different, we have a bijection between preimages of π and subsets of $LTR(\pi)$ with the required properties. As a consequence, the number of preimages of π is the number of subsets of $LTR(\pi)$ of size k, containing n and not containing $n-1$, which is 2^{k-2}. □

Notice that the case described in the previous proposition gives a result analogous to the general result concerning the enumeration of the preimages of a permutation under `Bubblesort` [17]. Specifically, the number of preimages under `Bubblesort` of a permutation π ending with n and having k LTR maxima is 2^{k-1}. In fact, there is one preimage for every subset of $LTR(\pi)$ containing n, because (in the case of `Bubblesort`) the LTR maximum $n-1$ does not give troubles and can be selected. On the other hand, if there are consecutive elements in $LTR(\pi)$ other than $n-1$ and n, the analogy fails and there does not seem to be any link between the two situations.

To conclude, we provide some results concerning permutations with a given number of preimages. Define $c_n^{(k)}$ as the number of permutations of length n which have exactly k preimages under `Cons`. We already noticed that a permutation has preimages if and only if it ends with its maximum, therefore $c_n^{(0)} = (n-1)(n-1)!$, for every $n \geq 1$, and $c_0^{(0)} = 0$.

The next propositions deal with permutations having exactly 1, 2 or 3 preimages.

Proposition 5.5. *Let $\pi = \pi_1 \cdots \pi_n \in S_n$. Then π has exactly one preimage under* `Cons` *if and only if $\pi_n = n$ and $\pi_1 = n-1$, for every $n \geq 3$.*

Therefore $c_n^{(1)} = (n-2)!$ for every $n \geq 3$, and $c_0^{(1)} = 1$, $c_1^{(1)} = 1$, $c_2^{(1)} = 0$.

Proof. If $\pi_n = n$ and $\pi_1 = n-1$ then π has one preimage by Proposition 5.4.

On the other hand, suppose that $\pi_n \neq n$ or $\pi_1 \neq n-1$. If π does not end with n, then it has no preimages. Otherwise, we have that $\pi_n = n$ and $\pi_1 \neq n-1$ is a LTR maximum of π. Writing $\pi = \pi_1 M n$, we have that both $n\pi_1 M$ and $\pi_1 n M$ are preimages of π, so π has more than one preimage.

The formula for $c_n^{(1)}$ follows immediately, since there are no restrictions on the terms $\pi_2, \dots, \pi_{n-1}$. □

The following lemma helps dealing with the case in which $n-1$ and n are adjacent in π.

Lemma 5.6. *Let π be a permutation of length $n > 2$ such that $\pi_{n-1} = n-1$ and $\pi_n = n$. Then π has at least four preimages.*

Proof. Since $n > 2$, then π_1, $n-1$ and n are all distinct LTR maxima of π. Writing π as $\pi_1 M(n-1)n$, we have that $(n-1)\pi_1 M n$, $n\pi_1 M(n-1)$, $(n-1)n\pi_1 M$ and $\pi_1 n M(n-1)$ are all distinct preimages of π. □

Proposition 5.7. *Let $\pi = \pi_1 \cdots \pi_n \in S_n$. Then π has exactly two preimages under* `Cons` *if and only if $\pi_n = n$, $\pi_1 \neq n-1 \neq \pi_{n-1}$ and $LTR(\pi) = \{\pi_1, n-1, n\}$, for every $n \geq 4$.*

Therefore $c_n^{(2)} = (n-2)! \sum_{j=1}^{n-3} \frac{1}{j}$ for every $n \geq 4$, and $c_0^{(2)} = 0$, $c_1^{(2)} = 0$, $c_2^{(2)} = 1$, $c_3^{(2)} = 0$.

Proof. If π is of the form described in the statement, then we can use Proposition 5.4 to count its preimages, of which there are precisely two.

On the other hand, if $\pi_n \neq n$ or $\pi_1 = n-1$, then we can use the same argument as in the proof of Proposition 5.5 to see that π has at most one preimage. If $\pi_{n-1} = n-1$, then, by Lemma 5.6, we have that π has more than two preimages. The only remaining case is that of π having more than three LTR maxima. Suppose that α is a LTR maximum, with $\pi_1 < \alpha < n-1$. Writing $\pi = \pi_1 M_1 \alpha M_2 (n-1) M_3 n$, the following three permutations are preimages of π: $n\pi_1 M_1 \alpha M_2 (n-1) M_3$, $\pi_1 n M_1 \alpha M_2 (n-1) M_3$ and $\alpha \pi_1 M_1 n M_2 (n-1) M_3$.

To find a closed formula for $c_n^{(2)}$ we recall Foata's fundamental bijection [18], which maps a permutation σ written in one-line notation to the permutation in cycle notation obtained by inserting a left parenthesis in σ preceding every LTR maximum, then a right parenthesis where appropriate. Applying such a map to the set of permutations of length $n \geq 4$ having exactly two preimages, returns the set of permutations of length n in which n is a fixed point and consisting of two further cycles, the one containing $n-1$ having size at least two. If the cycle not containing $n-1$ has size j, with $1 \leq j \leq n-3$, then we have $\binom{n-2}{j}$ different ways to choose the elements in that cycle, and $(j-1)!$ different ways to arrange the elements into the cycle. The remaining $n-1-j$ elements (including $n-1$) can be arranged in $(n-2-j)!$ different ways. Summing up, we obtain that the number of permutations of length n with exactly two preimages is

$$\sum_{j=1}^{n-3} \binom{n-2}{j} (j-1)!(n-2-j)! = \sum_{j=1}^{n-3} \frac{(n-2)!}{j!(n-2-j)!} (j-1)!(n-2-j)! = (n-2)! \sum_{j=1}^{n-3} \frac{1}{j},$$

which concludes the proof. □

The quantities $H_n = \sum_{j=1}^{n} \frac{1}{j}$ are commonly known as *harmonic numbers*, so the above formula for $c_n^{(2)}$ can be written as $c_n^{(2)} = (n-2)! H_{n-3}$.

Proposition 5.8. *Let $\pi = \pi_1 \cdots \pi_n \in S_n$ with $n \geq 4$. Then π has exactly three preimages under* `Cons` *if and only if $\pi_n = n$, $LTR(\pi) = \{k, k+1, n-1, n\}$, with $1 \leq k \leq n-3$, and both k, $k+1$ and $n-1$, n are not adjacent.*

Proof. If π is as in the statement above, then we can use Proposition 5.3 to count its preimages, of which there are precisely three, one for each of the subsets $\{k, n\}$, $\{k+1, n\}$, $\{n\}$.

On the other hand, we can argue as in the previous propositions to observe that, if a permutation has less than four LTR maxima or $n-1$ and n are adjacent, then it cannot have three preimages. If k and $k+1$ are adjacent, then π has at least four preimages, since each of the four subsets $\{k, n\}$, $\{k+1, n\}$, $\{n\}$, $\{k, k+1, n\}$ corresponds to at least one preimage. The same holds when the four LTR maxima are $k, h, n-1, n$, with $h > k+1$.

Finally, if π has more than four LTR maxima, suppose that $\{\alpha, \beta, \gamma, n-1, n\} \subseteq LTR(\pi)$. Then there is a least a preimage for each of the subsets $\{\alpha, n\}$, $\{\beta, n\}$, $\{\gamma, n\}$, $\{n\}$, so π has more than three preimages. $\square$

Looking at the previous results, we may wonder whether every number of preimages is allowed. More formally, does there exist any permutation π such that $|\texttt{Cons}^{-1}(\pi)| = k$, for every k?

For `Stacksort`[19], `Queuesort`[13] and `Bubblesort`[17] the answer is negative, so we could expect that also `Cons` may have some forbidden cardinality for preimages. Surprisingly, this is not the case.

Proposition 5.9. *For every $k \geq 0$, there exist (at least) a permutation π such that $|\texttt{Cons}^{-1}(\pi)| = k$.*

Proof. For $k \geq 5$, a permutation of length $k-1$ with k preimages is $(k-3)\rho(k-2)(k-1)$, with $\rho \in S_{k-4}$. Moreover, 3 152 647 has four preimages. We can thus conclude thanks to Propositions 5.5, 5.7 and 5.8. $\square$

6. Conclusion and further work

Motivated by the sorting algorithm `Popstacksort`, we have introduced similar sorting algorithms for permutations which make use of a popqueue. We have shown that there are at least two natural optimal sorting algorithms, that are able to sort all sortable permutations (which turn out to be characterized by the avoidance of two patterns). We have also investigated what happens when each of the two algorithms is performed twice, and for one of them (namely `Cons`) we have provided information on the preimages of a given permutation.

In Section 4, we have already mentiond an intriguing open problem, concerning the exact enumeration of permutations sortable by two passes of `Cons` (or `Min`). Moreover, the study of combinatorial properties of the sorting trees associated with each of our two optimal sorting algorithms would be desirable, along the same lines it has been done for `Bubblesort` in [17]. Recall that the sorting tree of order n associated with a given sorting algorithm `Algsort` is the infinite rooted tree whose nodes are all permutations of length n, whose root is the identity permutation $12\cdots n$ and such that, for every node $\pi \in S_n$, the parent of π is the permutation $\texttt{Algsort}(\pi)$ obtained by performing `Algsort` on π. Another potentially interesting issue could be to compose `Cons` or `Min` with other sorting algorithms: the problem of characterizing and enumerating the resulting sets of sortable permutations may reveal some nice combinatorial structure. Another suggestion for further work is to determine the average number of applications of `Min` or `Cons` needed to sort a permutation; this was considered for Stacksort in [20], for Popstacksort in [21] and for Bubblesort in [17]. Finally, the study of preimages of single permutations through sorting operators is a field of research still to be fully explored. Apart from the results concerning the algorithms cited throughout the paper, there are many other algorithms that can be investigated along the same line. The MSc thesis of Magnusson [12] is a source of interesting examples in this respect.

Acknowledgements. We would like to thank the anonymous referees, whose comments and suggestions have contributed to improve the overall quality of the paper.

Funding. L.C. and L.F. are both members of the INdAM research group GNCS; they are partially supported by the 2022 INdAM-GNCS project "Stringhe e matrici: combinatoria, enumerazione e algoritmi".

References

[1] V.R. Pratt, Computing permutations with double-ended queues. Parallel stacks and parallel queues. *Proc. Fifth Annual ACM Symposium on Theory of Computing* (1973) 268–277

[2] R.E. Tarjan, Sorting using networks of queues and stacks, *J. ACM* **19** (1972) 341–346.

[3] J. West, Sorting twice through a stack. *Theoret. Comput. Sci.* **117** (1993) 303–313.

[4] G. Cerbai, A. Claesson and L. Ferrari, Stack sorting with restricted stacks. *J. Combin. Theory Ser. A* **173** (2020) 105230.

[5] G. Cerbai, A. Claesson, L. Ferrari and E. Steingrimsson, Sorting with pattern-avoiding stacks: the 132-machine. *Electron. J. Combin.* **27** (2020) 3.32.

[6] C. Defant, M. Engen and J.A. Miller, Stack-sorting, set partitions, and Lassalle's sequence. *J. Combin. Theory Ser. A* **175** (2020) 105275.

[7] R. Smith, Two stacks in series: a decreasing stack followed by an increasing stack. *Ann. Comb.* **18** (2014) 359–363.

[8] J. West, *Permutations with Forbidden Subsequences and Stack Sortable Permutations*. PhD thesis, Massachusetts Institute of Technology (1990).
[9] D. Avis and M. Newborn, On pop-stacks in series. *Utilitas Math.* **19** (1981) 129–140.
[10] L. Pudwell and R. Smith, Two-stack-sorting with popstacks. *Australas. J. Combin.* **74** (2019) 179–195.
[11] A. Claesson and B.A. Guðmundsson, Enumerating permutations sortable by k passes through a pop-stack. *Adv. Appl. Math.* **108** (2019) 79–96.
[12] H. Magnusson, *Sorting operators and their preimages*. MSc thesis, Reykjavik University (2013).
[13] L. Cioni and L. Ferrari, Preimages under the `Queuesort` algorithm. *Discrete Math.* **344** (2021) 112561.
[14] L. Cioni and L. Ferrari, Characterization and enumeration of preimages under the `Queuesort` algorithm, in Extended Abstracts EuroComb 2021. Trends in Mathematics, Vol. 14, edited by J. Nešetřil, G. Perarnau, J. Rué, J. and O. Serra. Birkhäuser, Cham.
[15] J. West, Generating trees and forbidden sequences. *Discrete Math.* **157** (1996) 363–374.
[16] N.J.A. Sloane, *The On-Line Encyclopedia of Integer Sequences*. Available at oeis.org.
[17] M. Bouvel, L. Cioni and L. Ferrari, Preimages under the Bubblesort operator. *Electron. J. Combin.* **29** (2022) 4.32.
[18] D. Foata and M.-P. Schützenberger, Théorie géométrique des polynômes Eulériens. *Lecture Notes Math.* **138** (1970).
[19] C. Defant, Fertility numbers. *J. Comb.* **11** (2020) 527–548.
[20] C. Defant, Fertility monotonicity and average complexity of the stack-sorting map. *Eur. J. Combin.* **93** (2021) 103276.
[21] L. Lichev, Lower bound on the running time of Pop-Stack sorting on a random permutation. Available at https://arxiv.org/abs/2212.09316.