Sorting with a popqueue

Lapo Cioni

lapo.cioni@unifi.it

Università degli Studi di Firenze

(This talk is based on joint work with Luca Ferrari.)

1 Preliminary notions and results

`Stacksort` is a classical and well-studied algorithm that attempts to sort an input permutation by (suitably) using a stack. It has been introduced and first investigated by Knuth [1], and it is one of the main responsible for the great success of the notion of *pattern* for permutations. Among the many research topics connected with `Stacksort`, one concerns the characterization and enumeration of the permutations sorted by two (or more) iterations of the algorithm. For example, West [3] described the permutations that are sortable by two iterations of `Stacksort` in terms of avoidance of a standard pattern and a barred pattern, and he also formulated a conjecture regarding their enumeration, which was later proved by Zeilberger [4].

We will address the same kind of problems for a different sorting device, namely a popqueue. A popqueue is a sorting device in which we can insert and extract elements, following some restrictions. Namely, these are the allowed operations:

- e: *enqueue*, insert the current element into the popqueue, in the rightmost position;
- p: *pop*, remove all the elements currently in the popqueue, from left to right, sending them into the output;
- b: *bypass*, send the current element into the output.

These operations resemble those of a queue. Indeed, the sole difference is the fact that *pop* removes all the elements instead of removing only the first one, and this is the reason for the name "popqueue".

Starting from an arbitrary input, we can output several different permutations, just by executing the operations in different order. Our interest is to sort the permutation, so we say that a permutation is *sortable* if there exists a sequence of operations that output the identity permutation. In particular, we want to give an optimal sorting algorithm, i.e. we want to describe an algorithm that is able to sort every sortable permutation. We also want our algorithm to give an output when the input permutation is not sortable, so that we can study which permutations are sortable by two iterations of the same algorithm.

We start by giving a necessary condition for a permutation to be sortable.

Proposition 1. *If a permutation $\pi \in S_n$ contains an occurrence of the patterns* 321 *or* 2413*, then it is not sortable using a popqueue.*

2 The algorithms *Min* and *Cons*

We provide two different sorting algorithms which, although similar, will give us remarkably different results when applied twice. The first algorithm is Min.

Algorithm. *Min*
input: a permutation $\pi = \pi_1 \cdots \pi_n$
output: a permutation $Min(\pi)$

```
for $i = 1, \ldots, n$ do:

  • if $Front(Q)$ is the minimal element not yet in the output (i.e., if $Front(Q)$ is
    smaller than all the unprocessed elements $\pi_i, \ldots, \pi_n$), then pop and enqueue;

  • else compare $\pi_i$, $Back(Q)$ and $Front(Q)$;

      – if $Back(Q) < \pi_i$, enqueue;
      – otherwise, if $Front(Q) > \pi_i$, then bypass;
      – else, pop end enqueue.

Finally, pop.
```

We will call $Min(\pi)$ the output of the algorithm Min on input π. The algorithm name, Min, comes from the first instruction, which empties the popqueue when the first element of the queue is the first element to be output. Min is an optimal sorting algorithm, as recorded in the next proposition.

Proposition 2. *Let* $\pi \in S_n$*. Then* $Min(\pi) \neq id_n$ *if and only if* $321 \leq \pi$ *or* $2413 \leq \pi$.

The second algorithm is $Cons$.

Algorithm. *Cons*
input: a permutation $\pi = \pi_1 \cdots \pi_n$
output: a permutation $Cons(\pi)$

```
for $i = 1, \ldots, n$ do:

  • if $\pi_i = Back(Q) + 1$, then enqueue;

  • else, compare $\pi_i$ and $Front(Q)$;

      – if $Front(Q) > \pi_i$, then bypass;
      – else, pop and enqueue.
```

Finally, pop.

We will call $Cons(\pi)$ the output of the algorithm $Cons$ on input π. The algorithm name, $Cons$, comes from the first instruction, that only allows consecutive elements to be in the popqueue, and thus forces the content of the popqueue to be consecutive at all times. As for the Min algorithm, it is an optimal sorting algorithm.

Proposition 3. *Let $\pi \in S_n$. Then $Cons(\pi) \neq id_n$ if and only if $321 \leq \pi$ or $2413 \leq \pi$.*

We thus have two different optimal sorting algorithms, which follow different heuristics to sort π. It is easy to see that, althought they sort the same permutations, their output is different in general. For example, $Min(2413) = 1243$ but $Cons(2413) = 2134$. Still, it is important to note that both heuristics are reasonable in the context of permutation sorting. Indeed, neither Min nor $Cons$ create new inversions in their output. Also, we can say that if the first element of the popqueue is the minimal element that is not yet in the output, then we may as well empty the popqueue immediately, as Min does, because we would surely empty it before outputting any element from the input. On the other hand, if we were to allow non consecutive elements in the popqueue, then we would be sure that the output would not be sorted, so it is reasonable to maintain consecutivity fot elements inside the popqueue, as $Cons$ does. Both ideas give us optimal algorithms, whose outputs differs only for permutations that are not sortable, moreover the order in which the operations are executed may be different even for sortable permutations. One could be tempted to see what happens when using both heuristics, with an algorithm that only allows consecutive elements in the popqueue, but also empties it whenever the first element is the smallest not-outputted element. That would actually be an algorithm equivalent to $Cons$, because the output would be the same, although the operation would be performed in different orders by prioritizing the outputting of elements over the insertion into the popqueue.

We end 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 LTR maximum if and only if it is greater than all π_j's for $j < i$. We have the following lemma.

Lemma 4. *Let $\pi = \pi_1 \cdots \pi_n \in S_n$ and apply Cons to it. Then an element π_i enters the popqueue if and only if it is a LTR maximum. Moreover, the relative order of the non-LTR maxima of π is preserved in $Cons(\pi)$. That is, if a appears before b in π, and neither are LTR maxima, then a appears before b even in $Cons(\pi)$.*

This lemma does not fully hold for Min, because although every LTR maxima does enter the popqueue during its execution, some non LTR maxima may also enter it (for example, the element 4 in 25143 enters the popqueue). It is interesting to note that the algorithm that sorts using a proper queue (that is, a queue instead of a popqueue) also have the property described in the lemma.

3 Two passes from a popqueue

In this section we will investigate what permutations are sortable when we apply twice each of the previous algorithms. This has been done for Stacksort by West [3], and the resulting set of sortable permutation is not a class, even if it can be expressed by the avoidance of a pattern and a barred pattern.

We start by defining the sets $Sort_M$ and $Sort_C$ of permutations sorted by two applications of Min and $Sort$, respectively, so that $Sort_M = \{\pi \in S \mid Min(Min(\pi)) = id_n, n \in \mathbb{N}\}$ and $Sort_C = \{\pi \in S \mid Cons(Cons(\pi)) = id_n, n \in \mathbb{N}\}$. The following remark shows the relationships between the two sets.

Remark 5. *Consider the permutations* 2431 *and* 35214*. Then* $2431 \in Sort_C \setminus Sort_M$ *and* $35214 \in Sort_M \setminus Sort_C$*. This shows that each of the two algorithms is able to sort some permutations that the other algorithm cannot sort.*

Clearly, there are permutations that both algorithms can (cannot) sort. For example, it is easy to see that none of them can sort permutations containing the pattern 4321; on the other hand both algorithms can sort the permutations 321 and 2413. As we have already remarked, both algorithms follow natural and easy rules, as Min outputs the smaller non outputted element whenever possible, while $Cons$ keeps the popqueue consecutive at all times. Still, the fact that there are two different optimal single-pass sorting algorithm that follow natural rules is by itself interesting. However the two sets $Sort_C$ and $Sort_M$ are different on a deeper level. In fact, we will see that $Sort_C$ is a permutation class, while $Sort_M$ is not.

Proposition 6. *The set* $Sort_M$ *is not a permutation class.*

Proof. Consider the permutation 241653. Then $Min(Min(241653)) = Min(124536) = 123456$, but $Min(Min(2431)) = Min(2413) = 1243$, and $2413 \leq 241653$. □

Conversely, the following proposition shows that $Sort_C$ is a class.

Proposition 7. *Let* $\pi \in S_n$*. Then* $Cons(Cons(\pi)) \neq id_n$ *if and only if* π *contains at least one of the following nine patterns:*

- 4321*;*
- 35241*;*
- 35214*;*
- 52413*;*
- 25413*;*
- 246153*;*
- 246135*;*
- 426153*;*
- 426135.

We do not know much about the sequences $|Sort_{M,n}|$ of the number of permutations of length n in $Sort_M$, and $|Sort_{C,n}|$ of the number of permutations of length n in $Sort_C$. Their first terms are 1, 2, 6, 22, 89, 379, 1660, 7380, 33113, 149059 and 1, 2, 6, 23, 99, 445, 2029, 9292, 42608, 195445, respectively, and do not appear in [2]. Still, it seems that $|Sort_{M,n}|$ is smaller than $|Sort_{C,n}|$ for every $n > 3$.

Conjecture 8. *For every $n > 3$, $|Sort_{M,n}| < |Sort_{C,n}|$.*

[1] D. Knuth, The Art of Computer Programming, Volume 1, *Boston: Addison-Wesley*, 1968.

[2] N. J. A. Sloane, editor, The On-Line Encyclopedia of Integer Sequences, published electronically at https://oeis.org.

[3] J. West, Permutations with forbidden subsequences and Stack sortable permutations, *PhD thesis, Massachusetts Institute of Technology*, 1990.

[4] D. Zeilberger, A proof of Julian West conjecture that the number of two-stack-sortable pemutations of length n is $2(3n)!/((n+1)!(2n+1)!)$. *Discrete math.* 102 (1992), no. 1, 85-93.