Enumerating permutations sortable by k passes through a pop-stack

Anders Claesson[1,*], Bjarki Ágúst Guðmundsson[1,*]

Science Institute, University of Iceland, Dunhaga 5, IS-107 Reykjavik, Iceland

Abstract

In an exercise in the first volume of his famous series of books, Knuth considered sorting permutations by passing them through a stack. Many variations of this exercise have since been considered, including allowing multiple passes through the stack and using different data structures. We are concerned with a variation using pop-stacks that was introduced by Avis and Newborn in 1981. Let $P_k(x)$ be the generating function for the permutations sortable by k passes through a pop-stack. The generating function $P_2(x)$ was recently given by Pudwell and Smith (the case $k = 1$ being trivial). We show that $P_k(x)$ is rational for any k. Moreover, we give an algorithm to derive $P_k(x)$, and using it we determine the generating functions $P_k(x)$ for $k \leq 6$.

Keywords: enumeration, generating function, pop-stack, permutation, rational, automaton

1. Introduction

Knuth [10, Exercise 2.2.1.5] noted that permutations sortable by a stack are precisely those that do not contain a subsequence in the same relative order as the permutation 231. This exercise inspired a wide range of research and can be seen as the starting point of the research field we now call permutation patterns. Our interest lies in Knuth's original exercise and its variations.

In 1972 Tarjan [14] considered sorting with networks of stacks and queues, a problem that, in general, has proven itself to be beyond the reach of current methods in permutation patterns and enumeration. There has been some recent significant progress though: In 2015 Albert and Bousquet-Mélou [1] enumerated permutations sortable by two stacks in parallel. Sorting with two stacks in series is, however, an open problem. A related problem is that of sorting permutations by k passes through a stack, where the elements on the stack are required to be increasing when read from top to bottom. West [17] characterized the permutations sortable by two passes through a stack in terms of pattern avoidance and conjectured their enumeration, a conjecture that was subsequently proved by Zeilberger [18]. Permutations sortable by three passes have been characterized by Úlfarsson [15], but their enumeration is unknown.

*Corresponding author, akc@hi.is

[1]This work was supported by The Doctoral Grants of the University of Iceland Research Fund and The Icelandic Infrastructure Fund

In other variations of Knuth's exercise different data structures are used for sorting. One notable example is that of pop-stacks: a stack where each pop operation completely empties the stack. Avis and Newborn [5] enumerated the permutations sortable by pop-stacks in series, with the modification that each pop leads to the popped elements being immediately pushed to the following pop-stack. Atkinson and Stitt [4] considered two pop-stacks in genuine series. Permutations sortable by pop-stacks in parallel have been studied by Atkinson and Sack [3], who characterized those permutations by a finite set of forbidden patterns. They also conjectured that the generating function for their enumeration is rational, which was subsequently proved by Smith and Vatter [13], who gave an insertion encoding [2] for the sortable permutations.

Pudwell and Smith [11] recently characterized the permutations sortable by two passes through a pop-stack in terms of pattern avoidance and gave their enumeration. They also gave a bijection between certain families of polyominoes and the permutations sortable by one or two passes through a pop-stack, but noted that the bijection does not generalize to three passes. In this paper we consider, more generally, the permutations sortable by k passes through a pop-stack. In particular, we give an algorithm to derive a rational generating function for the permutations sortable by k passes through a pop-stack, for any fixed k.

2. Sorting plans and traces

A single pass of the pop-stack sorting operator formally works as follows. Processing a permutation $\pi = a_1 a_2 \dots a_n$ of $[n] = \{1, \dots, n\}$ from left to right, if the stack is empty or its top element is smaller than the current element a_i then perform a single pop operation (`a`), emptying the stack and appending those elements to the output permutation; else do nothing (`d`). Next, push a_i onto the stack and proceed with element a_{i+1}, or if $i = n$ perform one final pop operation (`a`), again emptying the stack onto the output permutation, and terminate. Define $P(\pi)$ as the final output permutation and $w(\pi)$ as the word over the alphabet $\{\mathtt{a}, \mathtt{d}\}$ defined by the operations performed when processing π. For instance, with $\pi = 752491863$ we have $P(\pi) = 257419368$ and $w(\pi) =$ `addaadadda`. Note that $w(\pi)$ will always begin and end with the letter `a`. We will call any word of length $n+1$ with letters in $\{\mathtt{a}, \mathtt{d}\}$ that begin and end with the letter `a` an *operation sequence*.

Let us now introduce what we call sorting traces. Consider applying the pop-stack operator P to a permutation π of $[n]$. Start by interleaving $w(\pi)$ with π; for instance, with $\pi = 752491863$ (as before) we have `a7d5d2a4a9d1a8d6d3a`. Replacing `a` with a bar and `d` with a space, and placing $P(\pi)$ below this string, we have

7	5	2	4	9	1	8	6	3
2	5	7	4	1	9	3	6	8

We call the numbers between pairs of successive `a`'s *blocks*. Above, the blocks of π are 752, 4, 91, and 863. Note that $P(\pi)$ can be obtained from π by reversing each block. An index $i \in [n-1]$ of a permutation $\pi = a_1 a_2 \dots a_n$ is an *ascent* if $a_i < a_{i+1}$. Similarly, i is a *descent* if $a_i > a_{i+1}$. With this terminology $w(\pi) = c_1 c_2 \dots c_{n+1}$ is simply the ascent/descent word of $-\infty\, \pi\, \infty$;

i.e. $c_1 = c_{n+1} = \mathtt{a}$ and, for $2 \le i \le n$,

$$c_i = \begin{cases} \mathtt{a} & \text{if } i-1 \text{ is an ascent,} \\ \mathtt{d} & \text{if } i-1 \text{ is a descent.} \end{cases}$$

A figure as the one above can be extended to depict multiple passes through a pop-stack. Applying the pop-stack operator to the example permutation, π, until it is sorted gives:

7	5	2	4	9	1	8	6	3
2	5	7	4	1	9	3	6	8
2	5	1	4	7	3	9	6	8
2	1	5	4	3	7	6	9	8
1	2	3	4	5	6	7	8	9

We will call such figures sorting traces, or traces for short. The structure that remains when removing the numbers from a trace we call its sorting plan. Each row of a sorting plan corresponds to an operation sequence, and for convenience we shall number the rows 1 through k, from top to bottom. The example sorting plan can be viewed as the following array of operation sequences:

```
addaadadda
aaaddadaaa
aadaadadaa
adaddadada
```

By interpreting each column as a binary number with $a = 0$ and $d = 1$ the sorting plan can more compactly be represented, or *encoded*, with the sequence of numbers

$$0,\ 9,\ 10,\ 5,\ 5,\ 10,\ 5,\ 10,\ 9,\ 0.$$

Formally, we define a trace and its sorting plan as follows.

Definition 2.1. Let $A = (\alpha_1, \alpha_2, \ldots, \alpha_k, \alpha_{k+1})$ be a $(k+1)$-tuple of permutations of $[n]$ in which α_{k+1} is the identity permutation. Let $M = (\mu_1, \mu_2, \ldots, \mu_k)$ be a k-tuple of operation sequences, each of length $n+1$. We call $T = (A, M)$ a *trace* of *length* n and *order* k, and M its *sorting plan*, if the following conditions are satisfied for $i = 1, \ldots, k$:

1. The word μ_i records the sequence of operations performed by the pop-stack operator when applied to α_i—in symbols, $w(\alpha_i) = \mu_i$. As to the picture of the trace, two adjacent numbers form an ascent if and only if they are separated by a bar.

2. The sequence of permutations in A records repeated application of the pop-stack operator to the permutation α_1—in symbols, $P(\alpha_i) = \alpha_{i+1}$. Or, assuming the first condition is satisfied, $\mathrm{rev}(\alpha_i, \mu_i) = \alpha_{i+1}$, where the operator rev is defined below.

Definition 2.2. Let π be a permutation of $[n]$ and $\mu = c_1 c_2 \dots c_{n+1}$ be an operation sequence. Let $i_1 < i_2 < \dots < i_k$ be the sequence of indices i for which $c_i = \mathtt{a}$. Write $\pi = \gamma_1 \gamma_2 \dots \gamma_{k-1}$, where $|\gamma_j| = i_{j+1} - i_j$ for $j = 1, \dots, k-1$. In other words, the length of γ_j is the same as the length of the jth block of π with respect to μ. Define $\mathrm{rev}(\pi, \mu) = \gamma_1^r \gamma_2^r \dots \gamma_{k-1}^r$, where $(\,\cdot\,)^r$ is the reversal operator. We call this the *blockwise reversal of π according to μ.*

Because of the strict constraints on the operation sequences and the permutations, much of the information stored in a trace is redundant. As we have seen, the first permutation α_1 alone determines the complete trace. Similarly, if we have the sorting plan M, then we can recover the permutations α_1, …, α_{k+1}. This is possible because the last permutation α_{k+1} is the identity, the permutation α_k is the blockwise reversal of α_{k+1} according to μ_k, the permutation α_{k-1} is the blockwise reversal of α_k according to μ_{k-1}, etc. In symbols, $\mathrm{rev}(\alpha_i, \mu_i) = \alpha_{i+1}$ if and only if $\alpha_i = \mathrm{rev}(\alpha_{i+1}, \mu_i)$. In this way a sorting plan uniquely determines a trace.

Our goal is to count how many permutations of $[n]$ are sortable by k passes through a pop-stack. Note that a permutation sortable by k passes is also sortable by $k+1$ passes. For brevity we will sometimes refer to a permutation sortable by k passes through a pop-stack as *k-pop-stack-sortable*. Starting with a k-pop-stack-sortable permutation of $[n]$ and performing k passes of the pop-stack sorting operator results in a trace of length n and order k. Conversely, the first row of that trace is the k-pop-stack-sortable permutation we started with. Thus, k-pop-stack-sortable permutations of $[n]$ are in one-to-one correspondence with traces of length n and order k. Those are, in turn, in one-to-one correspondence with their sorting plans, and hence it suffices to count sorting plans of length n and order k.

Let us call any k-tuple of operation sequences of length $n+1$ an *operation array* of length $n+1$ and order k. Note that sorting plans are operation arrays, but not all operation arrays are sorting plans. Let an operation array $(\mu_1, \mu_2, \dots, \mu_k)$ be given. Let the permutations α_1, α_2, …, α_{k+1} be defined by requiring that the permutation α_{k+1} is the identity, the permutation α_k is the blockwise reversal of α_{k+1} according to μ_k, the permutation α_{k-1} is the blockwise reversal of α_k according to μ_{k-1}, etc. In symbols, $\mathrm{rev}(\alpha_i, \mu_i) = \alpha_{i+1}$. Then the tuple $(\alpha_1, \dots, \alpha_{k+1})$ is called a *semitrace*. In other words, a semitrace is defined by requiring that Property 2 of Definition 2.1 holds, but not necessarily Property 1 of the same definition.

We shall characterize those operation arrays that are sorting plans, then count the sorting plans of order k, and by extension the k-pop-stack-sortable permutations. We start by considering cases where k is small.

2.1. 1-pop-stack-sortable permutations

Let us count permutations of $[n]$ sortable by one pass through a pop-stack, or, equivalently, sorting plans of length n and order 1. We want to determine which operation sequences of length $n+1$ when interpreted as operation arrays are sorting plans. Consider the operation sequence `addaadada` as an example. The semitrace corresponding to it is

3	2	1	4	6	5	8	7
1	2	3	4	5	6	7	8

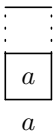
Is this a trace? By definition of a semitrace it satisfies Property 2 of Definition 2.1, but does it also satisfy Property 1? Yes, because the bottom-most permutation is the identity we find that any two adjacent numbers separated by a bar form an ascent, and two adjacent numbers within a block (i.e. adjacent numbers that are not separated by a bar) form a descent. Thus, each operation sequence represents a sorting plan of order 1, and hence a 1-pop-stack-sortable permutation. An operation sequence starts and ends with the letter `a`. The remaining letters can be either `a` or `d`. Thus, for $n > 0$, there are 2^{n-1} operation sequences of length $n+1$, and hence 2^{n-1} 1-pop-stack-sortable permutations of $[n]$. Based on how they are derived from the operation sequences it is easy to see that these are precisely the layered permutations (direct sums of decreasing permutations).

While this simple example outlines our approach to count k-pop-stack-sortable permutations, it is a little too simple. For larger k, most operation arrays will not be sorting plans. Before attacking the problem in full generality, let us consider the case $k = 2$ and see how to restrict the operation arrays to those that represent sorting plans.

2.2. 2-pop-stack-sortable permutations

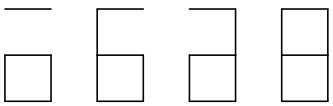
Pudwell and Smith [11] enumerated 2-pop-stack-sortable permutations. Let us reproduce their results by classifying sorting plans of order 2. Given an arbitrary operation array of order 2, i.e. a pair of operation sequences of length $n + 1$, we will start with the identity permutation at the bottom, fill in the remaining two permutations, and then determine if the resulting semitrace is a trace. As we saw in the previous section, the second operation sequence will always satisfy Property 1. To determine when the first operation sequence satisfies Property 1 let us do case analysis based on the size of an arbitrary block in the second row of our trace.

If the block is of size 1, then we have a single integer $a \in [n]$ and the neighborhood around the block looks as follows:

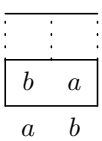


Here, a dotted line can either be solid or not, representing the presence or absence of a bar, respectively.

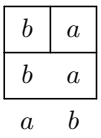
Since at least two numbers are needed to break Property 1 all four configurations of the two dotted lines are possible:



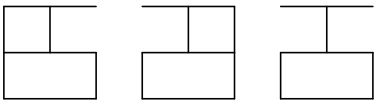
If the block is of size 2, then we have two integers $a, b \in [n]$, with $b = a + 1$, and the neighborhood around the block looks as follows:



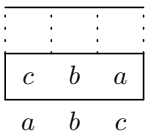
Consider the dotted line in the middle and assume for a moment that it is not solid. Then a and b will appear together in the block above, with a appearing before b, and this means that the numbers within this block are not in decreasing order, a contradiction. Hence the dotted line in the middle must be solid. Now, assume that the remaining two dotted lines are solid too. Then the placement of a and b in the first row is determined:



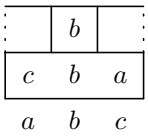
But here we reach a contradiction: in the first row, a and b form a descent but are separated by a bar. Thus, at most one of the two dotted lines on the boundary can be solid. This leaves three possibilities, and we will not be able to reduce their number any further:



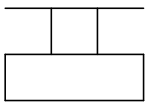
If the block is of size 3, then we have three integers $a, b, c \in [n]$, with $b = a+1$ and $c = b + 1$, and the neighborhood around the block looks as follows:



If either a and b, or b and c were together in a block in the first row, then the numbers in that block would not be in decreasing order, a contradiction. The two centermost dotted lines must thus be solid:



If either of the two remaining dotted lines were solid, then either a and b, or b and c would form an ascent and be separated by a bar, a contradiction. Hence neither of the two remaining dotted lines can be solid, leaving only one possibility:

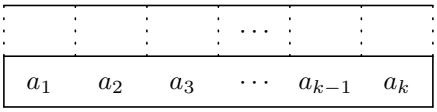


For blocks of size 4 or greater, we present the following lemma.

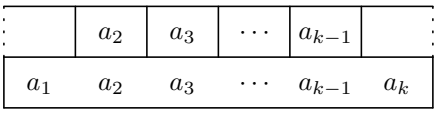
Lemma 2.3. *In a trace of order* 2 *or greater, each operation sequence—except for the first one—contains at most* 2 *consecutive* `d`*'s. Or, equivalently, each row of the sorting plan—except for the first one—has blocks of size at most* 3*.*

Proof. Given a trace of order 2 or greater, consider any block from any row of its sorting plan, excluding the first row. Assume the block has size $k \geq 4$, and

that it contains the numbers $a_1, a_2, \ldots, a_k$, where $a_1 > a_2 > \cdots > a_k$. The neighborhood around the block then looks as follows:

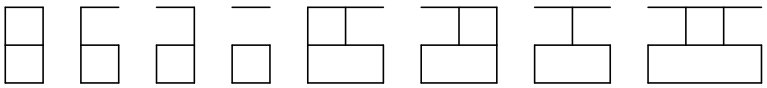


Consider the dotted lines in the upper row, except for the two outermost dotted lines. If any of them were not solid, then at least two of $a_1, a_2, \ldots, a_k$ would be together in a block. These two elements would occur in increasing order, violating Property 1 of Definition 2.1. These dotted lines must thus be solid:



Now, however, a_2 and a_3 are two adjacent numbers in distinct blocks that do not form an ascent, again violating Property 1. Hence our assumption that $k \geq 4$ must have been false. □

Returning to the traces of order 2, this lemma tells us that our case analysis, above, is complete. We have thus restricted the possible local neighborhoods, based on the size of any block in the second row, to the following:



We have shown that these eight possibilities are necessary, in the sense that, if we take a sorting plan of order 2 and slice it up along the boundaries of the blocks in the second row, then the pieces must all be contained in the above set. What is perhaps a bit surprising is that this holds in the other direction as well: gluing together any proper sequence of the above pieces gives a sorting plan, where proper means that

- the first piece starts with a fully solid boundary,
- the last piece ends with a fully solid boundary, and
- for every pair of adjacent pieces, the right boundary of the left piece and the left boundary of the right piece coincide.

This follows from the more general results of the next section. We will not provide a proof for this special case here, but the intuition is that all scenarios that could violate Property 1 happen locally, and the analysis above looks at large enough neighborhoods to get rid of all of these bad scenarios.

With this classification it now becomes straightforward to count the sorting plans of order 2. Let C be the ordinary generating function for "closed" sequences; that is, proper sequences of pieces that both begin and end with a fully solid boundary. Let H be the ordinary generating function for "half open" sequences; that is, proper sequences that begin with a fully solid boundary but

end with a half-solid boundary. Then

$$C = \; | \; + \; C \; \square \; + \; H \Big(\; \square \; + \; \square \; \Big) ;$$

$$H = \; C \Big(\; \square \; + \; \square \; \Big) \; + \; H \Big(\; \square \; + \; \square \; + \; \square \; \Big) .$$

Using the formal variable x to keep track of the length of the partial sorting plan we get

$$\begin{aligned} C &= 1 + xC + (x + x^2)H; \\ H &= (x + x^2)C + (x + x^2 + x^3)H. \end{aligned}$$

The sequences that both begin and end with a fully solid boundary are the sorting plans, so the ordinary generating function for sorting plans of order 2, and hence 2-pop-stack-sortable permutations, is C. Solving for it we recover Pudwell and Smith's [11] result:

$$C = (x^3 + x^2 + x - 1)/(2x^3 + x^2 + 2x - 1).$$

Here we managed to give a neat classification of sorting plans of order 2 by looking at local neighborhoods and getting rid of the invalid scenarios. In the next section we will generalize this approach, looking at what we call forbidden segments.

3. Forbidden segments

A semitrace fails to be a trace precisely when there is a pair of elements witnessing Property 1 of Definition 2.1 fail. With the following definition we single out such pairs.

Definition 3.1. Let T be a semitrace of length n and let a and b be two distinct elements of $[n]$. We call (a, b) a *violating pair* of T if, in any row, a and b occur in adjacent positions such that they violate Property 1. That is, a and b form a descent and are separated by a bar, or a and b form an ascent and are not separated by a bar.

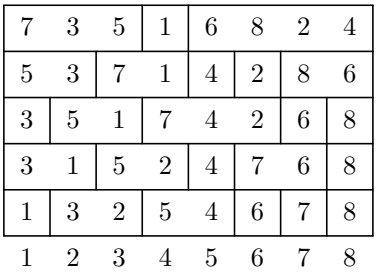
From Definitions 2.1 and 3.1 we immediately get the following characterization of sorting plans.

Lemma 3.2. *An operation array is a sorting plan if and only if its semitrace has no violating pair* (a, b).

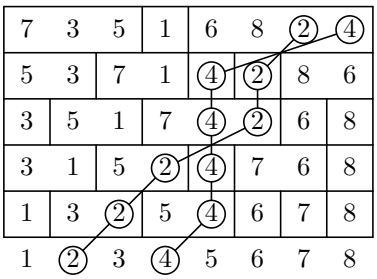
Definition 3.3. Let ψ be the bijection mapping a sorting plan to its encoding. Let M be a sorting plan of length n and let i and j be two indices with $1 \leq i \leq j \leq n + 1$. Let $\psi(M) = c_1 c_2 \dots c_{n+1}$ be the encoding of M. Then we call $S = \psi^{-1}(c_i c_{i+1} \dots c_j)$ a *segment* of M, and $|S| = j - i + 1$ its length.

Definition 3.4. Let $T = (A, M)$ be a semitrace of length n and let a and b be two distinct elements of $[n]$. Let B be the set of blocks that contain either a or b, excluding blocks in the first row. We define *the segment of* T *determined by* a *and* b, denoted $T_{a,b}$, as the smallest segment of M that fully contains all the blocks in B.

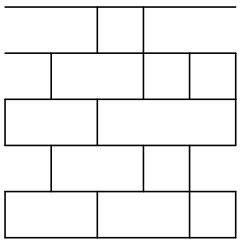
As an example, consider the semitrace



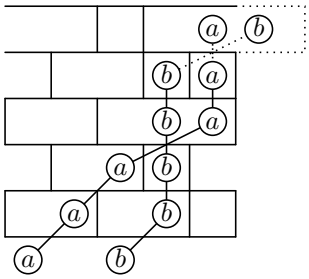
It contains four violating pairs, namely $(1, 5)$, $(2, 4)$, $(3, 5)$, and $(6, 8)$. Let us look at the pair $(2, 4)$ and how these two numbers progress through the trace:



The segment $T_{2,4}$ is the smallest segment that contains all blocks on rows 2, 3, 4, and 5 with at least one circled element:



Any sorting plan of order 5 that contains this segment, no matter where it occurs horizontally, will violate Property 1, just as the above semitrace. This is because we can follow the two elements a and b playing the roles of 2 and 4 from the bottom to the second row.



Formally, we make the following definition.

Definition 3.5. A segment $T_{a,b}$ is *forbidden* if, after inferring the positions of a and b in rows 2 to k, either the two numbers violate Property 1 on row i, where $2 \leq i \leq k$, or

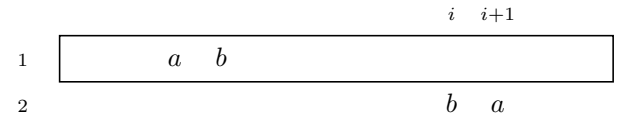


Figure 1: The first case of Definition 3.5, assuming $a < b$

1. a and b form a descent on row 2 and, if a and b are in columns i and $i+1$ on row 2, there is no bar separating columns i and $i+1$ on row 1, or

2. a and b are in decreasing order on row 2 and, if a and b are in columns i and j on row 2, with $i < j$, there is a bar immediately to the left of column i on row 1, a bar immediately to the right of column j on row 1, and exactly one bar between columns i and j on row 1.

In both cases the bars that we reference belong to the segment $T_{a,b}$. In other words, we can determine if $T_{a,b}$ is forbidden or not without knowing in which semitrace it is embedded.

Lemma 3.6. *A segment $T_{a,b}$ is forbidden if and only if (a, b) is a violating pair of T.*

Proof. Consider a forbidden segment $T_{a,b}$. If, after inferring the positions of a and b in rows 2 to k, the two numbers violate Property 1 on row i, where $2 \leq i \leq k$, then (a, b) is a violating pair of T. Otherwise we have two cases, corresponding to the two cases of Definition 3.5, and these are illustrated in Figures 1 and 2, respectively. In the first case a and b form an ascent and are in the same block on row 1. In the second case a and b form a descent and are separated by a bar on row 1. In both cases we have found a violation of Property 1 and hence (a, b) is a violating pair of T. Conversely, consider a violating pair (a, b) of T, as well as the segment $T_{a,b}$. If a and b violate Property 1 on row i, where $2 \leq i \leq k$, then $T_{a,b}$ is a forbidden segment. If, on the other hand, the two numbers violate Property 1 on row 1, then we have two cases:

- If, on row 1, a and b form an ascent and are not separated by a bar, then the two numbers are in the same block on row 1, and will form a descent on row 2. Furthermore, if the two numbers are in columns i and $i+1$ on row 2, there will not be a bar separating columns i and $i+1$ on row 1. Hence $T_{a,b}$ is a forbidden segment.

- If, on row 1, a and b form a descent and are separated by a bar, then the two numbers are in adjacent blocks on row 1. If i is the leftmost column that the left block intersects, and j is the rightmost column that the right block intersects, then the two numbers will be in decreasing order on row 2, with the larger number in column i and the smaller number in column j. Furthermore, there is a bar immediately to the left of column i on row 1 and a bar immediately to the right of column j on row 1, and exactly one bar between columns i and j on row 1. Hence $T_{a,b}$ is a forbidden segment.

In both cases $T_{a,b}$ is a forbidden segment. □

From Lemmas 3.2 and 3.6 we get the following lemma.

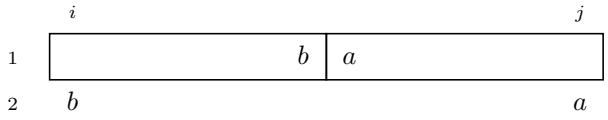


Figure 2: The second case of Definition 3.5, assuming $a < b$

Lemma 3.7. *An operation array is a sorting plan if and only if it does not contain any forbidden segment $T_{a,b}$.*

While this lemma is interesting, it is of limited practical use the way it is stated. The reason being that there are potentially an infinite number of forbidden segments $T_{a,b}$. This motivates the following definition.

Definition 3.8. A segment of order k is *bounded* if each of its blocks in rows 2 to k has size at most 3. Equivalently, a segment is bounded if its operation array has no occurrence of three consecutive `d`'s on rows 2 to k.

Proposition 3.9. *An operation array is a sorting plan if and only if it does not contain any bounded forbidden segment $T_{a,b}$ and each block on rows 2 through k is of size at most 3.*

Proof. Let a sorting plan be given. By Lemma 3.7 it contains no forbidden segment $T_{a,b}$, and, in particular, no bounded forbidden segment $T_{a,b}$. Furthermore, each block on rows 2 through k is of size at most 3 by Lemma 2.3.

Conversely, assume that we are given an operation array that does not contain any bounded forbidden segment $T_{a,b}$ and that each block on rows 2 through k is of size at most 3. If the operation array is not a sorting plan, then the operation array contains a forbidden segment $T_{a,b}$ by Lemma 3.7. This forbidden segment is not bounded, so it has to contain a block of size greater than 3 in one of the rows 2 through k, a contradiction. Hence the operation array must be a sorting plan. □

Lemma 3.10. *Let T be a semitrace of length n and order k and let a and b be two distinct elements of $[n]$. If $T_{a,b}$ is a bounded segment, and there is a block, not on the first row, that includes both a and b, then $|T_{a,b}| \leq 4k - 5$.*

Proof. First note that we can completely disregard the first row, as neither this lemma nor the definition of $T_{a,b}$ includes blocks on that row. Since $T_{a,b}$ is a bounded segment, each of the remaining blocks that either contains a or b has size at most 3. If we consider two adjacent rows, and $x \in \{a, b\}$, this implies that the horizontal distance between x in the upper row and x in the lower row is at most 2. In total, the horizontal distance between x on the second row and x on the k-th row is at most $2(k-2)$ as illustrated in Figure 3. From this we see that the length of the segment $T_{a,b}$ is at most $4(k-2)+m$, where m is the size of the block that contains a and b. Noting that $m \leq 3$ gives us the desired bound. □

Lemma 3.11. *Let T be a semitrace and (a, b) a violating pair of T. Then there is a block, not on the first row, that includes both a and b.*

Proof. Consider a row i where the pair (a, b) violates Property 1. We have the following two cases:

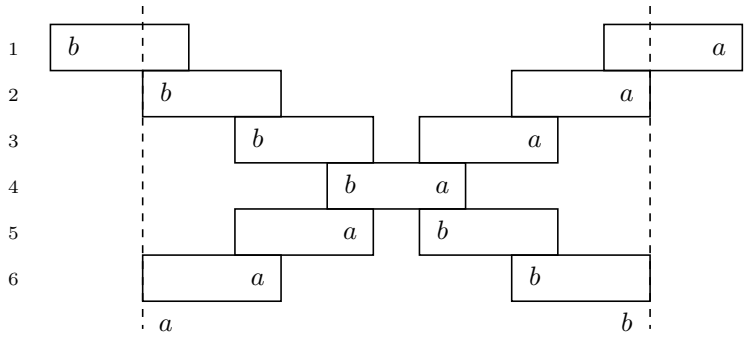


Figure 3: The blocks containing a or b in a trace of order 6 with the dashed lines indicating the boundary of the segment $T_{a,b}$

- If a and b form a descent and are separated by a bar, then the two elements are in distinct blocks on row i, and will still be in descending order on row $i+1$ after performing the blockwise reversals.
- If a and b form an ascent and are not separated by a bar, then the two elements are in the same block on row i, and will be in descending order on row $i+1$ after performing the blockwise reversals.

In either case, a and b will be in descending order on row $i+1$. Since the two elements are in increasing order in the last permutation, i.e. the identity permutation, the two elements must be reversed on at least one of the rows between $i+1$ and k. Since the relative order of elements is only reversed when they appear together in a block, there must be a block on one of the rows between $i+1$ and k that includes both a and b. □

Lemma 3.12. *For a fixed k, there are finitely many bounded forbidden segments $T_{a,b}$ of order k, and they can be listed.*

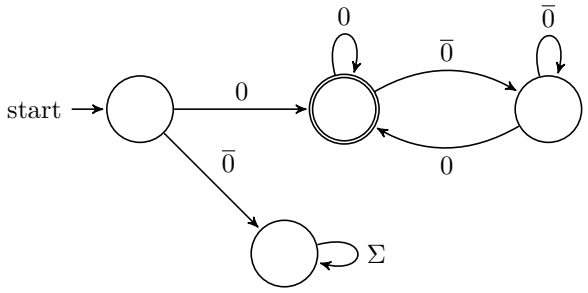
Proof. Assume that $T_{a,b}$ is a bounded forbidden segment. Then (a, b) is a violating pair by Lemma 3.6. Moreover, using Lemma 3.11, we find that there is a block, not on the first row, that includes both a and b. Lemma 3.10 thus applies and $|T_{a,b}| \leq 4k - 5$. Viewing the segment $T_{a,b}$ as an operation array, there consequently are at most as many forbidden segments $T_{a,b}$ as there are words in the finite language $\{\mathtt{a}, \mathtt{d}\}^{(4k-5)k}$, and they can be listed by checking each such operation array against Definition 3.5. □

4. Regular language

Now that we have a characterization of sorting plans in terms of forbidden segments, we will use that characterization to count sorting plans of order k, and hence the k-pop-stack-sortable permutations. To do so, we will employ the theory of formal languages.

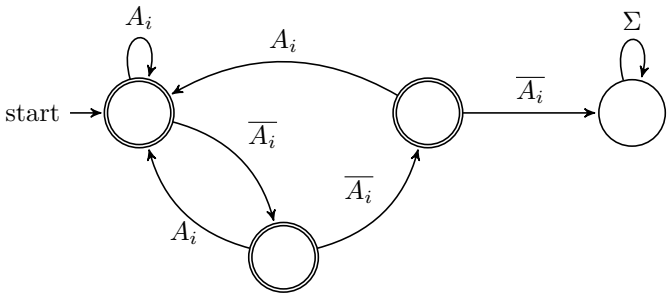
Recall that we can encode an operation array of length n and order k as a sequence of n integers, each in the range $[0, 2^k - 1]$. In this way we can consider operation arrays as strings of a formal language over the alphabet $\Sigma = \{0, 1, \ldots, 2^k - 1\}$. Conversely, strings over this alphabet can be considered as operation arrays, under one condition: that they both begin and end with a

solid boundary. Noting that a solid boundary corresponds to the integer 0 from Σ, and letting $\overline{0} = \Sigma \setminus \{0\}$, the following deterministic finite automaton (DFA), W, recognizes the strings over Σ that begin and end with a solid boundary, i.e. the strings that correspond to an operation array:



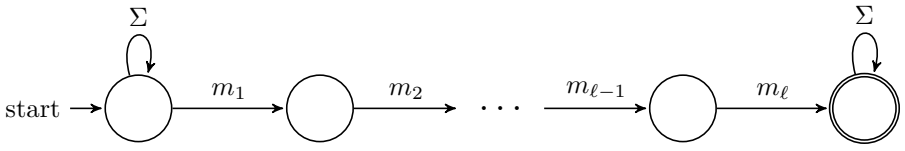
For ease of notation we will use the name of a DFA to also denote the language that it recognizes. Here, W denotes the DFA recognizing strings that begin and end with a solid boundary as well as the language consisting of such strings. We want to find the subset of W corresponding to sorting plans. Recall from Proposition 3.9 that an operation array is a sorting plan if and only if it does not contain any bounded forbidden segments and each block on rows 2 through k is of size at most 3. We shall start with the latter condition.

If A_i is the set of symbols from Σ that represent a column from the operation array that has a bar in the ith row, and $\overline{A_i} = \Sigma \setminus A_i$, then the following DFA, R_i, recognizes the operation arrays that have blocks of size at most 3 in row i:



Therefore, the set of operation arrays that have blocks of size at most 3 in all but the first row is recognized by the DFA $W \cap R_2 \cap \cdots \cap R_k$.

The other condition that sorting plans satisfy is that they do not contain any bounded forbidden segments. Consider a segment M and let us encode it in the same manner as we encode operation arrays, resulting in the sequence $m_1, \ldots, m_\ell$. Note that an operation array A contains the segment M if and only if the encoding of A contains $m_1 \cdots m_\ell$ as a factor. Furthermore, the following nondeterministic finite automaton (NFA), Q_M, recognizes the set of strings over Σ that contain the encoding of M as a factor:



Taking the complement of Q_M we get an automaton $\overline{Q_M}$ that recognizes the set of strings over Σ that do not contain the factor M. In particular, if F is a forbidden segment, then $W \cap \overline{Q_F}$ recognizes the set of operation arrays that do not contain the forbidden segment F.

Let $\mathcal{F}$ be the set of bounded forbidden segments, which is finite by Lemma 3.12. Then the automaton

$$S = W \cap \bigcap_{i=2}^{k} R_i \cap \bigcap_{F \in \mathcal{F}} \overline{Q_F}$$

recognizes the set of operation arrays that have blocks of size at most 3 in rows 2 through k, and do not contain any bounded forbidden segments. Hence, by Proposition 3.9, the automaton S recognizes exactly the set of sorting plans. We have the following proposition:

Proposition 4.1. *The language $S = \{ w \in \Sigma^* \mid w$ is a sorting plan $\}$ is regular.*

We can now present our main theorem.

Theorem 4.2. *For a fixed k, the generating function $P(x) = \sum_{n=0}^{\infty} p_n x^n$, where p_n is the number of k-pop-stack-sortable permutations of length n, is rational.*

Proof. We have a bijection between the k-pop-stack-sortable permutations of length n and the sorting plans of order k and length n, so the two sets are equinumerous. The sorting plans of length n are in bijection with words of length $n+1$ recognized by the automaton S, which is regular by Proposition 4.1. It is well known that regular languages have rational generating functions [12], and they can be derived from the corresponding DFA by setting up a system of linear equations. If $S(x)$ is the rational generating function for S, it is clear that $P(x) = S(x)/x$ and that this generating function is rational. □

Since all of the above results are constructive, it is possible to derive the generating function for any fixed k. Doing so by hand is, however, impractical for almost all values of k, so we implemented the above constructions in the programming languages C++ and Python; the source is on GitHub [6]. This way the generating function can be derived by a computer. Without going into detail, the outline of the mechanized procedure, for a fixed k, was as follows:

1. A smart variant of the procedure given in Lemma 3.12 was used to generate a compact representation of all the bounded forbidden segments. This was achieved by grouping together similar segments, and using "wildcards" to represent a position that could either be a bar or not. This was done to battle the exponential blow up in the number of segments.

2. For each group of bounded forbidden segments, an NFA recognizing the operation arrays containing one or more of the segments from the group was constructed. Each NFA was then turned into a DFA using the classic subset construction, complemented, and then minimized using an algorithm of Valmari [16].

3. Running a MapReduce-like [7] procedure on a cluster [9], these DFAs were intersected, two at a time. To keep the size of the intermediate DFAs down, they were also minimized.

k	Generating function
1	$(x-1)/(2x-1)$
2	$(x^3+x^2+x-1)/(2x^3+x^2+2x-1)$
3	$(2x^{10}+4x^9+2x^8+5x^7+11x^6+8x^5+6x^4+6x^3+2x^2+x-1)/(4x^{10}+8x^9+4x^8+10x^7+22x^6+16x^5+8x^4+6x^3+2x^2+2x-1)$
4	$(64x^{25}+448x^{24}+1184x^{23}+1784x^{22}+2028x^{21}+1948x^{20}+1080x^{19}+104x^{18}-180x^{17}+540x^{16}+1156x^{15}+696x^{14}+252x^{13}+238x^{12}+188x^{11}+502x^{10}+806x^9+544x^8+263x^7+185x^6+99x^5+33x^4+13x^3+3x^2+x-1)/(128x^{25}+896x^{24}+2368x^{23}+3568x^{22}+3928x^{21}+3064x^{20}+176x^{19}-2304x^{18}-2664x^{17}-1580x^{16}-352x^{15}-576x^{14}-1104x^{13}-760x^{12}-138x^{11}+686x^{10}+1238x^9+869x^8+382x^7+210x^6+102x^5+27x^4+12x^3+3x^2+2x-1)$

Table 1: The generating functions for the k-pop-stack-sortable permutations, $k \leq 4$

4. A system of linear equations was derived from the final DFA, and the system was solved to get the desired generating function.

We did so for $k = 1, \ldots, 6$ and in Table 1 we list the resulting generating functions, except for $k = 5$ and $k = 6$ whose expressions are too large to display. For each of them the degree of the polynomial in the numerator is the same as the degree of the polynomial in the denominator. Those degrees, the growth rates of coefficents of the generating functions, and the corresponding sequences for the number of vertices and edges in the final DFAs can be found in the table below.

k	1	2	3	4	5	6
degree	1	3	10	25	71	213
growth rate	2.0000	2.6590	3.4465	4.2706	5.1166	5.9669
vertices	4	5	12	32	99	339
edges	8	11	34	120	477	2010

All the generating functions, source code, and text files defining the DFAs can be found on GitHub [6]. The growth rate of the coefficients of a rational power series $p(x)/q(x)$ is given by $\max\{1/|\zeta| : q(\zeta) = 0\}$ and Sage [8] code for calculating the approximate growth rates, in the table above, can be found on the same GitHub page.

While we do not think it would be particularly interesting to compute generating functions for higher k using our algorithm, it would be interesting to find a closed formula for the generating functions, possibly leading to results about the distribution of the number of passes needed to sort a permutation using a pop-stack. It is not clear whether our approach can be used as a basis for such a formula.

When deriving the generating functions, we observed that, during the phase when the DFAs are intersected, the intermediate DFAs were quite large, often consisting of millions of states. As the final DFAs are so small, this may indicate that our approach is not a natural one, and that simpler, more direct approaches exist. And with a simpler approach it may be easier to find a closed formula.

Finally, as we only considered their enumeration, finding a useful permutation pattern characterization of the k-pop-stack-sortable permutations remains open.

[1] Michael Albert and Mireille Bousquet-Mélou. Permutations sortable by two stacks in parallel and quarter plane walks. *European Journal of Combinatorics*, 43:131–164, 2015.

[2] Michael H Albert, Steve Linton, and Nik Ruškuc. The insertion encoding of permutations. *The Electronic Journal of Combinatorics*, 12(1):R47, 2005.

[3] Mike D Atkinson and Jörg-Rüdiger Sack. Pop-stacks in parallel. *Information Processing Letters*, 70(2):63–67, 1999.

[4] Mike D Atkinson and Timothy Stitt. Restricted permutations and the wreath product. *Discrete Mathematics*, 259(1-3):19–36, 2002.

[5] David Avis and Monroe Newborn. On pop-stacks in series. *Utilitas Math*, 19(129-140):410, 1981.

[6] Anders Claesson and Bjarki Ágúst Guðmundsson. Enumerating the k-pop-stack-sortable permutations. https://github.com/SuprDewd/popstacks.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.6)*, 2017. `http://www.sagemath.org`.

[9] Garpur cluster. IHPC - Icelandic High Performance Computer - University of Iceland and Reykjavik University, 2017.

[10] Donald E Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 1968.

[11] Lara Pudwell and Rebecca Smith. Two-stack-sorting with pop stacks. https://arxiv.org/abs/1801.05005. Preprint.

[12] Marcel Paul Schützenberger. On the definition of a family of automata. *Information and control*, 4(2-3):245–270, 1961.

[13] Rebecca Smith and Vincent Vatter. The enumeration of permutations sortable by pop stacks in parallel. *Information Processing Letters*, 109(12):626–629, 2009.

[14] Robert Tarjan. Sorting using networks of queues and stacks. *Journal of the ACM (JACM)*, 19(2):341–346, 1972.

[15] Henning Úlfarsson. Describing West-3-stack-sortable permutations with permutation patterns. *Séminaire Lotharingien de Combinatoire*, 67:B67d, 2012.

[16] Antti Valmari. Fast brief practical DFA minimization. *Information Processing Letters*, 112(6):213–217, 2012.

[17] Julian West. *Permutations with forbidden subsequences, and, stack-sortable permutations*. PhD thesis, Massachusetts Institute of Technology, 1990.

[18] Doron Zeilberger. A proof of Julian West's conjecture that the number of two-stacksortable permutations of length n is $2(3n)!/((n+1)!(2n+1)!)$. *Discrete Mathematics*, 102(1):85–93, 1992.