A Problem in Card Shuffling

Gwen McKinley
Advisor: Anne Schilling

Submitted in partial satisfaction of the requirements for Highest Honors for the degree of

BACHELOR OF SCIENCE

in

MATHEMATICS

in the

COLLEGE OF LETTERS AND SCIENCES

of the

UNIVERSITY OF CALIFORNIA,
DAVIS

May 9, 2015

Abstract

We start with a deck of cards labeled 1 through n, arranged in an arbitrary permutation. At each stage, we move the card on the top of the deck to the position in the deck corresponding to its number. We ask the following questions: for a given n, what is the longest sequence of moves possible? Does every sequence of moves terminate? Considering all $n!$ possible permutations, what is the average number of moves? In the research presented here, we prove that every sequence of moves does terminate, and have characterized the longest sequence of moves for all n. However, the question of the average number of moves has proven much more difficult, and from the cases we examine for small n, it appears unlikely that the average will be given by a simple formula. This is an interesting and accessible problem, which yields surprisingly complex behavior given the simplicity of its statement.

Introduction

Consider the following algorithm: starting with a deck of n cards in an arbitrary permutation, take the top card in the deck and move it to the position in the deck corresponding to its number (Note: we refer to the "front" and "back" of the deck interchangeably with the "top" and "bottom"). We repeat this until there are no more possible moves. For example, if we were to start with a deck of four cards in the order $4, 2, 3, 1$, we would subsequently get $2, 3, 1, 4$, then $3, 2, 1, 4$, then $2, 1, 3, 4$, and then $1, 2, 3, 4$. At this point we have no possible moves; the card labeled 1 is in position 1, so if we attempt to play again, nothing in the deck will move; at this point we can say that the game enters an infinite sequence of trivial moves, or that it terminates. For simplicity, we will say that it terminates.

We ask the following questions: for a deck of size n, what is the longest sequence of moves possible? Considering all $n!$ possible permutations, what is the average number of moves? Will a sequence of moves necessarily terminate at all? In general, will our algorithm completely sort the deck? How do these outcomes change if we allow for repeated cards in the deck?

Although this algorithm (to the best of our knowledge) has not yet been studied in the literature, there are several similar models that have been investigated extensively. A random-to-random shuffle of a deck of cards is an algorithm wherein we repeatedly select a random card from the deck and move it to a randomly selected position in the deck [2]. In a top-to-random shuffle, we repeatedly take the top card of our deck and randomly select a position to move it to. For this shuffle, it has been proven that if the deck is of size n, it will take approximately $n \log(n)$ moves to thoroughly randomize the deck [1]. Interestingly, this is not sufficient to randomize the distribution of fixed points in the deck [6]. The entropy of the deck, which measures the amount of information remaining in the deck after a given number of moves, has also been studied [8].

Taking this process in reverse and generalizing slightly, the Tsetlin library, or heaps process [3], describes a shelf of n books from which we randomly select books one at a time, each with a certain probability of being selected, returning each to the top of the shelf before selecting the next. Over time, we expect to find the more popular books near the top of the shelf. A number of quantities related to this process have been exactly determined, including the long-run probability of any arrangement of the books [10], the mean depth of each book in the pile in the long-run [10], the stationary distribution of the books on the shelf [4, 5] and the eigenvalues of transition matrix of the associated Markov chain [7].

The primary difference between these examples and the algorithm presented here is that they deal with probabilities and random choice, whereas our game is completely deterministic. Nevertheless, it displays surprisingly complex behavior, some of which we have been able to characterize enumeratively, and some of we describe only asymptotically. Also, it is possible to view our algorithm as a random walk when taken in reverse; starting with any permutation of the deck that fixes at least one card, we randomly select a fixed card and move it to the front of the deck. It is possible that this reverse approach may be used to solve some of the problems associated with our algorithm.

In Section 1, we present the longest possible sequence of moves for a deck of size n. In Section 2, we represent our algorithm using directed graphs, and prove some properties of these graphs. Section 3 characterizes the permutations achieved along the sequence of moves described in Section 1. In Section 4, we give a partial characterization of the permutations on which our game acts as a sorting algorithm. Section 5 gives a conjecture and some numerical data about the average number of moves. In Section 6, we prove some facts about the behavior of our game if we allow for repeated cards in the deck. And last, Appendices A and B provide code used to find the numerical results in Sections 4 and 5.

Acknowledgments

Many thanks to Les Reid, my advisor at the Missouri State University summer REU program, who suggested this problem, and to Gerhardt Hinkle, who suggested it to him. Thanks also to Carlos Rojas and Sam Asher for their help with the programming, and to Simon Rubinstein-Salzedo at Stanford for his excellent insights on the bounds presented in Section 4. And last, thanks to my advisor Anne Schilling, for her invaluable insights, advice, and encouragement.

1 Longest sequence of moves

Theorem 1. *The longest possible sequence of moves for a deck of a given size n will be generated by the starting order:* $2,\ 3,\ \ldots,\ n,\ 1$*, and will consist of* $2^{n-1} - 1$ *moves.*

Lemma 2. *If a sequence of moves does not terminate, then it eventually enters an endlessly repeating finite sequence of moves.*

Proof. There are only $n!$ possible orders of a deck of size n. If a sequence of moves does not terminate, then there must be at least one permutation p that the deck attains twice in the first $n! + 1$ moves. Between the first and second time the deck attains this order, suppose we make k moves. Since the game is completely deterministic, after another k moves, the deck will once again be in the order p. We can repeat this infinitely many times to make the deck return again and again to order p. Moreover, since the game is completely deterministic, each of the $k - 1$ intermediate orderings that the deck attains will also be the same each time. Thus the deck eventually enters a loop in which a sequence of moves of length $k < \infty$ is repeated endlessly. □

Lemma 3. *Every sequence of moves terminates with the card labeled 1 at the front of the deck (equivalently, by Lemma 2, there are no endless loops).*

Proof. Given a deck of n cards, assume the opposite. Then at some point the game enters an endlessly repeating loop; assume we have reached this point of the game. We can divide the cards into two sets: the k cards that will advance to the front infinitely many times, and the $n - k$ cards that will not advance to the front again for the rest of the infinite sequence. Observe that both sets are nonempty; the first by assumption, and the second because 1 must be in it (if 1 advances to the front of the deck, then the sequence of moves will immediately terminate).

Since the first set contains k cards, none of which is 1, it must contain a card of face value $k' \geqslant k + 1$. By assumption k' must advance to the front at some point. In the next move, it will go to position $k' \geqslant k + 1$; here it must have at least 1 card c in front of it that is a member of the second set: there are only $k - 1$ cards other than k' in the first set, whereas there are k slots to be filled in front of position $k + 1$. By assumption, k' must advance to the front again, and in order for this to happen, every card in front of it must also advance to the front at least once, including card c. But this contradicts our assertion that c will not advance to the front for the remainder of the game. Thus we reach a contradiction. Therefore a sequence of moves may not endlessly loop.

And last, when the game terminates, the card labeled 1 must be at the front of the deck; else we can move the front card to a position further back in the deck, giving us another valid move. □

Lemma 4. *A sequence of moves of maximal length must start with 1 in the last position of the deck.*

Proof. Given a deck of $n > 1$ cards, suppose some index $i > 1$ comes after 1 in our starting permutation; this ordering can be written as $p,\ 1,\ i,\ p'$, where i is any card other than 1, and p and p' are any two (possibly empty) permutations of the rest of the cards. Assume towards contradiction that this ordering produces a sequence of moves of maximal length

m. Clearly 1 does not advance to the front of the deck until the last move: as soon as 1 is at the front of the deck, the sequence of moves terminates. So we can put any other card in the place of 1 (as long as we don't change the permutation p at all), and in m moves it will have advanced to the front of the deck. Now consider the ordering $p,\ i,\ 1,\ p'$. As just argued, in m moves we will have i at the front of the deck; hence we must make at least one more move to advance 1 to the front of the deck. Therefore the sequence of moves produced by the initial ordering $p,\ i,\ 1,\ p'$ has at least length $m+1$, contradicting the maximality of m. Thus a sequence of moves of maximal length must start with 1 in the last position of the deck. □

Lemma 5. *Given a deck of at least $n > 1$ cards, if we start with the ordering: $2,\ 3,\ \ldots,\ n,\ i,\ p$, where i is any card not in $\{2, 3, \ldots, n\}$, and p is some (possibly empty) fixed permutation of the remaining cards, then after $2^{n-2} - 1$ moves, the cards will be in the order $n, 2,\ 3,\ \ldots,\ n-1,\ i,\ p$.*

Proof. We will prove the lemma by induction on n.

Base Step: if $n = 2$, then starting with at least n cards in the order $2,\ i,\ p$, we perform $2^{n-2} - 1 = 2^0 - 1 = 0$ moves, remaining at $2,\ i,\ p$ as desired.

Inductive Step: Assume that the lemma holds up to n for some $n > 1$, and consider a deck of at least $n+1$ cards in the order $2,\ 3,\ \ldots,\ n,\ n+1,\ i,\ p$. By our induction hypothesis, after $2^{n-2} - 1$ moves, the cards will be in the order

$$n, 2,\ 3,\ \ldots,\ n-1,\ n+1,\ i,\ p.$$

(Here we have the card $n+1 = i_1$ in the place of i and the string $i,\ p = p_1$ in the place of p.) After one additional move, the cards will be the order

$$2,\ 3,\ \ldots,\ n-1,\ n+1,\ n,\ i,\ p.$$

Similarly, using the induction hypothesis for $n-1$, after another $(2^{(n-1)-2} - 1) + 1$ moves, the cards will be in the order

$$2,\ 3,\ \ldots,\ n-2,\ n+1,\ n-1,\ n,\ i,\ p.$$

Proceeding inductively, after $(2^{(n)-2} - 1) + 1 + (2^{(n-1)-2} - 1) + 1 + (2^{(n-2)-2} - 1) + 1 + \cdots + (2^{2-2} - 1) + 1$ total moves, the cards will be in the order: $n+1,\ 2,\ 3,\ \ldots,\ n,\ i,\ p$. And we observe that:

$$\begin{aligned}(2^{(n)-2} - 1) + 1 + (2^{(n-1)-2} - 1) + 1 + (2^{(n-2)-2} - 1) + 1 + \cdots + (2^{2-2} - 1) + 1 \\ &= 2^{n-2} + 2^{(n-1)-2} + 2^{(n-2)-2} + \cdots + 2^0 \\ &= \sum_{k=0}^{n-2} 2^k = \frac{1 - 2^{n-1}}{1 - 2} \\ &= 2^{n-1} - 1 \\ &= 2^{(n+1)-2} - 1\end{aligned}$$

So given a deck of at least $n+1$ cards in the starting position $2,\ 3,\ \ldots,\ n,\ n+1,\ i,\ p$, after $2^{(n+1)-2}-1$ moves the cards will be in position $n+1,\ 2,\ 3,\ \ldots,\ n-1,\ n,\ i,\ p$. Our induction is complete. □

Lemma 6. *Given at least $n > 1$ cards, the ordering $2,\ 3,\ \ldots,\ n,\ i,\ p$ produces the longest sequence of moves to advance i to the front of the deck, of all orderings of the form $p'(2,3,\ldots,n),\ i,\ p$, where i and p are as defined in Lemma 5, and p' is a permutation of cards $2,\ 3,\ \ldots,\ n$.*

Proof. We will prove the lemma by induction on n:

Base Step: If $n = 2$, then for a given i and p, there is only one ordering of the form $p'(2,3,\ldots,n), i, p$, namely $2, i, p$. Thus the claim is trivially true.

Inductive Step: Assume the statement holds up to some $n > 1$. Then by our induction hypothesis, for a deck of at least $n+1$ cards, and for any p, the ordering $2,\ 3,\ \ldots,\ n,\ n+1,\ i,\ p$ gives the longest sequence of moves to advance $n+1$ to the front of the deck, among orderings of the form $p'(2,3,\ldots,n),\ n+1,\ i,\ p$. (taking $i_1 = n+1$ in place of i and $p_1 = i,\ p$ in place of p).

Moreover, by Lemma 5, at this point the cards will be in the order $n+1,\ 2,\ 3,\ \ldots,\ n,\ i,\ p$. After one additional move, they will be in the order $2,\ 3,\ \ldots,\ n,\ i,\ n+1,\ p$. And by our induction hypothesis, this gives the longest possible sequence of moves to advance i to the front of the deck (among all orderings of the form $p'(2,3,\ldots,n),\ i,\ n+1,\ p$).

Observe that the first change of position of i must occur precisely when $n+1$ is moved behind it; i starts in position $n+1$, so it can only advance when a card with face value at least $n+1$ is played, and the only such card initially in front of i is $n+1$. The card $n+1$ moves behind i immediately after advancing to the front of the deck. So the length of the total sequence of moves to advance i to the front of the deck is the number of moves to advance $n+1$ to the front of the deck, plus one move to place it behind i, plus the number of moves to advance i to the front of the deck once $n+1$ is behind it. The starting order $2,\ 3,\ \ldots,\ n,\ n+1,\ i,\ p$ maximizes the first and last summands, as just shown; hence it gives the longest sequence of moves to advance i to the front of the deck, among orderings of the form $p'(2,3,\ldots,n),\ n+1,\ i,\ p$. Our induction is complete. □

Corollary 7. *For a given $n \geqslant 1$, the starting order $2,\ 3,\ \ldots,\ n,\ 1$ gives the longest sequence of moves to advance 1 to the front of the deck, among all strings of the form $p'(2,3,\ldots,n),\ 1$*

Proof. For $n = 1$ this is trivial, as there is only one possible order for the deck. For $n > 1$, this is a restatement of Lemma 6, in the specific case where $i = 1$ and p is the empty string. □

Lemma 8. *The sequence of moves determined by the initial ordering $2,\ 3,\ \ldots,\ n,\ 1$ is of length $2^{n-1}-1$.*

Proof. By Lemma 5, with $i = 1$ and p an empty string, we have: with initial ordering $2,\ 3,\ \ldots,\ n,\ 1$, after 2^{n-2} moves, the deck will be in the order $2,\ 3,\ \ldots, n-1,\ 1,\ n$. Similarly, after another $2^{(n-1)-2}$ moves, the deck will be in the order $2,\ 3,\ \ldots,\ n-2,\ 1,\ n-1,\ n$. Inductively, after $2^{n-2} + 2^{(n-1)-2} + 2^{(n-2)-2} + \cdots + 2^{2-2}$ moves, the deck will be in the order $1,\ 2,\ 3,\ \ldots,\ n$. And as computed in the proof of Lemma 5,

$$2^{n-2} + 2^{(n-1)-2} + 2^{(n-2)-2} + \cdots + 2^{2-2} = \sum_{k=0}^{n-2} 2^k = 2^{n-1} - 1$$

Thus the sequence of moves produced by the initial ordering $2,\ 3,\ \ldots,\ n,\ 1$ is of length $2^{n-1} - 1$. □

Proof of Theorem 1. By Lemma 3, for a deck of $n < \infty$ cards, every sequence of allowed moves is of finite length. By Lemma 4, any initial card-ordering producing the maximum number of moves must have 1 in the last position of the deck. By Corollary 7, the initial ordering $2,\ 3,\ \ldots,\ n,\ 1$ produces the longest sequence of moves of all initial orderings with 1 in the last position of the deck. Therefore $2,\ 3,\ \ldots,\ n,\ 1$ produces the longest possible sequence of moves overall. And by Lemma 8, it is of length $2^{n-1} - 1$. □

2 Graphs

We can use directed graphs to illustrate the effect of our algorithm on a deck of a given size. For a deck of n cards, we create $n!$ vertices, one for each permutation of the deck. We then make an edge from vertex p_1 to p_2 if the permutation corresponding to p_1 is sent to the permutation corresponding to p_2 by a single move of our game (note: for convenience, we will label vertices by their corresponding permutations, and refer to vertices and permutations interchangeably). We refer to this graph as G_n. We have used the computer algebra system Sage [9] to produce the graphs G_3, G_4, and G_5 shown in Figures 1, 2, and 3.

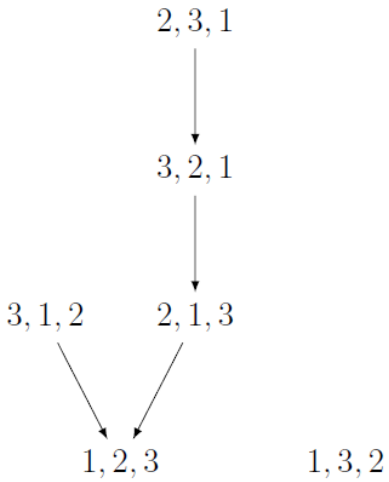

Figure 1: G_3

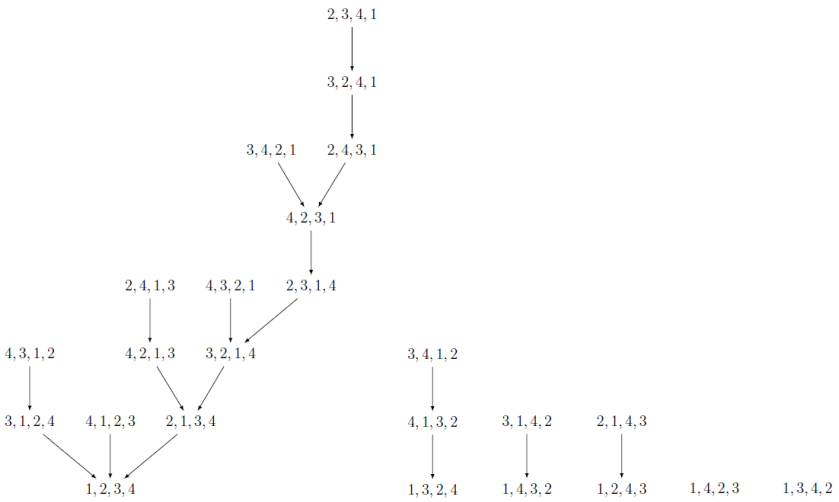

Figure 2: G_4

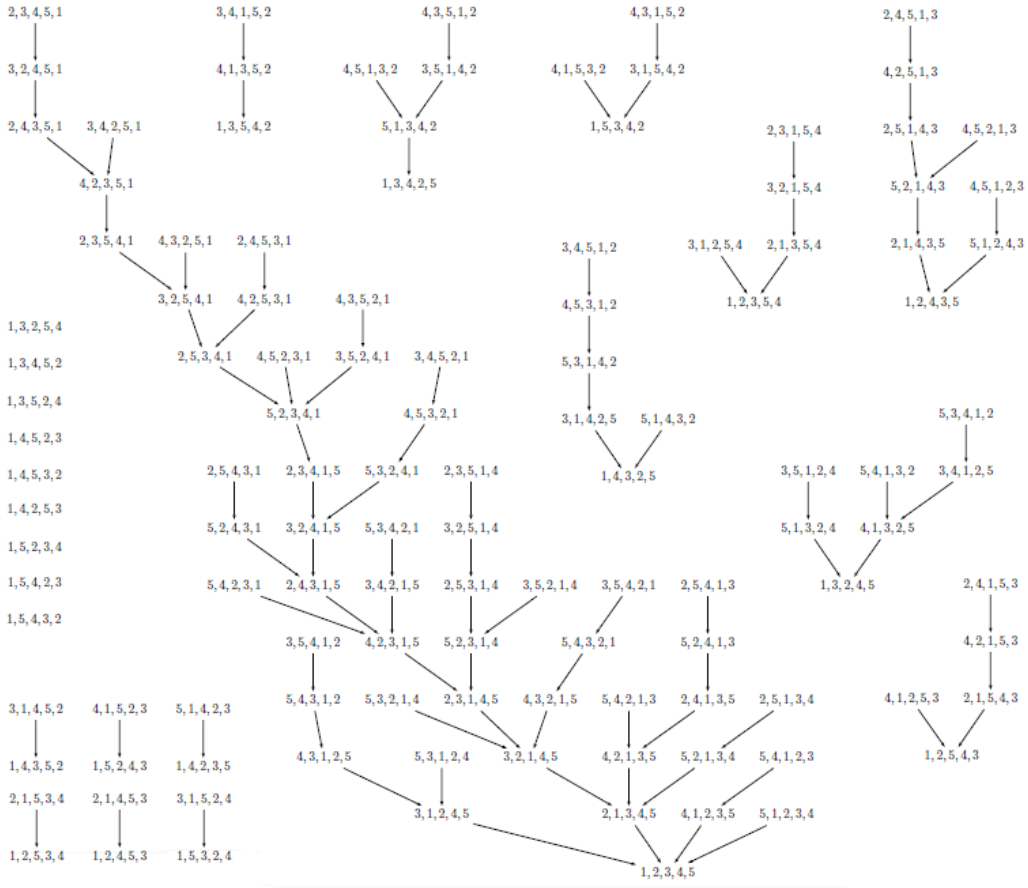

Figure 3: G_5

We can make a few observations, but first we will give some definitions:

Definition 9. A **tree** is a connected graph that does not contain a closed walk in which all vertices are distinct.

Definition 10. A **leaf** of a tree is a vertex of degree 1 (a vertex that connects to exactly one edge).

Definition 11. A **rooted tree** is a tree in which a particular node is designated as the **root**. Then if a vertex u is on the path from the root to another vertex v, we say that u is an **ancestor** of v, or equivalently, that v is a **descendant** of u.

Definition 12. A **rooted forest** is a disjoint union of rooted trees.

Now we are ready to make our observations: for any n, the graph G_n is a directed rooted forest, where we designate the root of each tree by a vertex corresponding to a permutation with 1 in the first position. The game always ends with 1 at the front of the deck, so each tree in the forest contains precisely one such vertex. This implies that G_n will be comprised of $(n-1)!$ disconnected rooted trees.

Proposition 13. *For any n, the leaves in G_n correspond precisely to the derangements of n indices.*

Proof. Consider any permutation $p = a_1, a_2, \ldots, a_{f-1}, f, a_{f+1}, \ldots, a_n$, where some index f is fixed by p, and $f \neq 1$. Then in one move of our game, the permutation $p' = f, a_1, \ldots, a_{f-1}, a_{f+1}, \ldots, a_n$ will be sent to p. So there will be an edge directed from p' to p in G_n, which means that p is not a leaf. Thus a permutation that fixes an index other than 1 is not a leaf. And a permutation that fixes 1 is a root vertex, as previously argued, so using standard terminology for rooted trees, it cannot be a leaf. Hence, any permutation that fixes a value is not a leaf.

Conversely, if a vertex p in G_n is not a leaf, then it is either a root, or there is an edge directed from some other vertex p' to p. If p is a root, then the permutation p fixes the index 1. If there is an edge directed from another vertex p' to p, then it corresponds to a move in our game in which we move the first index i of p' to place i, arriving at the new permutation p. Then by construction, i is a fixed point of p. Thus any vertex that is not a leaf corresponds to a permutation that fixes at least one index. Therefore the leaves in G_n correspond precisely to the derangements of n indices. □

Similarly, there will be precisely one edge directed to a vertex p for each index other than 1 fixed by the permutation p. (Note: we do not include loops on the root vertices.) From this, we see that a given tree in G_n will branch precisely at vertices corresponding to permutations that fix at least two indices excluding 1.

In addition, we can make several observations about the particular tree rooted by the vertex $1, 2, 3, \ldots, n$.

Definition 14. For a deck of size n, we denote by T_n the tree rooted by the vertex $1, 2, 3, \ldots, n$.

Theorem 15. *T_n contains disjoint subgraphs isomorphic to $G_1, G_2, \ldots, G_{n-2}$ and G_{n-1}. Explicitly, for each k, the subgraph of T_n consisting of the descendants of a permutation of the form $n, p_2, \ldots, p_k, 1, k+1, \ldots, n-1$ is isomorphic to the tree in G_k rooted at $1, p_2, \ldots, p_k$.*

This is illustrated for T_5 in Figure 4:

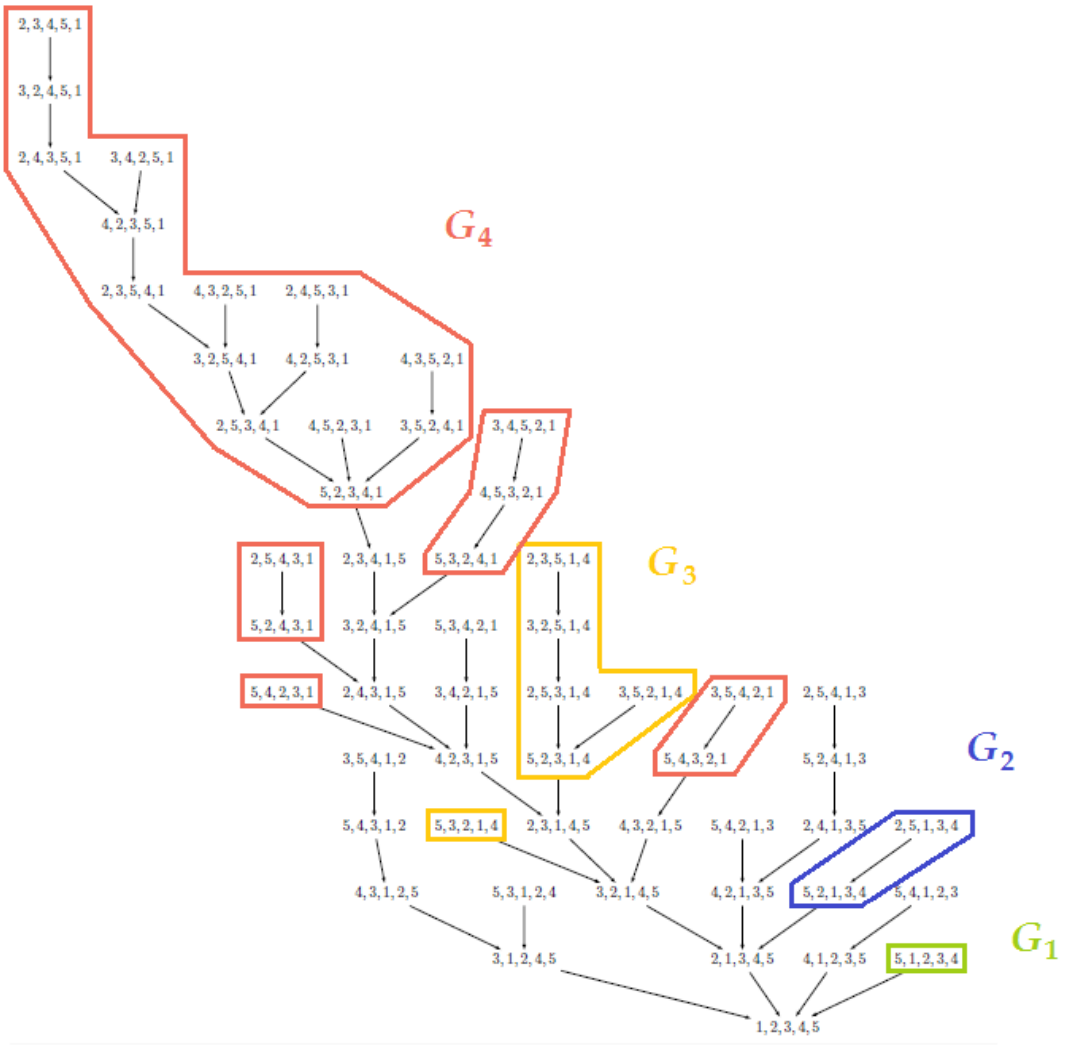

Figure 4: T_5

Before we prove this theorem, we will need another lemma:

Lemma 16. *Our algorithm will completely order any permutation of the form* $p_k, 1, k+1, \ldots, n-1, n$, *where* $k \leqslant n$, *and* p_k *is any permutation of the indices* $2, 3, \ldots, k$.

Proof. In the permutation $p_k, 1, k+1, \ldots, n-1, n$, the index 1 is in position k, so in order for the 1 to advance another position forward, a card with index at least k must be played. However all indices greater that k are already behind the 1, and cannot be played again. So the 1 will only advance forward when card k is played, at which point the deck will be in the order $p_{k-1}, 1, k, k+1, \ldots, n$, for some permutation p_{k-1} of the indices $2, 3, \ldots, k-1$. Inductively, the 1 will advance again when $k-1$ is played, then again when $k-2$ is played, and so on. Eventually the deck will be in the order $p_2, 1, 3, 4, \ldots, n$,

where p_2 is a permutation of the index 2. But the only such permutation is simply the index 2. So the deck will be in the order $2, 1, 3, 4, \ldots, n$, and after one more move, it will be in the order $1, 2, 3, \ldots, n$. Thus the deck is completely sorted. □

Now we can prove the theorem:

Proof of Theorem 15. Consider any permutation of the form $p = n, p_2, p_3, \ldots, p_k, 1, k+1, \ldots, n-1$, where $p_2, p_3, \ldots, p_k$ is a permutation of the indices $2, 3, \ldots, k$. In a single move of our game, the permutation p is sent to the permutation $p' = p_2, p_3, \ldots, p_k, 1, k+1, \ldots, n-1, n$. Thus there is an edge from p to p' in the graph G_n; this implies that they are on the same tree within G_n. By Lemma 16, our game will order p' completely; hence p' is a vertex on T_n, and therefore so is p. We now consider the descendants of the vertex p on T_n.

Observe that no permutation q that is a descendant of p can be of the form $n, a_2, a_3, \ldots, a_n$ for any string $a_2, a_3, \ldots, a_n$. If any q were of this form, then in one move of the game, it would be sent to a permutation $q' = a_2, a_3, \ldots, a_n, n$ at which point the index n would be behind the index 1. And after any number of moves starting from q', the index n would never again come to the front of the deck: in order for the n to come to the front, each cards in front of it would first have to be played at least once, including the 1, but that would end the game. So in any (nonzero) number of moves our game, q would not be sent to a permutation with n as the first entry. But p has n as its first entry, so this contradicts out assumption that q is a descendant of p. Therefore no descendant of p has n as its first index.

Moreover, every descendant of p is of the form $q_1, \ldots, q_k, 1, k+1, \ldots, n-1$: Consider any permutation $a = a_1, \ldots, a_n$ with, and suppose that in one move of our game, it is sent to $b = b_1, \ldots, b_k, 1, k+1, \ldots, n-1$. In this move, a_1 is sent to position a_1, so the index a_1 must be fixed in $b_1, \ldots, b_k, 1, k+1, \ldots, n-1$. We observe that the indices $\{1, k+1, \ldots, n-1\}$ are not fixed; hence $a_1 \notin \{1, k+1, \ldots, n-1\}$. So we must have $a_1 \leqslant k$. And in general, when a card a_1 is played, the indices in positions greater than the face value of a_1 do not move; here, since $a_1 \leqslant k$, the indices in positions greater than k do not move. Therefore the last $n-k$ entries of b are identical to the last $n-k$ entries of a. So $a = a_1, \ldots, a_k, 1, k+1, \ldots, n-1$. Inductively, a permutation that is carried to b after any number of moves must have $1, k+1, \ldots, n-1$ as its last $n-k$ entries. So since the last $n-k$ entries of p are $1, k+1, \ldots, n-1$, these must also be the last $n-k$ entries of any descendant q of p on the graph G_n.

Now consider any permutation a of k indices with $a(1) = i > 1$. We can write

$$a = a_1, a_2, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_k.$$

Suppose in one move of our game, a is sent to some permutation

$$b = b_1, b_2, \ldots, b_{j-1}, 1, b_{j+1}, \ldots, b_k.$$

Then the permutation

$$a' = a_1, a_2, \ldots, a_{i-1}, n, a_{i+1}, \ldots, a_k, 1, k+1, k+2, \ldots, n-1$$

is sent by one move of our game to

$$b' = b_1, b_2, \ldots, b_{j-1}, n, b_{j+1}, \ldots, b_k, 1, k+1, k+2, \ldots, n-1.$$

The converse also holds: if a' is sent to b', then a is sent to b. Both statements are true because in a', neither 1 nor n is the first index, so we can interchange the two indices in a' without affecting the destinations of the other cards under one move of our game. Also, we can append extra cards to a without affecting the destination of the first k cards.

From this, we inductively conclude that, given $i > 1$,

$$q = q_1, \ldots, q_{i-1}, n, q_{i+1}, \ldots, q_k, 1, k+1, \ldots, n-1$$

is sent after some number of moves to

$$q' = q'_1, \ldots, q'_{j-1}, n, q'_{j+1}, \ldots, q'_k, 1, k+1, \ldots, n-1$$

if and only if

$$q_1, \ldots, q_{i-1}, 1, q_{i+1}, \ldots, q_k$$

is sent in the same number of moves to

$$q'_1, \ldots, q'_{j-1}, 1, q'_{j+1}, \ldots, q'_k.$$

Observe here that if $j = 1$ and $q'_{j+1}, \ldots, q'_k = p_2, p_3, \ldots, p_k$, then

$$q' = p = n, p_2, p_3, \ldots, p_k, 1, k+1, \ldots, n-1$$

This implies that the subgraph in G_n of the descendants of p is isomorphic to the subgraph in G_k of the descendants of $1, p_2, p_3, \ldots, p_k$. But the graph of descendants of $1, p_2, p_3, \ldots, p_k$ is simply one of the trees that together comprise G_k; every tree in G_k is rooted by a permutation of the form $1, p_2, p_3, \ldots, p_k$. And recall that for any k, and for any permutation $p_2, p_3, \ldots, p_k$ of the indices $2, 3, \ldots, k$, the permutation $p = n, p_2, p_3, \ldots, p_k, 1, k+1, \ldots, n-1$ is a vertex of T_n. Therefore each tree of each G_k is isomorphically contained in T_n in the manner just described. Thus T_n contains subgraphs isomorphic to $G_1, G_2, \ldots, G_{n-2}$ and G_{n-1}. It remains only to show that these subgraphs are disjoint. However, as we have shown, the vertices of the isomorphic copy of each G_k are permutations in which the last entries are $1, k+1, k+2 \ldots, n-1$. This string is distinct for each k; hence the subgraphs cannot share any vertices, and are therefore disjoint. □

Corollary 17. *T_n contains disjoint subgraphs isomorphic to $T_1, T_2, \ldots, T_{n-2}$ and T_{n-1}. Explicitly, for each k, the subgraph of T_n consisting of the descendants of the permutation $p_k = n, 2, 3, \ldots, k, 1, k+1, \ldots, n-1$ is isomorphic to T_k. Moreover, the longest path from the root vertex of T_n to any vertex on this isomorphic copy of T_k is $2^k - 1$.*

Proof. The first statement follows directly from Theorem 15. And after one move, p_k is sent to $2, 3, \ldots, k, 1, k+1, \ldots n-1, n$. By Lemma 8 (ignoring the indices after the 1), after $2^{k-1}-1$ additional moves, the deck will be in the order $1, 2, \ldots, n$. So the path from p_k to $1, 2, \ldots, n$ is of length $1 + (2^{k-1} - 1) = 2^{k-1}$.

Moreover, for any k, by Theorem 1, the longest path from the root of T_k to any other vertex on T_k is of length $2^{k-1}-1$ (the permutation $2, 3, \ldots, k, 1$ is on T_k). And as we have just shown, for $n > k$, the path from the root vertex of the isomorphic copy of T_k within T_n to the root vertex of T_n is of length 2^{k-1}. So the longest path from the root vertex of T_n to any vertex on the isomorphic copy of T_k is $2^{k-1} - 1 + 2^{k-1} = 2^k - 1$. □

3 Permutations achieved along the longest sequence of moves

We can characterize the permutation that results from performing m moves of our game on a deck of n cards starting in the order $2, 3, \ldots, n$, as long as $m \leqslant 2^{m-1} - 1$, so as not to exceed the length of the game (by Lemma 8, the sequence of moves starting with permutation $2, 3, \ldots, n$ has length $2^{n-1} - 1$).

We construct a permutation as follows: we choose any subset $\{f_1, \ldots, f_k\}$ of the indices $\{2, 3, \ldots, n\}$ and set our permutation to fix each of these indices. We then fill in the rest of the permutation with the indices $2, 3, \ldots, n, 1$ in that order, omitting the indices $\{f_i\}$. For example, the permutation $3, 2, 6, 4, 5, 1, 7$ is of this form, with $\{f_i\} = \{2, 4, 5, 7\}$. Since $\{2, 3, \ldots, n\}$ has 2^{n-1} subsets, there will be 2^{n-1} permutations of this form as long as they are all distinct from each other (it is not difficult to show that they are, but it will not be necessary here).

Lemma 18. *Starting with the permutation $2, 3, \ldots, n, 1$, after $2^{f_k-2} + \cdots + 2^{f_1-2}$ moves, the order of the deck will be of the form described above, with the indices $f_1, \ldots, f_k$ fixed.*

Proof. We prove this by downward induction. At the beginning of the game, the order of the deck is of the given form, with the empty set forming our set of fixed indices. This forms our base step. Without loss of generality, assume $f_1 < \cdots < f_k$. Inductively, assume that after $2^{f_k-2} + \cdots + 2^{f_{j+1}-2}$ moves, the permutation of the deck is of the given form, with the indices $f_k, \ldots, f_{j+1}$ fixed. Moreover, assume that the deck is in the order $2, 3, \cdots, f_j, i, p$, for some index i, where p is a permutation of the remaining indices. Then by Lemma 5, after 2^{f_j-2} moves, the deck will be in the order $2, 3, \ldots, f_j - 1, i, f_j, p$. The index f_j is fixed by this new permutation, and the non-fixed indices are in the same order relative to each other as in the previous permutation; we have simply removed f_j from the set of non-fixed indices and enlarged the set of fixed indices by one. Thus the new

permutation is of the given form, with the indices $\{f_k, f_{k-1}, \ldots, f_j\}$ fixed.

Moreover, since $f_{j-1} < f_j$ by assumption, the new permutation can be written as $2, 3, \ldots,$ f_{j-1}, i', p' where $i' = f_{j-1} + 1$ if $f_{j-1} < f_j - 1$, and $i' = i$ if $f_{j-1} = f_j - 1$; this completes our induction. So after $2^{f_k-2} + \cdots + 2^{f_1-2}$ moves, the order of the deck will be of the form described above, with the indices $\{f_1, \ldots, f_k\}$ fixed. □

With this in mind, we write the number m of moves in its binary expansion. Since $m < 2^{n-1}$, we can write $m = c_{n-2}2^{n-2} + c_{n-3}2^{n-2} + \cdots + c_1 2 + c_0 1$, where $c_i \in \{0, 1\}$ for all i. So

$$m = 2^{x_k} + 2^{x_k-1} + \cdots + 2^{x_2} + 2^{x_1}, \text{ for some } \{x_1, \ldots, x_k\} \subseteq \{0, 1, \ldots, n-2\}.$$

Therefore, setting $f_i - 2 = x_i$ for all i (note: this implies that $2 \leqslant f_i \leqslant n$ for all i, as desired), then by Lemma 18, starting with the deck in the order $2, 3, \ldots n, 1$, after m moves, the order of the deck will be of the form given above, with the indices $\{f_1, \ldots, f_k\}$ fixed.

By Lemma 8, the sequence of moves starting at permutation $2, 3, \ldots, n, 1$ has length $2^{n-1} - 1$, so there are 2^{n-1} permutations of the deck generated by this sequence, including the starting permutation. And as we have just shown, each of these permutations is of the form defined above. Therefore there are at least 2^{n-1} distinct permutations of this form. But as previously argued, since there are 2^{n-1} subsets of $\{2, 3, \ldots, n\}$, there can be at most 2^{n-1} permutations of that form. So there are precisely 2^{n-1} distinct permutations of this form, and each of them is generated by performing m moves of our game on the deck in starting order $2, 3, \ldots, n, 1$, for some $0 \leqslant m \leqslant 2^{n-1} - 1$.

We summarize our results in the following theorem.

Theorem 19. *The permutations generated by performing m moves of our game, with $0 \leqslant m \leqslant 2^{n-1} - 1$, on a deck of n cards starting in the order $2, 3, \ldots, n$ are precisely those permutations which fix some set of indices $\{f_1, \ldots, f_k\} \subseteq \{2, 3, \ldots, n\}$, and in which the remaining indices appear in the order $2, 3, \ldots, n, 1$ (with the indices $\{f_i\}$ omitted).*

4 Our game as a sorting algorithm

One question we may ask: will this algorithm always sort the deck completely? The answer is no: we may take as an example a deck of 4 cards in the order $1, 3, 4, 2$. Since there is a 1 at the front of the deck, the game is already over, and so the deck must remain out of order. To take a slightly less trivial example, consider $2, 1, 4, 3$. After one move, the deck is in the order $1, 2, 4, 3$, and the game is over. Again, the deck is out of order. But observe that the indices behind the 1 were out of order initially. We can make a general observation:

Proposition 20. *If in its starting permutation, the section of the deck behind the index 1 contains an inversion, then our algorithm will not sort the deck completely.*

Proof. Suppose the section of the deck behind the index 1 contains an inversion b, a. Any card behind the 1 can never be played; in order to do so we would have to play the 1 first, but this ends the game. So every move of the game consists of playing some card initially in front of the 1. This card will either move to some position in front of the 1, leaving the cards behind the 1 entirely unaffected, or else it will move behind the 1, perhaps moving but never inverting the cards behind the 1. So a and b can never change order with each other, and at the end of the game they will still be inverted. Thus the deck does not end up in order. □

Here is a less trivial example. With a deck of 5 cards, consider the starting order $4, 5, 1, 2, 3$. The indices behind the 1 are initially arranged in increasing order, but after the first move, the deck is in the order $5, 1, 2, 4, 3$, and at this point the portion of the deck behind the 1 contains the inversion $4, 3$. Thus the deck cannot possibly end up in order, by the previous proposition. So when precisely does the deck end up in order? Had we chosen a slightly different permutation, $5, 4, 1, 2, 3$, the game would have subsequently produced the orderings $4, 1, 2, 3, 5$, and $1, 2, 3, 4, 5$; the algorithm would have sorted the deck completely. We can produce a general criterion:

Lemma 21. *Given a permutation p with the indices behind the 1 arranged in increasing order, if the remaining indices move behind the 1 in decreasing order as the game is played, then our algorithm will sort the deck completely.*

Proof. It suffices to consider the effect of each move on the string of indices behind the 1. At the end of the game, the 1 will be in front of the deck, and this string will necessarily be some permutation of the indices $\{2, 3, \ldots, n\}$, where n is the size of the deck. And if this string remains in increasing order throughout every move of the game, then it must necessarily finish as $2, 3, \ldots, n$; in other words, the deck must finish in order.

Now let us start with an arbitrary permutation $a_1, \ldots, a_\ell, 1, b_1, \ldots, b_m$, where the sequence $b_1, \ldots, b_m$ is increasing. Assume that the indices $\{a_i\}$ all move behind the 1 in decreasing order as the game is played. When a_1 is played, if it goes to a position to the left of the 1, then the string of cards behind the 1 remains unchanged, and is still in increasing order. Now suppose that a_1 goes to a position to the right of the 1; in other words, assume it moves behind the 1.

Each of the cards a_i must move behind the 1 in order for the game to end. Since we have assumed that they will move behind the 1 in decreasing order, this implies that a_1 is the largest of all the indices $\{a_i\}$. Suppose $b_i < a_1 < b_{i+1}$ for some i (we will handle the cases $a_1 < b_1$ and $b_m < a_1$ later). By assumption, the sequence $b_1, \ldots, b_m$ is increasing, so every index to the right of b_{i+1} is also greater than a_1. Thus every index less than a_1 is to the left of b_{i+1}. So the set $\{a_1, \ldots, a_\ell, 1, b_1, \ldots, b_i\}$ has cardinality at least a_1. And since a_1 is the largest element of $\{a_j\}$, and is larger than every element of $\{b_1, \ldots, b_i\}$,

the cardinality is precisely a_1. Thus when a_1 is played, it will advance to the position of b_i, and the deck will be in the order $a_2, \ldots, a_\ell, 1, b_1, \ldots, b_i, a_1, b_{i+1}, \ldots, b_\ell$. So since $b_i < a_1 < b_{i+1}$, the string of indices behind the 1 is still in increasing order.

We now address the two remaining cases: If $a_1 < b_1$, then every index to the right of b_1 is also greater than a_1. Thus all indices less than a_1 are contained in the set $\{a_1, \ldots, a_\ell, 1\}$, giving it cardinality at least a_1. And since $a_1 > a_i$ for all $i \neq 1$, this set has cardinality exactly a_1. Thus when a_1 is played, the deck will be in the order $a_2, \ldots, a_\ell, 1, a_1, b_1, \ldots, b_n$. So since $a_1 < b_1$, the string of indices behind the 1 is still in increasing order.

Last, if $a_1 > b_m$, then since $a_1 > a_i$ for $i \neq 1$ and $b_m > b_i$ for $i \neq m$, we see that a_1 is the largest card in the deck. So when a_1 is played, the deck will be in the order $a_2, \ldots, a_\ell, 1, b_1, \ldots, b_m, a_1$. Then since $a_1 > b_m$, the string of indices behind the 1 is still in increasing order.

Inductively, if we start with a permutation in which the string of indices behind the 1 is arranged in increasing order, and if all the other indices move behind the 1 in decreasing order as the game is played, then at each move of the game, the string of indices behind the 1 will remain in increasing order. So as previously argued, our algorithm will order the deck. □

Lemma 22. *Given a permutation p with the indices behind the 1 arranged in increasing order, if the remaining indices do not all move behind the 1 in decreasing order as the game is played, then our algorithm will not sort the deck completely.*

Proof. If the remaining indices do not all move behind the 1 in decreasing order, then we must have two indices x and y with $x < y$ that start out in front of the 1, such that x moves behind the 1 while y is still in front of the 1. Immediately before this move, the deck is in the order $x, a_1, \ldots, a_k, y, a_{k+1}, \ldots, a_\ell, 1, b_1, \ldots, b_m$, where $\{a_i\}$ and $\{b_i\}$ comprise the remainder of the deck. If at this point, the sequence $b_1, \ldots, b_m$ is not arranged in increasing order, then by the Proposition 20, our algorithm will not sort the deck, and we are done.

Now assume the sequence $b_1, \ldots, b_m$ is arranged in increasing order, and suppose $b_i < x < b_{i+1}$ for some i (we will handle the cases $x < b_1$ and $b_m < x$ later). Then every card to the right of b_{i+1} is also greater than x. So every card less than x is to the left of b_{i+1}. Thus the set $\{x, a_1, \ldots, a_k, y, a_{k+1}, \ldots, a_\ell, 1, b_1, \ldots, b_i\}$ has cardinality at least x. And since $y > x$, it has cardinality at least $x+1$. So when x is played, it will not advance as far as the position of b_i. However, by assumption, it will advance past the 1. Thus we will have x to the left of b_i, and both behind the 1, but $x > b_i$. Hence the portion of the deck behind the 1 will contain the inversion x, b_i, and by Proposition 20, our algorithm will not sort the deck.

We now address the two remaining cases: If $x < b_1$, then every index to the right

of b_1 is also greater than x. Thus all indices less than x are contained in the set $\{x, a_1, \ldots, a_k, y, a_{k+1}, \ldots, a_\ell, 1\}$. So this set has cardinality at least x, and since $y > x$, it has cardinality at least $x + 1$. Thus when it is moved, x does not advance as far as the position of the 1. But this contradicts our assumption that x advances past the 1; hence this case is impossible.

If $x > b_m$, consider our deck of cards. It must have size at least y. Then when x is played, it cannot advance to the last position of the deck, since $x < y$. Thus it goes to a position left of b_m. And by assumption, it advances past the 1. So since $x > b_m$, the cards behind the 1 contain the inversion x, b_m, and by the Proposition 20, our algorithm will not order the deck. □

Together, Lemmas 21 and 22 give a necessary and sufficient condition for our algorithm to order the deck:

Theorem 23. *Given a permutation p with the indices behind the 1 arranged in increasing order, our algorithm will order the deck completely if and only if the remaining indices move behind the 1 in decreasing order as the game is played.*

In practice, this theorem is difficult to use, as it is usually hard to predict whether the indices in the front of the deck will move behind the 1 in decreasing order. For example, our algorithm will order the permutation $3, 4, 7, 6, 1, 2, 5$, but not the permutation $4, 3, 7, 6, 1, 2, 5$, which is identical except for the first two indices. It is at least possible, however, to get reasonably good bounds on the number of permutations ordered by our algorithm for a given size of deck.

Proposition 24. *For a deck of n cards, the proportion of all permutations that our algorithm sorts completely is at least* $\frac{1}{n}$.

Proof. There are $(n-1)! = \frac{n!}{n}$ permutations with 1 in the last position. By Lemma 16 (with $k = n$), our algorithm sorts these permutations completely. So the proportion of permutations that our algorithms sorts is at least $\frac{1}{n}$. □

Proposition 25. *For a deck of n cards, the proportion of all permutations that our algorithm sorts completely is at most* $\frac{e}{n}$.

Proof. If our algorithm sorts a permutation, then by Proposition 20, the indices behind the 1 must at least be in increasing order. Let us count the total number of such permutations. For any k from 0 to $n-1$, there are $\binom{n-1}{k}$ choices of k cards to go behind the 1. For each choice, there is one possible increasing order for these cards, and $(n-k-1)!$ possible orders for the remaining cards (excluding the 1). Thus the total number of permutations that our algorithm will sort is at most

$$
\begin{aligned}
&\sum_{k=0}^{n-1} \binom{n-1}{k} (n-k-1)! \\
=&\sum_{k=0}^{n-1} \frac{(n-1)!}{k!(n-1-k)!}(n-k-1)! \\
=&\sum_{k=0}^{n-1} \frac{(n-1)!}{k!} \\
=&(n-1)! \sum_{k=0}^{n-1} \frac{1}{k!}.
\end{aligned}
$$

Therefore the proportion of all permutations that our algorithm sorts is at most

$$
\frac{1}{n!}\left[(n-1)! \sum_{k=0}^{n-1} \frac{1}{k!}\right] = \frac{1}{n} \sum_{k=0}^{n-1} \frac{1}{k!}
$$

It is well known that $e^x = \sum_{k=0}^{\infty} \frac{x^n}{k!}$, and for positive real numbers x, the sequence of partial sums is clearly strictly increasing, so the partial sums are all less than e^x. In particular, the sequence of partial sums $\sum_{k=0}^{\infty} \frac{1}{k!}$ converges from below to $e^1 = e$. So the proportion of all permutations that our algorithm sorts is less than $\frac{e}{n}$. □

In particular, this upper bound tells us that the percentage of permutations sorted completely by our algorithm becomes vanishingly small as n grows large. We have written a program to estimate the proportion of permutations sorted by our game for small n (the code is provided in the Appendix A). The results for $2 \leqslant n \leqslant 24$, as well as the upper and lower bounds just determined, are shown in Figure 5. It appears that the proportion of permutations sorted quickly becomes close to $\frac{1}{n}$.

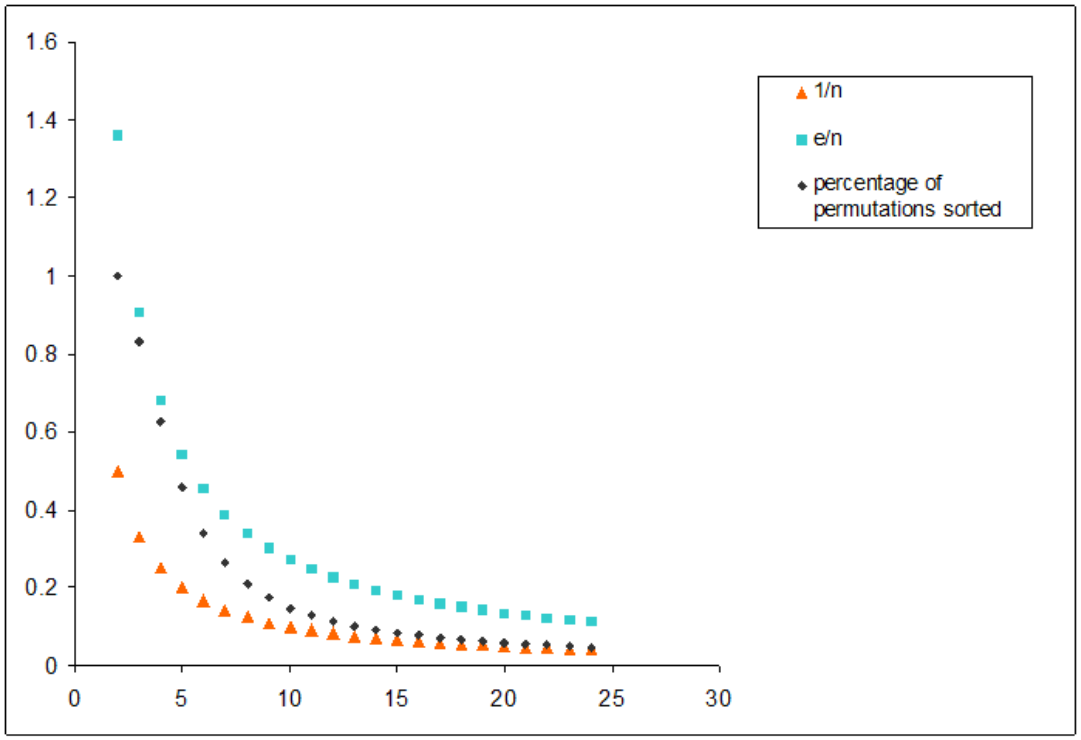

Figure 5: Proportion of Permutations Sorted

5 Average number of moves

We can ask: for a deck of n cards, considering all $n!$ possible permutations, what is the average length of the sequence of moves generated by our game? We have shown that the longest sequence of moves has length $2^{n-1} - 1$. It seems reasonable to suppose that the average number of moves may also be given by a shifted exponential function. However, this is not the case: the average numbers of moves for $n = 1, 2$, and 3 are $0, \frac{1}{2}$, and $\frac{7}{6}$ respectively, so if the average is $av(n) = a \cdot b^n + c$ for some $a, b, c \in \mathbb{R}$, then we can solve for a, b, and c:

$$\begin{cases} 0 = a \cdot b^1 + c \\ \frac{1}{2} = a \cdot b^2 + c \\ \frac{7}{6} = a \cdot b^3 + c \end{cases}$$

The solution to this system is: $a = \frac{9}{8}$, $b = \frac{4}{3}$, and $c = -\frac{3}{2}$. Thus if the average number of moves were a shifted exponential function of n, it would be $av(n) = \frac{9}{8} \cdot \left(\frac{4}{3}\right)^n - \frac{3}{2}$. Therefore we would have $av(4) = \frac{37}{18}$. But we can directly compute the average number of moves for a deck of 4 cards; it is $\frac{50}{4!} = \frac{25}{12} \neq \frac{37}{18}$. Therefore the average number of moves is not given by a shifted exponential function.

But perhaps it comes close: does the average number of moves as a function of n asymptotically approach an exponential function as n grows large? If so, then for large n, we would have $\log(av(n+1)) - \log(av(n)) \approx \log(a \cdot e^{c(n+1)}) - \log(a \cdot e^{c \cdot n})$ for some constants $a, c \in \mathbb{R}$. And

$$\begin{aligned}\log(a \cdot e^{c(n+1)}) - \log(a \cdot e^{c \cdot n}) &= \log(a) + \log(e^{c(n+1)}) - \log(a) - \log(e^{c \cdot n}) \\ &= \log(e^{c(n+1)}) - \log(e^{c \cdot n}) \\ &= c(n+1) - c \cdot n \\ &= c.\end{aligned}$$

Therefore if the average number of moves asymptotically approaches an exponential function, then $\log(av(n+1)) - \log(av(n))$ will approach some constant c as n grows large. In particular, if the average approaches some constant multiple of $2^n = e^{\log(2) \cdot n}$ as n grows large, then $\log(av(n+1)) - \log(av(n))$ will approach $\log 2$. Conversely, if $\log(av(n+1)) - \log(av(n))$ approaches $\log 2$ as n grows large, then it can be shown that $av(n)$ will approach some constant multiple of 2^n, provided that the sequence of error terms $[\log(av(n+1)) - \log(av(n))] - \log 2$ converges to zero sufficiently fast.

We have written a program to estimate the quantity $\log(av(n+1)) - \log(av(n))$ for small n; its output is given in Figure 6.

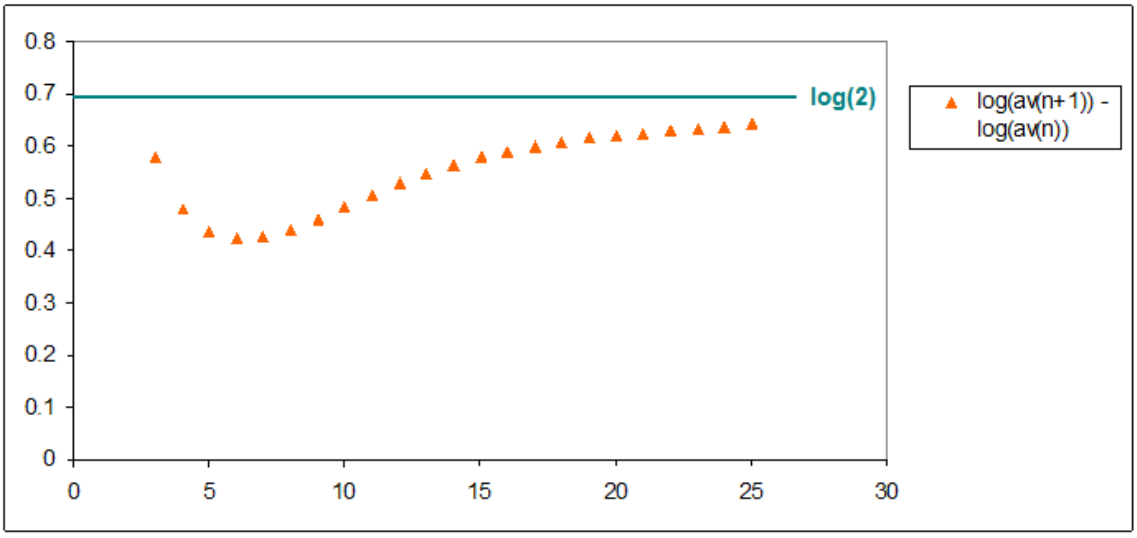

Figure 6: Asymptotic Behavior of $\log(av(n+1)) - \log(av(n))$

From this data, it seems likely that $\log(av(n+1)) - \log(av(n))$ may approach $\log 2$ as n grows large, and that $av(n)$ may approach a constant multiple of 2^n, but we have not yet discovered a rigorous argument for this.

6 Repeated cards and infinite sequences of moves

Our proof that every sequence of moves terminates depended on every card appearing precisely once in the deck. What would happen if we allowed for repeated cards and missing cards? Would we get infinite loops of moves, and if so, under what conditions? In order for the moves of the game to be defined, we stipulate that the face value of each card cannot be greater than the total size of the deck. We also stipulate that the deck must be of finite size (although some of the claims also hold for a deck of infinite size).

Lemma 26. *If the deck does not contain a card labeled* 1*, then from any starting permutation, the game will enter an infinite loop of moves.*

Proof. If we start with some arbitrary permutation of the deck, and perform our sequence of moves on it, at each stage the card in front of the deck will not be labeled 1, and we can move it to the place in the deck corresponding to its number. Thus we can always make another (nontrivial) move, and the sequence of moves cannot terminate. Hence by Lemma 2, the sequence of moves enters an infinite loop (note: the proof of Lemma 2 depended only on the deck being of finite size, and not on the face values of any of the cards). □

Lemma 27. *Given any permutation of the deck, starting from the top and working our way down, if before we encounter a card labeled* 1*, we first encounter some (nonempty) subset S of the deck such that every card in S has face value less than or equal to the cardinality $|S|$, then the sequence of moves generated by this starting permutation will enter an infinite loop.*

Proof. In order for the game to end, we must have a card labeled 1 at the front of the deck. In order for this to occur, we must move every card that is initially in front of the first appearing 1 behind it. By assumption, the entire set S starts in front of the first appearing 1. So until and unless some element of S moves behind it, the position of this 1 will be at least $|S| + 1$. As just stated every card $s \in S$ initially starts in front of the first appearing 1, and the face value of every such $s \in S$ is at most $|S|$, so if any given s is played, it cannot move to a position greater than $|S|$; hence it cannot move behind the 1. Thus the position of the first appearing 1 will always be at least $|S| + 1 > 1$. Hence the game never ends, and will enter an infinite sequence of (nontrivial) moves. By Lemma 2, this is equivalent to the game entering an infinite loop. □

Lemma 28. *If the sequence of moves generated by a given starting permutation eventually enters an infinite loop, then the set S of the cards that are played infinitely many times fulfills the "S-criterion": for each card $s \in S$, the face value of s is at most the cardinality $|S|$.*

Proof. Suppose we have a sequence of moves on a deck of cards that eventually enters an infinite loop. Once the sequence of moves has entered this loop, we can divide the deck into two sets: set S of cards that infinitely loop, and the set R of the remaining cards that will not be played again. Suppose towards contradiction that $s > |S|$ for some s in S.

Then we have two cases. The first is trivial: if S is the whole deck, then we have violated our starting assumption that the face value of each card is at most the size of the deck. In the second case R is nonempty. So when s is played it will go to position $s > |S|$. Thus there must be at least $|S|$ cards in front of s immediately after it is played. Since there are only $|S| - 1$ other cards in S, at least one of the $|S|$ cards must be a card $r \in R$. But s is played infinitely many times by assumption, so we must play it again, and in order for that to happen, we must first play every card that is in front of it, including the card r. But this violates our assumption that r will not be played again once the game has entered its infinite loop. Hence by contradiction, $s \leqslant |S|$ for all $s \in S$. □

Lemma 29. *Given any initial ordering of the deck, suppose that it has a subset S as defined in the statement of Lemma 27. Then consider the first such S we encounter, starting from the top of the deck and working our way down. If two such subsets exist (as in the ordering 2, 3, 2, 1; which has subsets $\{2, 2\}$ and $\{2, 3, 2\}$ both fulfilling the given criterion), then we take S to be the smallest such subset. Claim: S is now well-defined, and cards in the deck that infinitely loop are precisely the set S.*

Proof. Suppose towards contradiction that we have two distinct subsets S_1 and S_2 fulfilling the given criteria (in particular $|S_1| = |S_2|$). Then certainly the two sets must have a mutual last element, as we count from the top of the deck; otherwise (without loss of generality) we would encounter S_1 before S_2 as we count from the top of the deck, and S_2 would then not fulfill the criteria outlined above. So S_1 and S_2 must share their last card s_ℓ, and since they are distinct and of equal cardinality, each must have a card that is not in the other: $s_1 \in S_1$ and $s_2 \in S_2$, but $s_1 \notin S_2$ and $s_2 \notin S_1$. Then

$$\begin{aligned}
|S_1 \setminus \{s_\ell\} \cup S_2 \setminus \{s_\ell\}| &= |S_1 \setminus \{s_\ell\}| + |S_2 \setminus \{s_\ell\}| - |S_1 \setminus \{s_\ell\} \cap S_2 \setminus \{s_\ell\}| \\
&= (|S_1| - 1) + (|S_2| - 1) - |S_1 \setminus \{s_\ell\} \cap S_2 \setminus \{s_\ell\}| \\
&= (2|S_1| - 2) - |S_1 \setminus \{s_\ell\} \cap S_2 \setminus \{s_\ell\}| \quad \text{since } |S_1| = |S_2| \\
&\geqslant (2|S_1| - 2) - (|S_1| - 2) \quad \text{since } s_1 \text{ and } s_2 \text{ are not in this intersection} \\
&= |S_1|
\end{aligned}$$

Observe that the set $|S_1 \setminus \{s_\ell\} \cup S_2 \setminus \{s_\ell\}|$ is encountered in the deck (counting from the top down) before either S_1 or S_2, since all its elements are in either S_1 or S_2, but it does not contain their mutual last card, s_ℓ. Moreover, for every card $s \in (S_1 \setminus \{s_\ell\} \cup S_2 \setminus \{s_\ell\})$, we have $s \in S_1$ or $s \in S_2$, so by the definition of S_1 and S_2, we have $s \leqslant |S_1| = |S_2|$. But as shown above, $|S_1| \leqslant |S_1 \setminus \{s_\ell\} \cup S_2 \setminus \{s_\ell\}|$; hence $s \leqslant |S_1 \setminus \{s_\ell\} \cup S_2 \setminus \{s_\ell\}|$. Therefore $S_1 \setminus \{s_\ell\} \cup S_2 \setminus \{s_\ell\}$ fulfills the properties of the set S, and is encountered in the deck before S_1 or S_2, contradicting the definitions of S_1 and S_2. Therefore the set S as defined above is well-defined.

Now we let S be as defined above, and attempt to show that every card in S must be played at least once. If some card s in S is never played, then some subset A of the cards in front of s in the initial ordering of the deck must never move behind s. In turn, some set of cards $B \subseteq A$ must be played infinitely many times (there are only finitely

many cards in A, but we have just assumed that they remain in front of s for infinitely many moves). However, by Lemma 28, the set of cards that infinitely loop must fulfill the "S-criterion," so B must fulfill the S-criterion. And observe that B comes before S in the initial ordering of the deck (since s is behind all the cards of B in the initial ordering). This contradicts our definition of S. Therefore every card $s \in S$ must be played at least once.

Moreover, when a card $s \in S$ is played, it moves to position s, and it will never subsequently move to a position with higher value: if another card is moved to a position $p \geqslant s$, then the position of card s decreases by 1, but if a card is moved to a position $p > s$, then the position of card s remains unchanged. So after each card in S is played, the cards in S occupy the first $|S|$ slots of the deck, and will remain in these slots indefinitely. Thus the set S_0 of cards that are played infinitely many times will be some subset of S (in order for any other card to be played, it would first have to advance into the first $|S|$ slots, displacing a card from S). Moreover, if S_0 is a proper subset of S, then by Lemma 28, it must fulfill the S-criterion, contradiction our assumption that S has no proper subset with this property. Thus the set of cards that infinitely loop are precisely the set S. □

Conclusion

We have been able to explicitly characterize some of the behavior of this algorithm, including the length and structure of the longest sequence of moves, the conditions under which the game will enter an infinite sequence of moves when repeated cards are allowed, and the recursive structure of parts ofs the directed graphs that we have used to represent the game. However, we have been unable to exactly determine or even rigorously estimate the average number of moves, and we have not yet found a useful complete characterization of the permutations that will be sorted by our algorithm.

Future work could be done to place tighter bounds on the proportion of permutations sorted completely by our algorithm; in particular, it seems likely that the upper bound of $\frac{e}{n}$ could be improved substantially. It may also be possible to find an asymptotic formula for the average number of moves, or perhaps even an exact formula.

Appendix A

Below is a C program that estimates the proportion of permutations sorted by our algorithm:

```
1  // This program computes the proportion of permutations of a deck of //
2  // size n that are sent by our game to 1, 2, 3, ..., n.              //
3
4  #include <stdio.h>
5  #include <stdlib.h>
```

```
6  #include <time.h>
7
8
9  // Produces a randomly shuffled deck of n cards
10 int* makeDeck(int n, int* deck){
11    int i,j;
12    int temp;
13
14    for (i = 0; i < n; ++i){          // produces an ordered deck
15       deck[i] = i + 1;
16    }
17
18    for (i = n - 1; i > 0; i--){      // randomly shuffles the deck,
19       j = rand() % (i + 1);          // using the Fisher-Yates shuffle
20       temp = deck[i];
21       deck[i] = deck[j];
22       deck[j] = temp;
23    }
24    return deck;
25 }
26
27 int* oneMove(int n, int* deck){     // Performs a single move of
28    int i;                           // our game
29    int temp;
30
31    temp = deck[0];
32    for (i = 1; i < temp; i++){
33       deck[i - 1] = deck[i];
34    }
35    deck[temp - 1] = temp;
36
37    return deck;
38 }
39
40 // Determines if a given permutation is sorted by our game, returning //
41 // 1 if it is sorted and 0 otherwise                                   //
42 int isSorted(int n, int* deck){
43
44    int i, sorted = 1;
45
46    makeDeck(n, deck);               // produces a randomly ordered deck
47
48    while (deck[0]>1){               // keeps playing the game until 1
49       oneMove(n, deck);             // is at the front of the deck
50    }
51
52    sorted = 1;
53    for(i = 0; i < n; i++){          // gives "sorted = 1" if the deck is
54       if(deck[i] != i+1){           // now in the order 1,2,...,n, and
55          sorted = 0;                // sets "sorted = 0" otherwise
56          break;
57       }
```

```
58      }
59      return sorted;
60  }
61
62
63  int main(){
64
65      int n = 0;
66      int iterations;
67      double count = 1, totalCount = 1;
68      double numberSorted = 0, proportion = 0;
69      char more = 'y';
70
71      srand ( time(NULL) );            // seeds pseudorandom number
72                                       // generator function
73
74      printf("Enter the size of your deck: ");
75      scanf("%d", &n);
76
77      int deck[n];
78
79      while (more == 'y'){
80          printf("How many iterations would you like to perform \n(for
                indefinitely many, enter any negative integer) ");
81          scanf("%d", &iterations);
82
83          for (count = 1; count <= iterations || iterations < 0; count++){
84              numberSorted += isSorted(n, deck);        // computes the
85              proportion = numberSorted / totalCount;   // proportion of
86                                                        // permutations
87              printf("%lf\n", proportion);              // sorted by our game
88              totalCount++;
89          }
90
91          printf("Continue? (enter y or n) ");
92          scanf(" %c", &more);
93      }
94
95      return 0;
96  }
```

Appendix B

Below is a C program that estimates the quantity $\log(av(n+1)) - \log(av(n))$ for a given n. Note: to compile this program, it is necessary to link to the math library by writing "-lm" at the end of the compile statement (this library contains the natural log function).

```
1  // This program computes the quantity log(av(n+1)) - log(av(n)) //
2  // (the difference of logs of the average number of moves for   //
```

```
3  // consecutive deck sizes)                                          //
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <time.h>
8  #include <math.h>
9
10
11 // Produces a randomly shuffled deck of n cards //
12 int* makeDeck(int n, int* deck){
13     int i, j;
14     int temp;
15
16     for (i = 0; i < n; ++i){        // produces an ordered deck
17         deck[i] = i + 1;
18     }
19
20     for (i = n - 1; i > 0; i--){    // randomly shuffles the deck,
21         j = rand() % (i + 1);       // using the Fisher-Yates shuffle
22         temp = deck[i];
23         deck[i] = deck[j];
24         deck[j] = temp;
25     }
26     return deck;
27 }
28
29
30 // Performs a single move of our game //
31 int* oneMove(int n, int* deck){
32     int i;
33     int temp;
34
35     temp = deck[0];
36     for (i = 1; i < temp; i++){
37         deck[i - 1] = deck[i];
38     }
39     deck[temp - 1] = temp;
40
41     return deck;
42 }
43
44 // Counts the number of moves in our game, starting with a random //
45 // permutation of n indices                                        //
46 long long unsigned int countMoves(int n, int* deck){
47     long long unsigned int count;
48
49     makeDeck(n, deck);                      // produces a randomly
50                                             // ordered deck
51
52     for (count = 0; deck[0] > 1; count++){  // keeps playing the game
53         oneMove(n, deck);                   // until 1 is at the front
54     }                                       // of the deck
```

```
55
56      return count;
57  }
58
59  // Computes the difference of the logs of two given numbers //
60  double logDiff(double avg_1, double avg_2){
61
62      double constant = log(avg_2) - log(avg_1);
63      return constant;
64  }
65
66  int main(){
67
68      int n = 0;
69      int iterations;
70      double count = 1, totalCount = 1;
71      double avgBig = 0, avgLittle = 0;
72      char more = 'y';
73
74      srand(time(NULL));                    // seeds pseudorandom number
75                                            // generator function
76
77      printf("Enter the size of your deck: ");
78      scanf("%d", &n);
79
80      int littleDeck[n];
81      int bigDeck[n+1];
82
83      while (more == 'y'){
84          printf("How many iterations would you like to perform \n(for
               indefinitely many, enter any negative integer) ");
85          scanf("%d", &iterations);
86
87          // Iteratively computes the average number of moves in our  //
88          // game for decks of size n and n+1, at each step computing //
89          // and printing the difference of the natural logs of these //
90          // two averages                                             //
91
92          for (count = 1; count <= iterations || iterations < 0; count++){
93              avgBig = ((totalCount - 1) / totalCount) * avgBig
94                     + (countMoves(n+1, bigDeck) / (totalCount + 1));
95              avgLittle = ((totalCount - 1) / totalCount) * avgLittle
96                       + (countMoves(n, littleDeck) / (totalCount + 1));
97
98              printf("%lf\n", logDiff(avgLittle, avgBig));
99              totalCount++;
100         }
101
102         printf("Continue? (enter y or n) ");
103         scanf(" %c", &more);
104     }
105
```

```
106         return 0;
107     }
```

References

[1] David Aldous and Persi Diaconis. Shuffling cards and stopping times. *Amer. Math. Monthly*, 93(5): 333-348, 1986.

[2] P. Diaconis. Mathematical developments from the analysis of riffle shuffling. *Groups, Combinatorics, Geometry (Durham 2001)*, 73-97, 2003.

[3] Peter Donnelly. The heaps process, libraries, and size-biased permutations. *J. Appl. Probab.*, 28(2): 321-335, 1991.

[4] W. J. Hendricks. The stationary distribution of an interesting Markov chain. *J. Appl. Probab.*, 9: 231-233, 1972.

[5] W. J. Hendricks. An extension of a theorem concerning an interesting Markov chain. *J. Appl. Probab.*, 10: 886-890, 1973.

[6] Lerna Pehlivan. On top to random shuffles, no feedback card guessing, and fixed points of permutations. *Thesis (Ph.D.)University of Southern California*, 141 pp, 2009.

[7] R. M. Phatarfod. On the matrix occurring in a linear search problem. *J. Appl. Probab.*, 28(2): 336-346, 1991.

[8] Dudley Stark. Information loss in top to random shuffling. *Combin. Probab. Comput.*, 11(6): 607-627, 2002.

[9] W.A. Stein et al. *Sage Mathematics Software (Version 5.0)*., The Sage Development Team, 2012. http://www.sagemath.org.

[10] M. L. Tsetlin. Finite automata and models of simple forms of behaviour. *Russian Mathematical Surveys*, 18(4):1-27s, 1963.