COMBINATORIAL ALGORITHMS INVOLVING PATTERN CONTAINING AND AVOIDING PERMUTATIONS

By

REBECCA NICOLE SMITH

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2005

I dedicate this work to my parents, my sister Amy, my aunt Linda and my uncle Michael, and my grandparents.

ACKNOWLEDGMENTS

I would first like to thank my family for being so supportive throughout my graduate school career. I would also want to thank my advisor, Miklós Bóna, for his guidance. His suggestion to look at stack sorting was definitely the turning point in my research. I am very appreciative of David Drake, Andrew Vince, and Neil White for their helpful suggestions and questions during my seminar presentations as well as for serving on my committee. In addition, I would like to thank Meera Sitharam for agreeing to be on my committee and bringing a computer science point of view to the algorithms considered in my thesis. I would also like to thank Kevin Keating and Jonathan King for consistently attending my seminar talks and asking questions I had not considered. In particular, Chapter 3 is the direct result of a question Prof. King asked me when I was presenting my work from Chapter 2. Finally, I am grateful to Bruce Sagan for his suggestions regarding Chapter 5. Thanks to his comments, the proofs in this chapter are significantly less complicated.

TABLE OF CONTENTS

page

ACKNOWLEDGMENTS iv

LIST OF FIGURES vi

ABSTRACT viii

1 AN INTRODUCTION TO STACK SORTING 1

1.1 Sorting Permutations by One Stack 1
1.2 Sorting Permutations by the Right-greedy Algorithm 2
1.3 Sorting Permutations by the Left-greedy Algorithm 11

2 ALGORITHMS FOR SORTING WITH t STACKS IN SERIES 16

2.1 Examples of Using the Left-greedy and Right-greedy Algorithms . 16
2.2 Comparing the Left-greedy and Right-greedy Algorithms 17
2.3 Obtaining More Permutations which are Sortable by t Stacks in Series 24
2.4 More Information about the Left-greedy Algorithm for $t > 2$ . . . 32

3 STACKS IN A CIRCULAR SERIES 38

4 PARITY AND SORTABLE PERMUTATIONS OF A GIVEN LENGTH 44

4.1 Parity of the Number of Permutations of Length n which are t-Stack Sortable for $t = 1$ or 2 44
4.2 Parity of the Number of Permutations of Length n which are Sortable by Two Stacks in Series 47

5 PERMUTATION RECONSTRUCTION 51

5.1 Introduction 51
5.2 The Case when $k = 1$ 54
5.3 The Case when $k = 2$ 56
5.4 The Case when $k = 3$ 58
5.5 Further Results and Directions 61

REFERENCES 64

BIOGRAPHICAL SKETCH 66

LIST OF FIGURES

Figure page

1–1 Sorting 4132 1

1–2 Sorting 4312 using the right-greedy algorithm 3

1–3 A rooted nonseparable planar map on twelve vertices. 6

1–4 A $\beta(1,0)$ tree on nineteen vertices. 7

1–5 Sorting 4312 using the left-greedy algorithm 12

1–6 A $\beta(0,1)$ tree on nineteen vertices. 13

2–1 Sorting 3241 fails using the right-greedy algorithm 17

2–2 Sorting 3241 using the left-greedy algorithm 18

2–3 Sorting 4123 until the third critical moment using the left-greedy algorithm and then doing the same with the right-greedy algorithm. . 19

2–4 Stacks using the right-greedy algorithm immediately before p_i enters . 20

2–5 Stacks when the left-greedy algorithm has been carried out until the $(i+1)^{st}$ critical moment. 21

2–6 Stacks using the right-greedy algorithm immediately after p_i enters. . 21

2–7 Stacks when the left-greedy algorithm has been carried out until the $(i+1)^{st}$ critical moment. 22

2–8 Stacks using the right-greedy algorithm immediately before x is to move into the $(k+1)^{st}$ stack. 22

2–9 Stacks using the right-greedy algorithm immediately before x is to move into the $(k+1)^{st}$ stack. 23

2–10 4231 and 42351 being sorted by three stacks in series using the left-greedy algorithm. 26

2–11 2341 and 23541 being sorted by three stacks in series using the left-greedy algorithm. 30

2–12 2541673 failing to be sorted by three stacks in series using the left-greedy algorithm. 33

2–13 Part I of 2541673 being sorted by three stacks in series. 34

2–14 Part II of 2541673 being sorted by three stacks in series. 35

2–15 Part I of 26351784 being sorted by three stacks in series using the left-greedy algorithm. 36

2–16 Part II of 26351784 being sorted by three stacks in series using the left-greedy algorithm. 37

3–1 Stacks in a circular series 38

3–2 Stacks at the moment of failure. 39

3–3 Part I of sorting 23451 by three stacks in a circular series. 42

3–4 Part II of sorting 23451 by three stacks in a circular series. 43

4–1 The three $\beta(0,1)$ trees on three vertices. 48

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

COMBINATORIAL ALGORITHMS INVOLVING
PATTERN CONTAINING AND AVOIDING PERMUTATIONS

By

Rebecca Nicole Smith

May 2005

Chair: Miklós Bóna
Major Department: Mathematics

For the first part of this dissertation, we look at two standard algorithms used to sort permutations by t stacks in series. The first is the right-greedy algorithm created by West. The second is the left-greedy algorithm introduced by Atkinson, Murphy, and Ruškuc. They showed that the left-greedy algorithm is optimal for $t = 2$. By considering where entries of the permutation are at key points in the process, we show that the use of the left-greedy algorithm on t stacks in series will result in sorting any permutation that could be sorted using the right-greedy algorithm on t stacks in series. For $t \geq 2$, we show how to construct permutations that can be sorted by just two stacks in series using the left-greedy algorithm, but cannot be sorted by t stacks in series using the right-greedy algorithm. We also prove that when applied to t stacks in series where $t > 2$, the use of the left-greedy algorithm is not optimal nor is the class of permutations sorted by the left-greedy algorithm closed.

Then we allow the addition of a type of movement on the stacks where entries from the last stack can return to the first stack instead of going to the output. We show that this allowance does nothing to improve the sorting ability of two stacks

in series. However, when this move is introduced to three or more stacks in series, we are then able to sort any permutation.

We also look at the parity of the number of permutations of length n sortable by one or two stacks. In the case of one stack, this result is well known. The parity of the number of 2-stack sortable permutations of length n has been recently determined combinatorially by Bóna. We present a semi-combinatorial proof that the number of permutations of length n which are sortable by two stacks in series is odd exactly half the time.

Finally, we consider the problem of permutation reconstruction. This problem is an analogue of graph reconstruction, a long-considered question in graph theory. In the case of permutations, the problem can be stated as follows: In all possible ways, delete k entries of the permutation $p = p_1p_2p_3...p_n$ and renumber accordingly, creating $\binom{n}{k}$ substrings. How large must n be in order for us to be able to reconstruct p from this multiset of substrings? That is, how large must n be to guarantee that this multiset is unique to p? It is not difficult to show that if $k = 1$, then n must be at least five to guarantee the reconstruction of p. We will then present a proof that when $k = 2$, we only need n to be at least six to guarantee the reconstruction of p. We also present some partial results for when $k > 2$.

CHAPTER 1
AN INTRODUCTION TO STACK SORTING

In this chapter we will define some key terms and discuss some of the previous work done in the area of stack sorting.

1.1 Sorting Permutations by One Stack

Definition 1.1 *When we refer to a* permutation of length n*, we will simply mean a sequence of the numbers* $1, 2, ..., n$ *arranged in some order.*

Definition 1.2 *We say that a permutation is* sortable by a single stack *if starting at the beginning of a permutation, we can put the entries in a stack and remove them from the stack to the output as necessary by a "last in, first out" method to get the identity permutation while always keeping the entries in the stack in an increasing order from top to bottom.*

Example 1.3 The permutation 4132 is stack sortable.

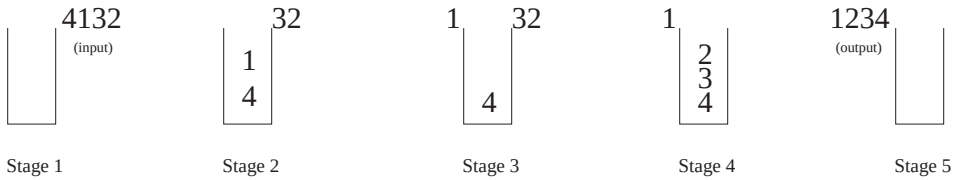

Figure 1–1: Sorting 4132

The ideas of stack sorting and sorting by stacks in series were introduced by Knuth [11]. In this book, he proved the following theorem.

Theorem 1.4 *(Knuth) The number of permutations of length n which can be sorted by one stack is the* n^{th} *Catalan number,* $C_n = \frac{\binom{2n}{n}}{n+1}$.

The Catalan numbers are especially interesting to combinatorialists since they are known to enumerate some 150 different kinds of combinatorial objects.

Stack sorting has also led to other studies of permutations. Due to the nature of the types of permutations that could be sorted by stacks, the concept of pattern avoidance was introduced.

Definition 1.5 *A permutation* $p = p_1p_2...p_n$ *is said to* avoid a pattern $q = q_1q_2...q_k$ *if there is no sequence* $\alpha_1, \alpha_2, ..., \alpha_k$ *such that* $0 < \alpha_1 < \alpha_2 < ... < \alpha_k < n+1$ *and* $p_{\alpha_i} < p_{\alpha_j}$ *if and only if* $q_i < q_j$ *for* $1 \leq i, j \leq n$.

Example 1.6 The permutation 1523746 contains the pattern 231 since the permutation contains the subsequence 574.

Example 1.7 The permutation 12543 avoids the pattern 231.

After its introduction, the study of permutations that avoided particular patterns have been studied independently of their relevance to stack sorting. The most highly considered conjecture in this field was the Stanley-Wilf Conjecture. This conjecture was proved to be true in 2004 by Marcus and Tardos [13].

Theorem 1.8 *(Marcus and Tardos) If* $q = q_1q_2...q_k$ *is any permutation of length* k*, then there exists a constant* c_q *such that the number of permutations of length* n *that avoid* q *is less than* c_q^n.

The relevance of pattern avoidance in stack sorting can be seen immediately. In the case when we sort by just one stack, the permutations that can be sorted are exactly the permutations that avoid the pattern 231.

1.2 Sorting Permutations by the Right-greedy Algorithm

Definition 1.9 *When we refer to* stack sorting using t stacks in series*, we use the same monotonicity rules for each stack, but once the entries leave a stack, they go directly into the next stack. The entries are put into the output when they have passed through all stacks. Note that the entries pass through the stacks from right to left.*

West [22], [23] introduced a right-greedy algorithm to sort permutations by t stacks in series. The right-greedy algorithm always takes the rightmost legal move.

An example of how the right-greedy algorithm can be used to sort a permutation when we have two stacks in series is shown in Figure 1–2.

Example 1.10 4312 is sortable by two stacks in series when the right-greedy algorithm is applied.

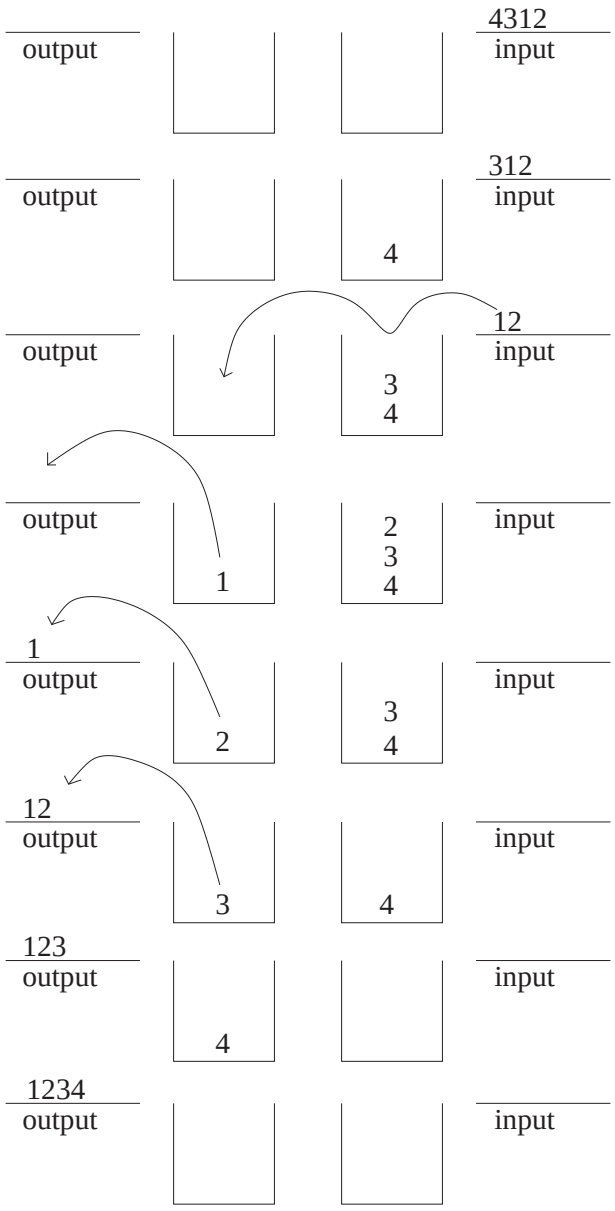

Figure 1–2: Sorting 4312 using the right-greedy algorithm

Definition 1.11 *The* optimal algorithm for 1-stack sorting *allows us only to remove entries from the stack if the next entry cannot enter the stack without*

violating the increasing order condition on the stack or if there are no more entries waiting to enter the stacks.

One can check that any permutation which can be sorted by one stack will be sorted by the stack in this manner.

Definition 1.12 *A permutation p is said to be* t-stack sortable *if p is sorted after being run through one stack t times by the optimal algorithm for 1-stack sorting. We will use the standard notation, $W_t(n)$, for the number of t-stack sortable permutations.*

One can see that the permutations that were sortable by the right-greedy algorithm on t stacks in series are exactly those that are t-stack sortable. In fact, the movements made by the entries are the same. The only difference is that with the right-greedy algorithm we put the entries immediately into the next stack instead of putting them into an output before running them through the stack again.

Though enumeration of 1-stack sortable permutations is relatively straightforward, the same is not true when there are more stacks involved. Part of the problem in these cases is that once there is more than one stack, the class of permutations which are sortable when applying the right-greedy algorithm is no longer closed. In the case of permutations, a closed class is a class such that if a permutation q is a member of the class, then every permutation which is contained in q is also a member of the class. The permutations which can be sorted by t stacks in series using the right-greedy algorithm when $t > 1$ are no longer categorized by pattern avoidance. The classic example is the following.

Example 1.13 In the case when we have two stacks in series, the permutation 3241 is not sortable when we apply the right-greedy algorithm, but the permutation 35241 is sortable by the right-greedy algorithm on the two stacks even though it contains the pattern 3241.

The key to this difference is that the 5 of 35241 forces the 3 out first so as not to form the 231 pattern that occurs if 2 enters the stacks before the 3 leaves.

In fact, it turns out that 2-stack sortable permutations can be categorized as follows:

1. The permutation must avoid 2341 and
2. any 3241 pattern contained within the permutation must be part of a 35241 pattern.

West [22], [23] conjectured that the number of 2-stack sortable permutations of length n was

$$W_2(n) = \frac{2}{(n+1)(2n+1)}\binom{3n}{n}.$$

This conjecture was shown to be true by Zeilberger [24]. In his proof, Zeilberger considered the largest decreasing subsequence $n(n-1)(n-2)...(n-i+1)$ of permutations of length n, applied a bijection and weight enumeration to the permutations, and then used a Pólya-Schützenberger-Tutte transform. All of this resulted in a ninth degree functional equation that required the use of a computer to solve.

It turns out that these permutations can be related to certain types of graphs and planar maps.

Definition 1.14 *A* cut-vertex *is a vertex v of a connected graph G such that if we were to remove v and all of its incident edges, then the graph G would no longer be connected.*

Definition 1.15 *A* nonseparable planar map *is created by taking a connected graph without a cut-vertex and embedding it into the surface of a sphere. The surface is divided into faces. To make such a map into a* rooted nonseparable planar map*, define one vertex r to be the root vertex and an edge incident to r to be the root edge.*

Example 1.16 A rooted nonseparable planar map is shown in Figure 1–3.

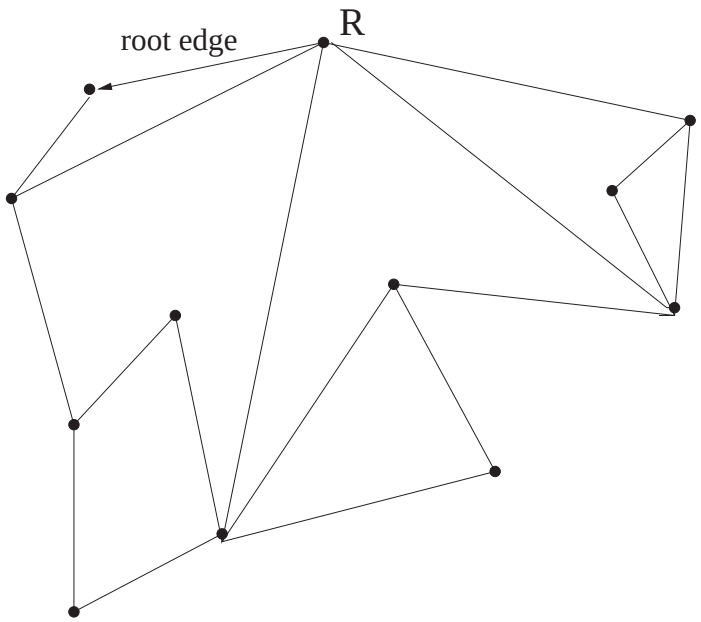

Figure 1–3: A rooted nonseparable planar map on twelve vertices.

Definition 1.17 *If v is a vertex, say that $\lambda(v)$ is its label. For $a, b \in \{0, 1, 2, ...\}$, a $\beta(a, b)$ tree is a rooted, labeled tree such that the following occur.*

1. *For any leaf vertex l, we have $\lambda(l) = a$.*
2. *If v is a vertex that is neither a leaf nor the root, look at its children $v_1, v_2, v_3, ..., v_k$. Then $a \leq \lambda(v) \leq b + \sum_{i=1}^{k} \lambda(v_i)$.*
3. *Finally if r is the root vertex and its children are $r_1, r_2, r_3, ..., r_k$, then $\lambda(r) = \sum_{i=1}^{k} \lambda(r_i)$.*

Example 1.18 A $\beta(1, 0)$ tree is shown in Figure 1–4.

Since then, Dulucq, Gire, and West [8] and Goulden and West [10] have exhibited bijections between nonseparable rooted planar maps on $n + 1$ edges and 2-stack sortable permutations of length n. The nonseparable rooted planar maps with $n + 1$ edges had been enumerated by Tutte [20]. In addition, a bijection between these nonseparable rooted planar maps on $n + 1$ edges and $\beta(0, 1)$ trees on $n + 1$ vertices was shown by Cori, Jacquard, and Schaeffer [7]. This is especially

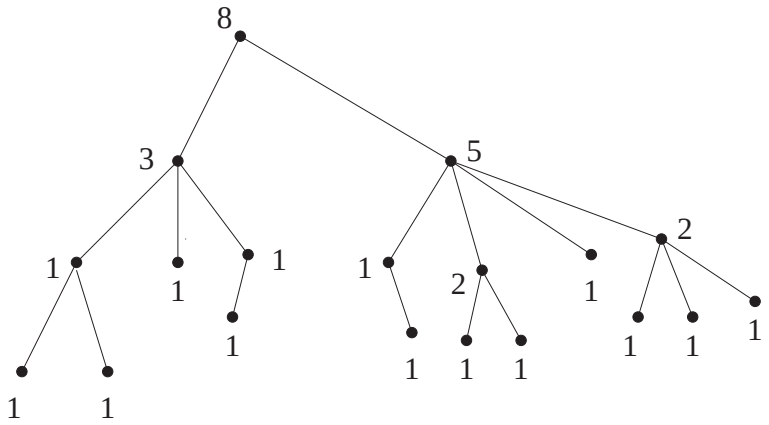

Figure 1–4: A $\beta(1,0)$ tree on nineteen vertices.

interesting because there is also a bijection between permutations which can be sorted by two stacks in series and a similar class of labeled trees.

However, none of the proofs of the formula for $W_2(n)$ directly indicate why $W_2(n) < \binom{3n}{n}$. In particular, no injection from the 2-stack sortable permutations into a set of size $\binom{3n}{n}$ has been found.

A possible key to this problem is that 2-stack sortable permutations of length n can be represented as unique words of length $3n$ consisting of the three letters, d, l, and u. The d represents a move from the input to the first stack, the l represents a move from the first stack to the second stack, and the u represents a move from the second stack to the output. Some necessary conditions are clear, such as the fact that the i^{th} d must appear before the i^{th} l which in turn must appear before the i^{th} u. It is also not difficult to see that any word corresponding to a 2-stack sortable permutation must not have consecutive letters ll or ud. However, satisfying these conditions is not enough to guarantee a word that represents a 2-stack sortable permutation. Determining sufficient conditions is an unsolved problem.

Another interesting connection to 2-stack sortable permutations is a set L_n of lattice paths. These paths start and end at $(0,0)$, never leave the first quadrant, and use some combination of $3n$ steps where each of the steps is one

of the following types: $(1,1)$, $(0,-1)$, and $(-1,0)$. Kreweras [12] proved a more general result which shows what the number of such paths is.

Theorem 1.19 *(Kreweras)*

$$L_n = \frac{4^n}{(n+1)(2n+1)}\binom{3n}{n}.$$

This is precisely $2^{2n-1}W_2(n)$. Yet no correlation between these lattice paths and 2-stack sortable permutations has been found besides the enumeration formulas. It seems that lattice paths have a relation to the words described above since each step will have to appear exactly n times and certainly the step $(1,1)$ would correspond to the letter d since the i^{th} occurrence of $(1,1)$ must come before i^{th} occurrence of both $(0,-1)$ and $(-1,0)$ to keep the path in the first quadrant. Now since there is not a requirement concerning the i^{th} time each of $(0,-1)$ and $(-1,0)$ appear with respect to each other, we have two choices for the i^{th} pair of letters l and u. This only gives us $2^nW_2(n)$ though. The difficulties in finding where the remaining lattice paths come from are the other restrictions on what makes a word correspond to a 2-stack sortable permutation.

Beyond the case when $t = 2$, very little is known. In general, only a trivial upper bound for $W_t(n)$ is known and this bound is only tight in the case when $t = 1$.

Proposition 1.20 $W_t(n) < (t+1)^{2n}$.

This upper bound is obtained by using the following results: one is by Tarjan [19] and the other is a corollary of a result due to Stankova and West [17].

Theorem 1.21 *(Tarjan) Any permutation containing the pattern* $234...(t+1)(t+2)1$ *is not t-stack sortable.*

Proof: Suppose that we apply the right-greedy algorithm when attempting to sort $234...(t+1)(t+2)1$ by t stacks in series. The 2 will enter the first stack and immediately have to move to the next stack so that the 3 can enter the first stack.

In fact, every time an entry is to be brought into the stacks, the entries already in the stacks must each be moved left. Thus $2, 3, 4, ...t + 1$ will each be in their own stack. As there are only t stacks, these t entries take up all of them and so $t + 2$ cannot enter the stacks unless the 2 leaves. Thus we cannot get the identity permutation and so this permutation is not t-stack sortable.

$\diamond$

Theorem 1.22 *(Stankova-West) The number of permutations of length n which avoid* $234...(t + 2)1$ *is the same as the number of permutations which avoid* $(t + 2)(t + 1)...321$.

The following corollary actually gives an example of the Marcus-Tardos theorem with an optimal c_q.

Corollary 1.23 *The number of permutations of length n that avoid* $234...(t + 2)1$ *is less than* $(t + 1)^{2n}$.

Proof: We will show that the number of permutations that avoid $(t+2)(t+1)...321$ is less than $(t + 1)^{2n}$, which proves the corollary due to the previous theorem.

Let p be a permutation of length n that avoids $(t + 2)(t + 1)...321$. For any entry p_i of p, let k_{p_i} be the length of the longest decreasing subsequence of p that ends with the entry p_i. Then we know $k_{p_i} < t + 2$ for all values of i.

For each set of entries $D_k = \{p_i | k_{p_i} = k\}$, we first show that these entries must appear in ascending order in p. If not, suppose we have two entries p_i and p_j both in D_k such that $i < j$ and $p_i > p_j$. Since $p_i \in D_k$, we know there exists a descending sequence $p_{\alpha_1}, p_{\alpha_2}, ..., p_{\alpha_{k-1}}, p_i$ in p. However, then we have a descending sequence $p_{\alpha_1}, p_{\alpha_2}, ..., p_{\alpha_{k-1}}, p_i, p_j$ of length $k + 1$ in p which contradicts the assumption that $p_j \in D_k$.

Hence there exist sets $D_1, D_2, ..., D_{t+1}$ which partition the entries of p. There are $(t + 1)^n$ choices for which of the sets each of the n entries is in. Then when

deciding which entry to put in each position, we have at most $t+1$ choices since as soon as we decide to take an entry from D_k, we must take the lowest entry that has not been used already. Note that there are clear cases such as the first position when we have strictly less than $t+1$ choices.

Hence the number of permutations of length n that avoid $(t+2)(t+1)...321$ is less than $(t+1)^{2n}$.

$\diamond$

Also interesting in the study of t-stack sortable permutations is the fact that the number of 1-stack sortable permutations is

$$W_1(n) = \frac{\binom{2n}{n}}{n+1}$$

and the number of 2-stack sortable permutations is

$$W_2(n) = \frac{\binom{3n}{n}}{\frac{1}{2}(n+1)(2n+1)}.$$

This would lead one to believe that that perhaps for larger values of t we would have

$$W_t(n) = \frac{\binom{(t+1)n}{n}}{\text{some polynomial}}.$$

However, numerical evidence first explored by West [22] and later by Guibert in 2003 (not published) seems to indicate that this is not the case for $t=3$. This is because $W_3(n)$ appears to have prime factors much larger than $4n$.

None the less, it would be interesting to know if the numerator of this formula still provides an upper bound as it did in the cases when $n=1$ and $n=2$.

Question 1.24 *Is* $W_t(n) < \binom{(t+1)n}{n}$ *?*

In the next section we look at another way to sort permutations by t stacks in series.

1.3 Sorting Permutations by the Left-greedy Algorithm

In 2000, Atkinson, Murphy, and Ruškuc [1] used a left-greedy algorithm for sorting permutations using two stacks in series. The left-greedy algorithm always takes the leftmost legal move. Here we consider a legal move to be moving an entry to a stack where it is smaller than the entry below it or moving an entry to the output if it is the smallest entry not already there. An example of the left-greedy algorithm being applied to sort a permutation by two stacks in series is shown in Figure 1–5.

Example 1.25 Sorting 4312 by two stacks in series using the left-greedy algorithm.

This algorithm proved to be crucial when looking at permutations sortable by two stacks in series due to the following theorem proved in the aforementioned paper.

Theorem 1.26 *(Atkinson, Murphy, and Ruškuc) The left-greedy algorithm can be used to sort any permutation that can be sorted by two stacks in series.*

As with permutations which are sortable by two stacks in series when using the right-greedy algorithm, the permutations of length n that can be sorted by two stacks in series using the left-greedy algorithm can also be represented by a unique word of length $3n$ with n of each of the letters d, l, and u representing the same types of moves as described earlier. However, with this algorithm they were able to determine necessary and sufficient conditions to be a word representing a permutation sortable by two stacks in series. Not only must the i^{th} d appear before the i^{th} l which appears before the i^{th} u, also these words cannot have consecutive letters ll, du, or Sl where S itself is a word representing a permutation of length k. The second set of conditions can be seen to be necessary due to the fact that one must always take the leftmost legal move available. The proof that these are

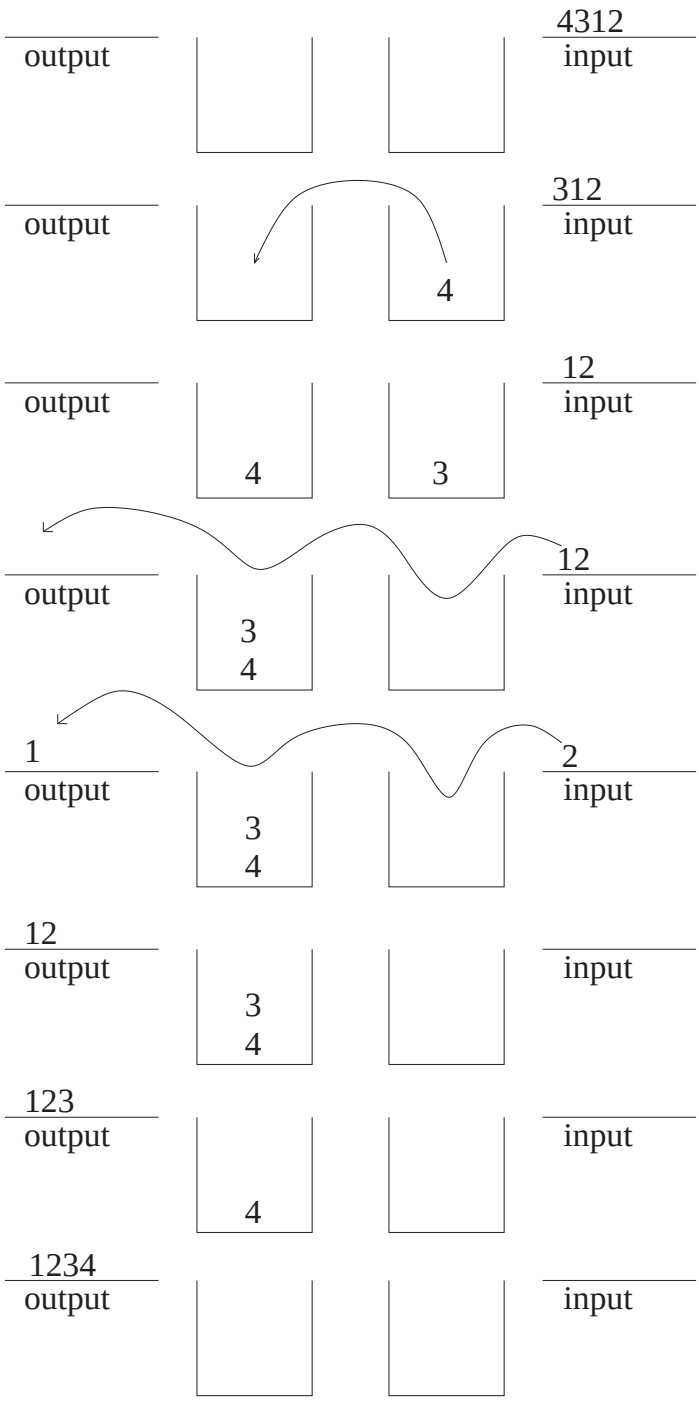

Figure 1–5: Sorting 4312 using the left-greedy algorithm

sufficient conditions is more difficult. Words satisfying these conditions are called GM-words.

This classification of what makes a word correspond to a permutation allowed Atkinson, Murphy, and Ruškuc to count the number of such permutations of a given length n. First an irreducible GM-word was defined to be a GM-word W such that it could not be broken down to get $W = W_1W_2$ where both W_1 and W_2 were GM-words themselves. They then set up a bijection between irreducible GM-words and a class of description trees called $\beta(0,1)$ trees on n vertices, previously defined in Definition 1.17. These trees were enumerated by Cori, Jacquard, and Schaeffer [7] by constructing a bijection between them and 2-colorable cubic planar maps with $3n$ edges which were enumerated by Tutte [20].

Example 1.27 A $\beta(0,1)$ tree is shown in Figure 1–6.

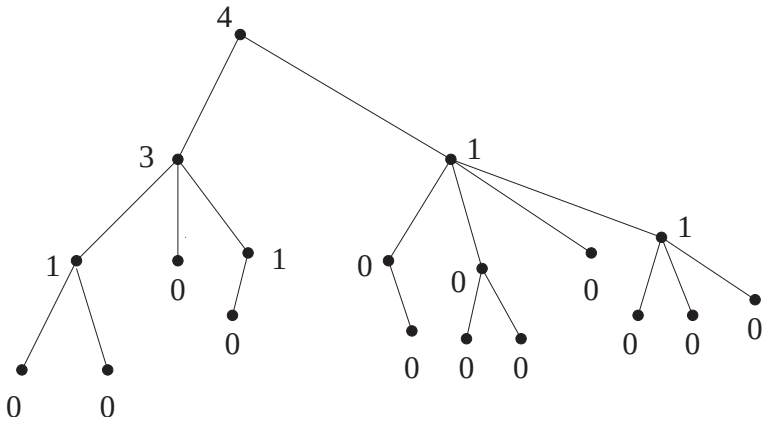

Figure 1–6: A $\beta(0,1)$ tree on nineteen vertices.

This bijection allowed Atkinson, Murphy, and Ruškuc [1] to then enumerate the number of permutations of length n which are sortable by two stacks in series. They constructed a bijection between the set of all GM-words of length $3n$ and plane forests of $\beta(0,1)$ trees on n vertices to deduce the following theorem.

Theorem 1.28 *(Atkinson, Murphy, and Ruškuc) Let z_n be the number of permutations of length n sortable by two stacks in series; then*

$$\sum_{n=0}^{\infty} z_n x^n = \frac{32x}{-8x^2 + 20x + 1 - (1-8x)^{\frac{3}{2}}}.$$

An explicit formula for z_n is given by

$$\frac{7n^2 - 3n - 2}{2} \cdot (-1)^{n-1} + 3\sum_{i=2}^{n} 2^{i+1} \cdot \frac{(2i-4)!}{i!(i-2)!} \cdot \binom{n-i+2}{2} \cdot (-1)^{n-1}.$$

Unlike when using the right-greedy algorithm, these sortable permutations are a closed class. In fact, it was shown that the basis of minimal unsortable permutations was infinite and contained exactly the permutations of the form $p = (2, 2m-1, 4, 1, 6, 3, 8, 5, ..., 2m-2, 2m-5, 2m, 2m-3)$ where $m \in \{2, 3, 4, ...\}$. This is especially interesting because they found that the number of permutations of length n that avoided all the permutations in this basis is the same as the number of permutations of length n that avoid only the permutation 1342 which had been enumerated by Bóna [3]. This is the first result where the number of permutations of a given length that avoid one basis set of permutations is the same as the number of permutations of the same length that avoid a basis set of a different cardinality.

Despite the numerical correlation between permutations that avoid all the permutations in the infinite basis described above and the permutations that avoid only the permutation 1342, there is no direct bijection linking the two. However, Bóna's proof of the enumeration of the permutations of length n that avoid 1342 had also shown a bijection between $\beta(0,1)$ trees and a subclass of permutations that avoid 1342.

Question 1.29 *Is there some sorting device that allows one to sort exactly the permutations that avoid 1342?*

Question 1.30 *If so, does the method of sorting yield a bijection between the permutations of length n that avoid 1342 and those which can be sorted by two stacks in series?*

Once again, for cases when t is larger than two, little is known about both the number of permutations that can be sorted by t stacks in series in general and the number of permutations that can be sorted by t stacks in series when the left-greedy algorithm is applied. The upper bound stated in Proposition 1.20 referring to permutations which are sortable by t stacks in series when using the right-greedy algorithm also applies to these permutations. This is since the monotonicity requirements we have placed on the stacks will not allow $234...(t+1)(t+2)1$ to be sorted by t stacks in series. Unfortunately as we will show, the left-greedy algorithm ceases to be optimal or provide a closed class as soon as there are more than two stacks in series.

CHAPTER 2
ALGORITHMS FOR SORTING WITH t STACKS IN SERIES

2.1 Examples of Using the Left-greedy and Right-greedy Algorithms

In the next section, we will compare the two algorithms. It will be shown that the use of the left-greedy algorithm will allow us to sort any permutation that could have been sorted by the right-greedy algorithm when sorting by t stacks in series. In addition, when we have two or more stacks in series, there are permutations that will be sorted by only the left-greedy algorithm when each algorithm is applied.

We now present an example of a permutation which can only be sorted by the left-greedy algorithm when we try applying each algorithm with two stacks in series. This also gives insight into why the right-greedy algorithm does not yield a closed class of permutations, since inserting a large element into an appropriate place might force the right-greedy algorithm to make a more optimal move than it would without the large element.

Note: For the purposes of this chapter, we will say that an algorithm *fails* when we cannot move any entry to a stack without violating the increasing order condition on the stacks and we cannot move the next entry of the identity permutation into the output.

Example 2.1 The permutation 3241 cannot be sorted by two stacks in series when using the right-greedy algorithm.

In Figure 2–1, notice that the 4 cannot enter the first stack because it is larger than 3. Likewise the 3 cannot move left to the second stack because it is larger than 2. Finally the 2 cannot exit the stacks because the 1 is not already in the output.

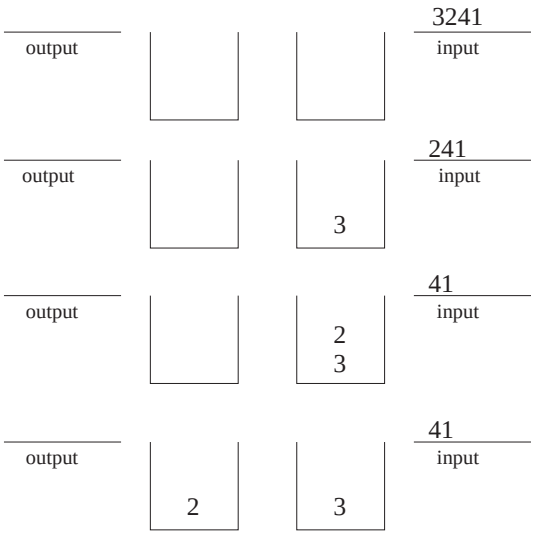

Figure 2–1: Sorting 3241 fails using the right-greedy algorithm

Example 2.2 The permutation 3241 is sortable by two stacks in series when using the left-greedy algorithm.

In Figure 2–2, notice that moving the 3 to the second stack before the 2 traps it in the first stack is what gives the left-greedy algorithm the ability to sort this permutation.

The results shown in the remainder of this chapter are reproduced with permission from the author's paper [18].

2.2 Comparing the Left-greedy and Right-greedy Algorithms

The fact that any permutation that can be sorted by the right-greedy algorithm will also be sorted by the left-greedy algorithm on t stacks in series will follow from the subsequent lemma. It was already known for the case when $t = 1$, since both algorithms are the same and when $t = 2$ since the left-greedy algorithm is known to be optimal.

Definition 2.3 *Define the* i^{th} critical moment *to be the time right before the entry* p_i *would enter the stacks or when the algorithm fails, whichever happens first.*

In Figure 2–3 we show the sorting of 4123 using two stacks in series by each algorithm carried out until the third critical moment.

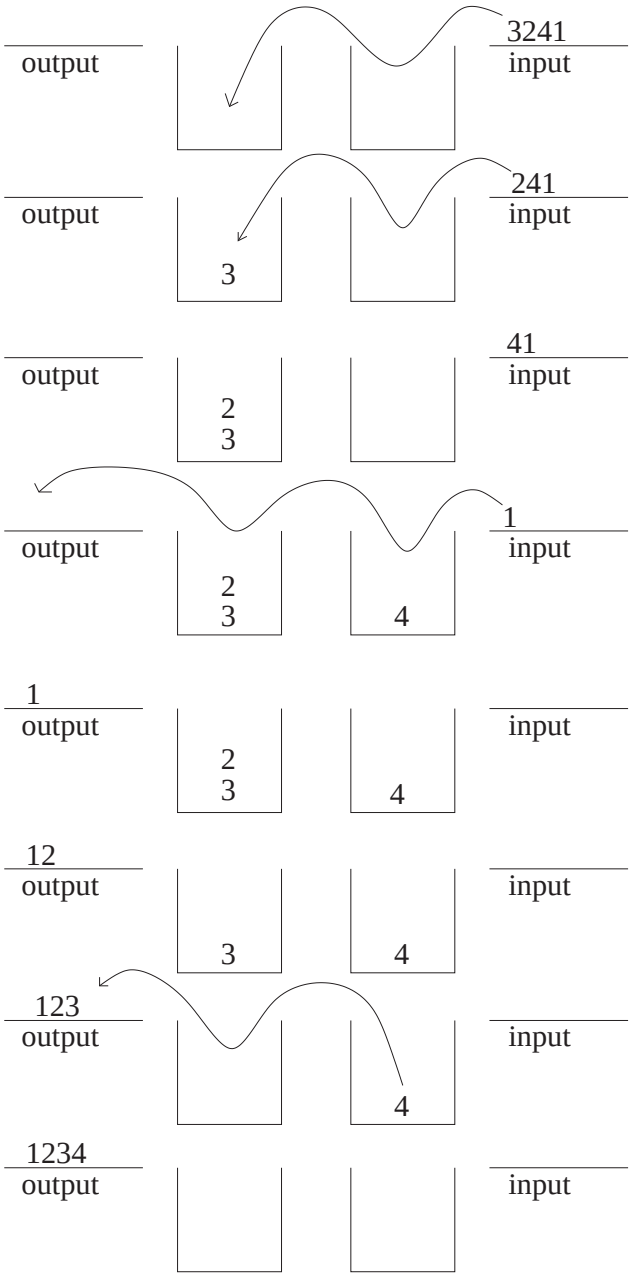

Figure 2–2: Sorting 3241 using the left-greedy algorithm

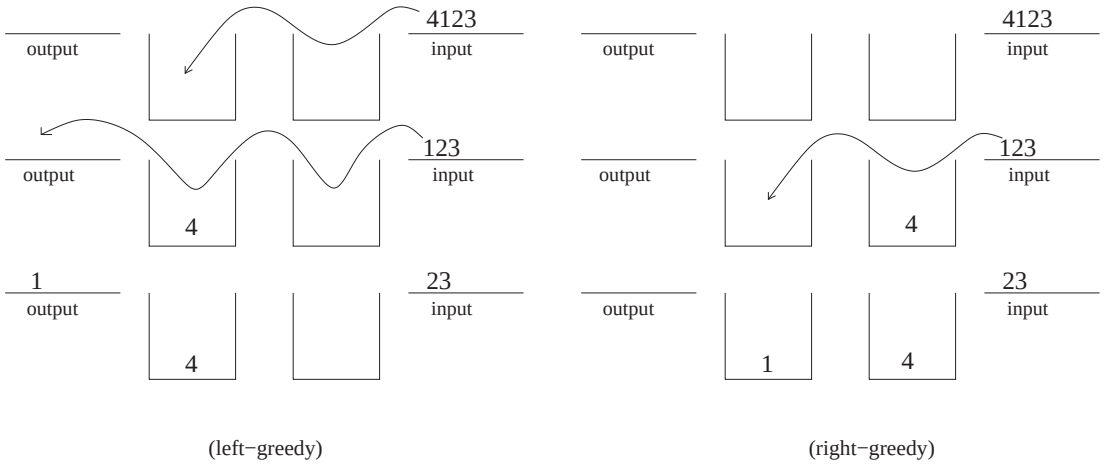

Figure 2–3: Sorting 4123 until the third critical moment using the left-greedy algorithm and then doing the same with the right-greedy algorithm.

Lemma 2.4 *Suppose we have t stacks in series and let* $p = p_1p_2...p_n$ *be a permutation. For every* $i = 1, 2, ..., n$*, at the* i^{th} *critical moment each entry of the permutation is at least as far left by the left-greedy algorithm as it would be by the right-greedy algorithm at its* i^{th} *critical moment.*

Note: *The output is the furthest left position and the input is the furthest right position.*

Proof: We use induction on the i^{th} critical moment. It is clear that the lemma is true for when $i = 1$ and when $i = 2$. Suppose it is true for i and show for $i + 1$.

If p_i can never enter the stacks by the left-greedy algorithm, then let p_j be the first entry of the permutation that could not enter the stacks. If the left-greedy algorithm failed before p_j entered the stacks for $j < i$, then the right greedy algorithm must have failed before p_j could enter also. Otherwise the induction hypothesis condition would be false since p_j would be further left at the i^{th} critical moment. If $j = i$, then there exists a $\gamma < p_i$ in the rightmost stack when using the left-greedy algorithm. However, if the right-greedy algorithm got even this far without being stuck, then γ must be in the rightmost stack by this algorithm as

well by the induction hypothesis. Therefore the right-greedy algorithm fails by this point also, so the lemma trivially holds for all subsequent p_i's including p_{i+1}.

Thus we may assume that p_i can enter the stacks by the left-greedy algorithm. Also, since we are still trivially done if the right-greedy algorithm fails before p_i enters the stacks, we may assume p_i can enter the stacks by the right-greedy algorithm as well.

Hence when using the induction hypothesis, we may assume that the stacks using the right-greedy algorithm are as in Figure 2–4 at the i^{th} critical moment, with all entries being at least as far left in the stacks using the left-greedy algorithm as those using the right-greedy algorithm.

Figure 2–4: Stacks using the right-greedy algorithm immediately before p_i enters

When p_i enters the first stack by each algorithm, we still have the condition that each entry is at least as far left by the left-greedy algorithm as by the right-greedy algorithm since no entries move left except for p_i moving into the first stack in both algorithms. The condition continues to hold if we carry out the left-greedy algorithm (while not continuing to carry out the right-greedy algorithm on the other series of stacks) until it is time for p_{i+1} to enter the stacks or the algorithm fails since we are only moving entries left in the stacks using the left-greedy algorithm. We now have the stacks that use the left-greedy algorithm being as shown in Figure 2–5 and the stacks that use the right-greedy algorithm as shown in Figure 2–6.

Now if when we carry out the right-greedy algorithm until p_{i+1} should enter or the algorithm fails, and it results with each entry being no further left than where

Figure 2–5: Stacks when the left-greedy algorithm has been carried out until the $(i+1)^{st}$ critical moment.

Figure 2–6: Stacks using the right-greedy algorithm immediately after p_i enters.

it was by the left-greedy algorithm, then we are done. Suppose there is an entry x such that x was in the k^{th} stack by the left-greedy algorithm at this point, but x moves further left by the right-greedy algorithm before p_{i+1} enters. Choose x to be the first entry for which this happens (in the order of the steps of the right-greedy algorithm) and look at what is happening when x moves from the k^{th} stack to the $(k+1)^{st}$ stack.

We first note that if such an entry exists, the $(k+1)^{st}$ stack is actually a stack and not the output. This is because x would not be in the last stack by the left-greedy algorithm since any entry that has already had all entries smaller than it enter the stacks, leaves by the left-greedy algorithm before a new entry enters. Thus no entry can leave the stacks by the right-greedy algorithm that has not also left by the left-greedy algorithm.

Since x did not move to the $(k+1)^{st}$ stack by the left-greedy algorithm, there must be an entry $y_0 < x$ in the $(k+1)^{st}$ stack at this point by this algorithm. As all entries are no further left by the right-greedy algorithm at the point when we are about to move x from the k^{th} stack to the $(k+1)^{st}$ stack, it must be that

y_0 is to the right of x at this point using the right-greedy algorithm. Hence the stacks where the left-greedy algorithm was applied must be as in Figure 2–7 and the stacks where the right-greedy algorithm was applied must be as in Figure 2–8 at this point.

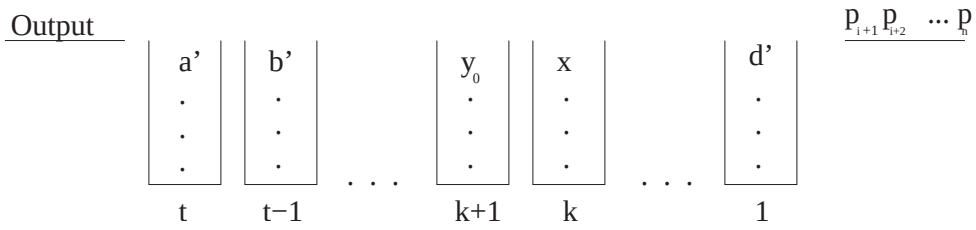

Figure 2–7: Stacks when the left-greedy algorithm has been carried out until the $(i+1)^{st}$ critical moment.

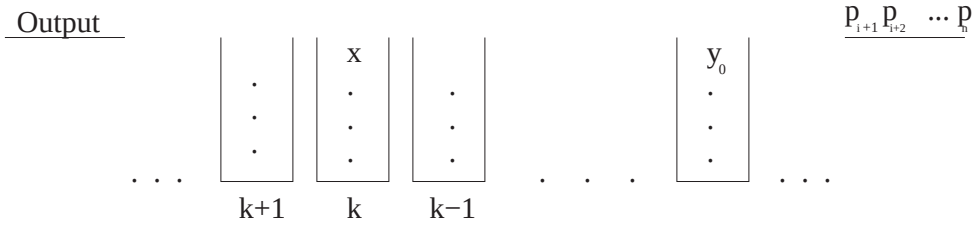

Figure 2–8: Stacks using the right-greedy algorithm immediately before x is to move into the $(k+1)^{st}$ stack.

Because y_0 is in the stacks and further right than x, if y_0 could have moved, it would have priority over x by the right-greedy algorithm. If y_0 was not the top entry of its stack at this point, then the smaller entry at the top of the stack would have priority over x. Thus there must be a $y_1 < y_0$ at the top of the stack directly to the left of y_0. Likewise, since y_1 is to the right of x and it also didn't move, there must be a $y_2 < y_1$ at the top of the stack directly to the left of y_1 as seen in Figure 2–9. This process continues until we have x directly to the left of y_j for some j. But then moving y_j to the k^{th} stack above x would be a further right move than moving x. This is a contradiction to the assumption that moving x was the next move by the right-greedy algorithm.

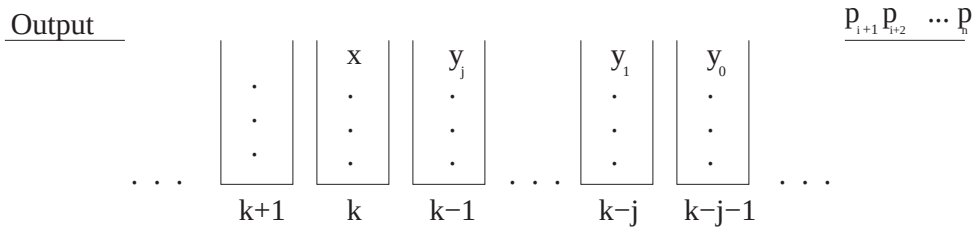

Figure 2–9: Stacks using the right-greedy algorithm immediately before x is to move into the $(k+1)^{st}$ stack.

Thus there is no entry that moves further left by the right-greedy algorithm than by the left-greedy algorithm at the $(i+1)^{st}$ critical moment. Therefore the Lemma is proved.

◇

Now we are ready to prove the main result of this chapter.

Theorem 2.5 *Any permutation sorted by the right-greedy algorithm on t stacks in series is also sorted by the left-greedy algorithm on t stacks in series.*

Proof: This argument follows that of the previous Lemma in the case of failure at the i^{th} critical moment. Let p be the permutation we are trying to sort. If the left-greedy algorithm fails, there must exist an entry of p that can never enter the stacks. Let p_i be the first entry of the permutation p that could not enter the stacks. Then the moment when no entries can move any further by the left-greedy algorithm, which is also the i^{th} critical moment, there exists a $\gamma < p_i$ in the rightmost stack stopping p_i from entering the stacks. However, if the right-greedy algorithm got even this far without being stuck, then γ must be in the rightmost stack by this algorithm as well by the previous Lemma. Therefore the right-greedy algorithm also fails.

◇

To see that the left-greedy algorithm sorts more permutations than the right-greedy algorithm when $t > 1$, we simply need an example of a permutation that is not sortable by t stacks in series using the right-greedy algorithm, but is sortable on t stacks using the left-greedy algorithm.

Example 2.6 For $t > 1$, the permutation $(t+1)t(t-1)...32(t+2)1$ is not sortable by t stacks in series when using the right-greedy algorithm, but is sortable by t stacks in series by the left-greedy algorithm. In fact, the left-greedy algorithm only needs two stacks in series to sort this permutation regardless of what t is.

In particular, when we have two stacks, this permutation is 3241. Example 2.1 demonstrated that using the right-greedy algorithm to sort 3241 by two stacks in series did not work in Figure 2–1. In Example 2.2 it was shown that the left-greedy algorithm could be used to successfully sort 3241 by two stacks in series in Figure 2–2.

Hence we have proved the following proposition.

Proposition 2.7 *The left-greedy algorithm is strictly better than the right-greedy algorithm for sorting permutations by t stacks in series when $t \geq 2$. By this we mean that the set of all permutations (of all lengths) sorted by t stacks in series when the right-greedy algorithm is applied is strictly contained in the set of all permutations sorted by t stacks in series when the left-greedy algorithm is applied.*

2.3 Obtaining More Permutations which are Sortable by t Stacks in Series

In the following lemma we show one way to obtain permutations of length n that are sortable by t stacks in series when the left-greedy algorithm is applied from permutations of length $n-1$ which can be sorted using the left-greedy algorithm on $t-1$ stacks in series. It will also be shown that if a permutation of length $n-1$ is sortable by $t-1$ stacks in series, then a permutation of length n that is sortable by t stacks in series can be constructed from it the same way.

However, this will not hold true when we restrict ourselves to using the right-greedy algorithm.

Lemma 2.8 *Let $p = p_1p_2...p_{n-1}$ be sortable by the left-greedy algorithm on $(t-1)$ stacks in series. Inserting n anywhere in the permutation will result in a permutation p' that is sortable by the left-greedy algorithm by t stacks in series.*

Example 2.9 The permutation 4231 is sortable by two stacks in series when using the left-greedy algorithm. Inserting 5 between 3 and 1 gives us 42351 which is sortable by three stacks in series. In Figure 2–10 we show 4231 and 42351 being sorted by three stacks in series. Here 4231 is shown being sorted by three stacks in series instead of two in order to demonstrate the process described in the proof of Lemma 2.8.

Proof: Since p is sortable on only $t-1$ stacks in series by the left-greedy algorithm, we know that if we try to sort our augmented permutation p' with the left-greedy algorithm on t stacks, as each entry in p' is about to enter the stacks until (possibly) after n enters, the rightmost stack will be empty. This is because the left-greedy algorithm would always move the entries in the stacks left before allowing a new entry to enter. Hence if we could not get one of the original entries of p to move out this first stack (before n has the possibility of interfering), p would not be sortable by the left-greedy algorithm on $t-1$ stacks. Because of this, it is clear that all entries up to and including n of p' will enter the stacks by the left-greedy algorithm.

Claim: If we ignore the moves made by n, the left-greedy algorithm on p' proceeds exactly the same way as it does on p when we try to sort both by t stacks.

An example of this is claim being true has already been shown in Figure 2–10.

Proof of Claim: This is evident until the entry n enters the stacks when sorting p'. If there is some discrepancy, let moving x from the k^{th} stack to the $(k+1)^{st}$ stack be the first move made on p' that does not match that on p. Note that $x \neq n$. Say

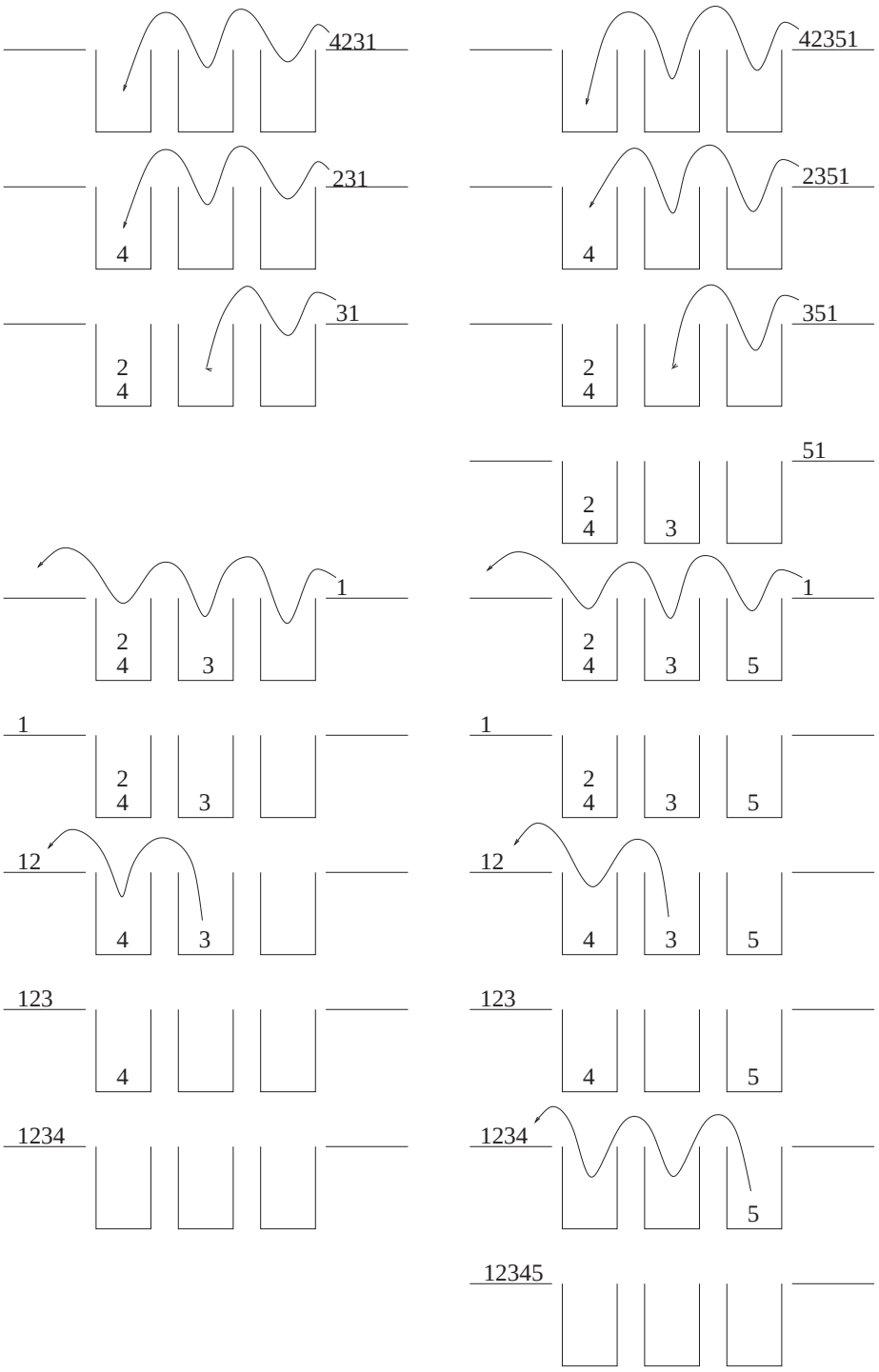

Figure 2–10: 4231 and 42351 being sorted by three stacks in series using the left-greedy algorithm.

moving y from the l^{th} stack to the $(l+1)^{st}$ stack is the corresponding move on p. Because the previous moves all corresponded before this point, each entry $p_i \neq n$ is in the same position for both permutations until we make this move.

Note that neither x nor y can be leaving the stacks for the output at this time. This is because both permutations have the same output and if one permutation has an entry other than n at the top of the leftmost stack, the other has the same entry there. Thus if the entry should be the next out, it is the leftmost move for both.

Case 1: $k > l$

In this case, moving x from the k^{th} stack to the $(k+1)^{st}$ stack is the further left move. But then x should have been able to move when the left-greedy algorithm was applied to p as well since there can be no additional entries in the $(k+1)^{st}$ stack for p that are not there for p'.

Case 2: $l > k$

The same argument as in Case 1 holds because the only extra entry that could be in the $(l+1)^{st}$ stack for p' is n and since all the other entries are smaller, n cannot prevent another entry from moving left.

This completes the proof of the claim. Since we know n can enter the t stacks for the permutation p' using the left-greedy algorithm and all entries $1, 2, 3, ..., (n-1)$ can enter the t stacks for the permutation p by the left-greedy algorithm, by the claim we know all the entries of p' can enter the t stacks when using the left-greedy algorithm. Hence p' is sortable by t stacks in series using the left-greedy algorithm.

$\diamond$

It is even easier to show the result is true if we allow any algorithm to be used to sort the permutation.

Lemma 2.10 *Let $p = p_1 p_2 ... p_{n-1}$ be sortable by some algorithm on $(t-1)$ stacks in series. Inserting n anywhere in the permutation will result in a permutation p' that is sortable by t stacks in series.*

Proof: Since we only needed $t-1$ stacks in series to sort p by some algorithm *, we may consider an algorithm on p' that proceeds exactly as * does on the entries originally in p except that when the entries first come in, they move one stack further left than they originally stopped. That is, for the entries $p_i \neq n$, the first stack is essentially skipped. This leaves the first stack open for n to enter. We leave n in that first stack until all the other entries have left the stacks and then finally we move n through the remaining $(t-1)$ stacks and into the output, sorting p'. This type of movement is shown in our earlier example in Figure 2–10. (All other entries that enter the stacks after n may easily move past n since they are smaller than n.)

◇

As stated before, this method of constructing permutations that are sortable by t stacks in series will not work if we only want to allow the right-greedy algorithm to sort the permutation.

Example 2.11 Consider the permutation 321, which is sortable by one stack using the right-greedy algorithm. Putting 4 into the third position results in the permutation 3241 which is not sortable by the right-greedy algorithm on two stacks in series.

We can also start with permutations that are sortable by t stacks in series to get longer permutations that are also sortable by t stacks in series. In this first proposition, we look only at the permutations sortable by the left-greedy algorithm.

Proposition 2.12 *Let p be a permutation of length $n-1$ that is sortable by t stacks in series using the left-greedy algorithm. We may insert n into any one of*

the first t positions or into the last position to get a new permutation p' that is also sortable by t stacks in series using the left-greedy algorithm.

Example 2.13 The permutation 2341 can be sorted by three stacks in series when using the left-greedy algorithm. Inserting 5 between 3 and 4 we get the permutation 23541 which is also sortable by three stacks in series when the left-greedy algorithm is applied. We show both permutations being sorted this way in Figure 2–11.

Proof: This result is clear for when n is put in the last position. This is shown in Figure 2–11. Suppose n was inserted into one of the first t positions. It is clear that n may enter the stacks at its turn since there are at most $t-1$ entries ahead of n and there are t stacks. The remainder of the proof follows exactly as the proof for Lemma 3.1.

◇

The previous proposition allows us to give the following lower bound on the number of permutations of length n which are sortable by t stacks in series.

Corollary 2.14 *Let $n \geq t$. Then the number of permutations of length n that are sortable by t stacks in series (using the left-greedy algorithm) is at least $t!(t+1)^{n-t}$.*

Proof: Every permutation of length $t+1$ is sortable by t stacks in series using the left-greedy algorithm. We may insert $t+2$ into any of the first t positions or into the last position to get a permutation of length $t+2$ that can be sorted using the left-greedy algorithm on t stacks in series. Repeat with $t+3, t+4, ..., n$. This gives us $(t+1)!(t+1)^{n-t-1} = t!(t+1)^{n-t}$.

◇

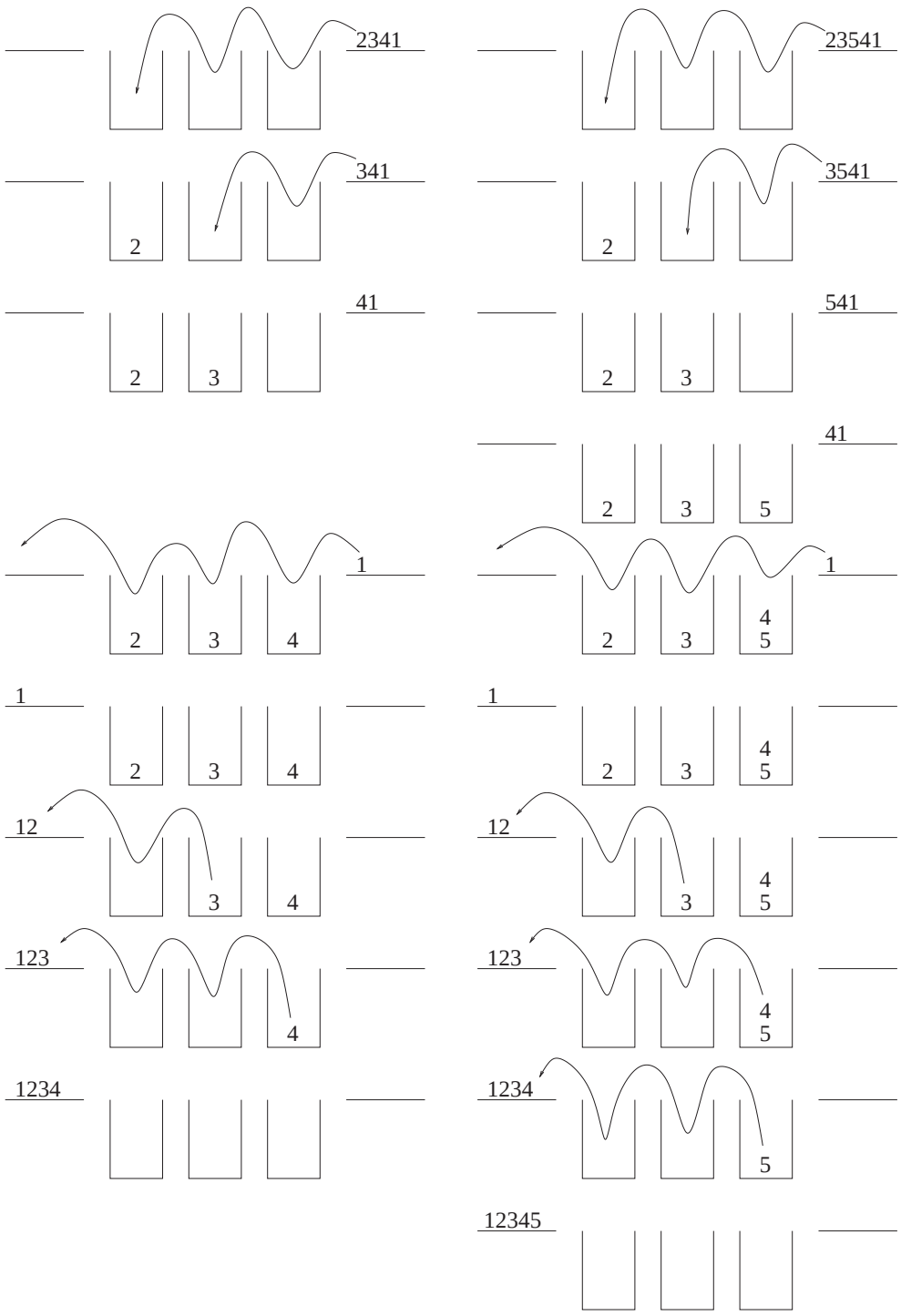

Figure 2–11: 2341 and 23541 being sorted by three stacks in series using the left-greedy algorithm.

The previous proposition will also hold for sorting permutations on t stacks in series using any algorithm that works. We first need the following lemma.

Lemma 2.15 *Suppose $p = p_1, p_2, ..., p_n$ is a permutation that can be sorted by t stacks in series by some algorithm. Then there is an algorithm that never needs to leave an empty stack to the left of a nonempty stack at the i^{th} critical moment for any given $i = 1, 2, ..., n$.*

Proof: Suppose not. Then given an algorithm with a minimal number of such stacks left empty at a minimal number of critical moments that does work, consider the first critical moment an offending stack must be left empty. Let p_j be an entry in the first nonempty stack to the right of an empty stack. Move p_j one place to the left. Notice that any algorithm that works can still proceed as before, since p_j does not block any entry that would have been able to move to the left of it had it not been moved. Hence the algorithm still works with one less empty stack at a critical moment. This is a contradiction.

◇

Proposition 2.16 *Let p be a permutation of length $n-1$ that is sortable by t stacks in series. We may insert n into any one of the first t positions or into the last position to get a new permutation p' that is also sortable by t stacks.*

Proof: This is clearly true when we insert n into the last position. Suppose that n was inserted into one of the first t positions. Since we do not need to leave empty stacks, it is clear that we may follow an algorithm that works for p and n can enter the stacks as there are at most $t-1$ entries ahead of it. After that we may leave n in the first stack until we are done carrying out the remainder of the algorithm since n will not interfere with any other entries moving into the stacks. Finally, move n through the remaining stacks and into the output to finish sorting p'.

◇

Consider instances when we insert n into one of the first t positions of a permutation of length $n-1$ that is sortable by the right-greedy algorithm on t stacks in series. In this case, the result is not always a permutation that is still sortable by the right-greedy algorithm on t stacks in series.

Example 2.17 The permutation 6372451 is sortable by the right-greedy algorithm by three stacks in series, but 68372451 is not.

However, note that inserting an n at the end of such a permutation is easily seen to produce another permutation that is sortable by t stacks in series using the right-greedy algorithm.

2.4 More Information about the Left-greedy Algorithm for $t > 2$

Although the left-greedy algorithm sorts any permutation that can be sorted by t stacks in series for $t = 1$ and $t = 2$, the left-greedy algorithm is not optimal when $t > 2$.

Example 2.18 While it can be seen that the permutation $254167...(t+3)(t+4)3$ is sortable by t stacks in series when $t > 2$, the left-greedy algorithm fails when $(t+4)$ would need to enter.

We show this in the case when $t = 3$ and we have the permutation 2541673. In Figure 2–12 we show how the left-greedy algorithm fails to sort this permutation by three stacks in series. In Figures 2–13 and 2–14 we show how this permutation can be sorted by three stacks in series when one is careful with how the entries move through the stacks. Notice that being patient with moving the 4 left is what makes the difference.

Also, the permutations sorted by the left-greedy algorithm are not a closed class when $t > 2$. (Since it is known [11] that the permutations sorted by t stacks in series is a closed class, this is actually enough to show that the left-greedy algorithm is not optimal for $t > 2$.)

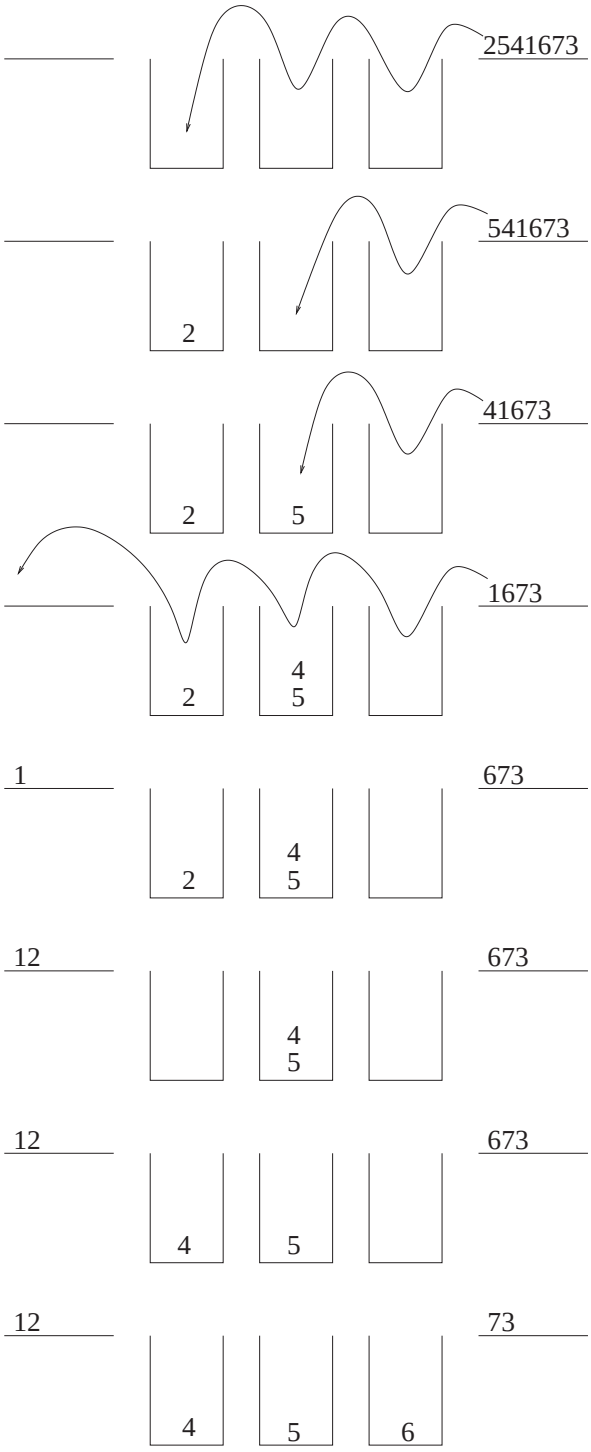

Figure 2–12: 2541673 failing to be sorted by three stacks in series using the left-greedy algorithm.

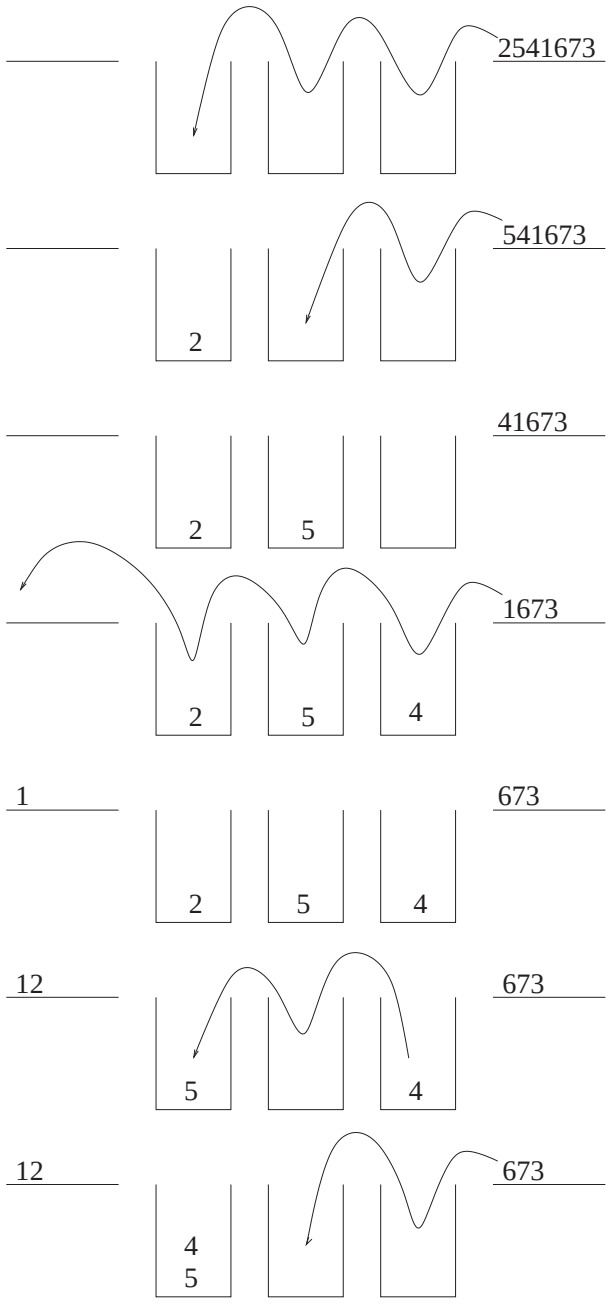

Figure 2–13: Part I of 2541673 being sorted by three stacks in series.

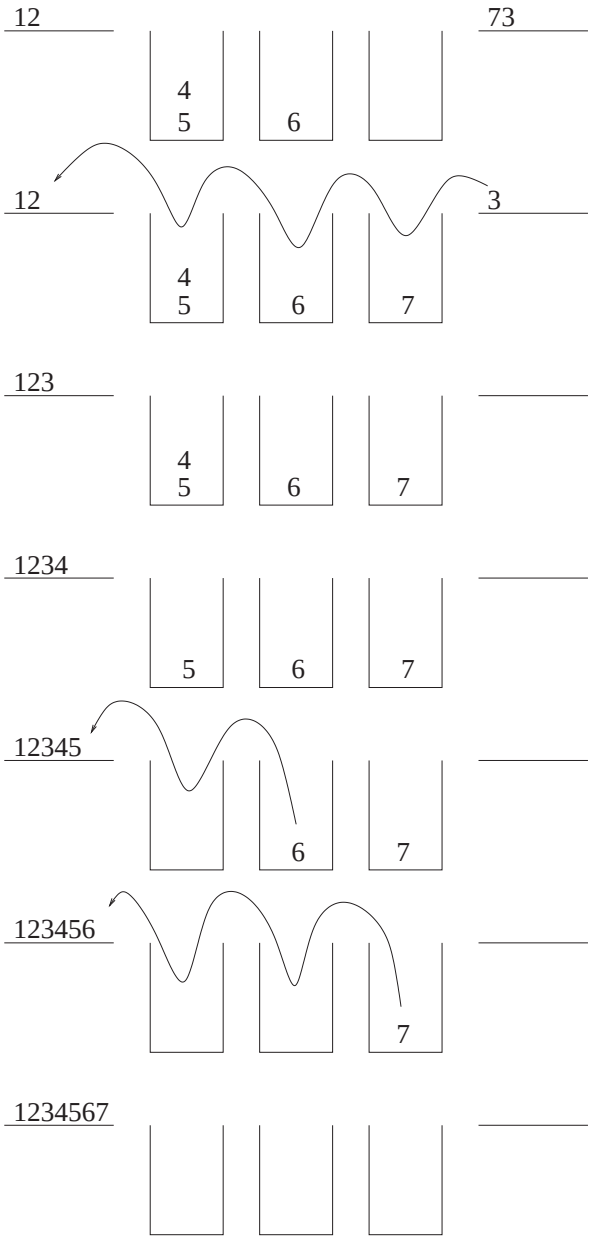

Figure 2–14: Part II of 2541673 being sorted by three stacks in series.

Example 2.19 Let $t > 2$. While the permutation $254167...(t+4)3$ is not sortable by t stacks in series using the left-greedy algorithm, the permutation $2635178...(t+5)4$ which contains the pattern $254167...(t+4)3$ is sortable using the left-greedy algorithm by t stacks in series.

We had already shown that in the case when $t = 3$, the permutation 2541673 is not sortable by three stacks in series when the left-greedy algorithm is applied. However, the permutation 26351784 is sortable by three stacks in series using the left-greedy algorithm as shown in Figures 2–15 and 2–16.

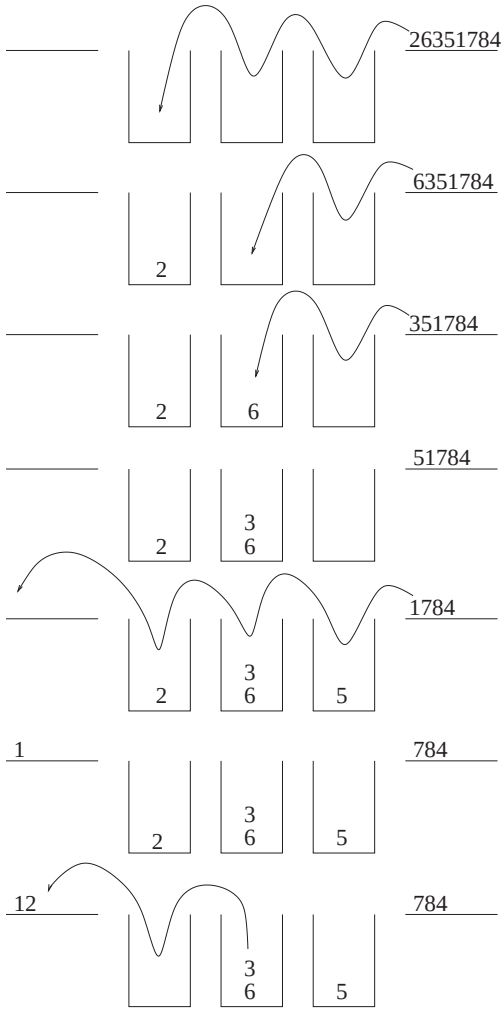

Figure 2–15: Part I of 26351784 being sorted by three stacks in series using the left-greedy algorithm.

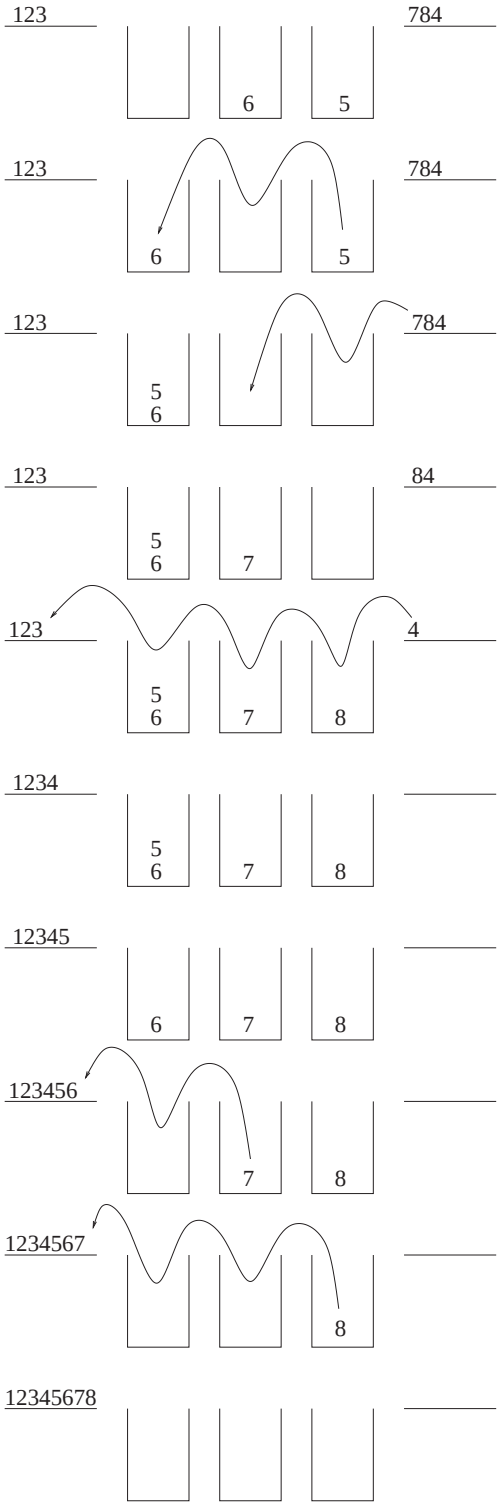

Figure 2–16: Part II of 26351784 being sorted by three stacks in series using the left-greedy algorithm.

CHAPTER 3
STACKS IN A CIRCULAR SERIES

Suppose instead of simply letting the entries of a permutation enter the stacks from left to right, we now allow entries from the last stack to continue to the output or return to the first stack. We will still keep both the monotonicity and "first in, last out" restrictions on all of the stacks. We will refer to this as *sorting by stacks in a circular series*. See Figure 3–1.

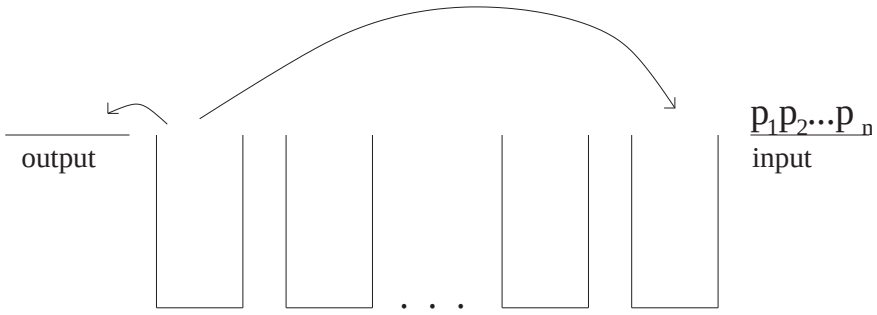

Figure 3–1: Stacks in a circular series

How much easier does having this extra allowance make it to sort permutations? This question is answered in the following two propositions.

Proposition 3.1 *When we have just two stacks, using the circular series will result in sorting exactly the same permutations that can be sorted by just having two stacks in series.*

Proof: Since we are not required to have any entries of a permutation circle back to the first stack, it is clear that any permutation that can be sorted by two stacks in series can be sorted by two stacks in a circular series.

To see that there is no permutation that can be sorted by two stacks in a circular series that cannot be sorted by just two stacks in series, we consider a permutation p that is not sortable by two stacks in series.

We may assume that the left-greedy algorithm was applied as it is optimal in the case when we have just two stacks. Let p_i be the first entry of p that cannot legally enter the stacks. There must be an entry c in the first stack that is smaller than p_i which keeps p_i from entering the stacks. Likewise, there is an entry b in the second stack that is smaller than c which keeps c from moving to the next stack. Finally there must be an entry a smaller than b that has not yet entered the stacks and thus prevents b from going to the output. See Figure 3–2.

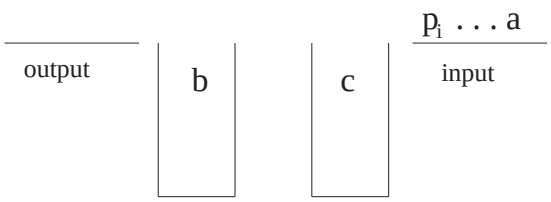

Figure 3–2: Stacks at the moment of failure.

In order for this permutation to be sorted by two stacks in a circular series, there needs to be a way for p_i to enter the stacks. For this to happen, c cannot be in the first stack. Thus at some point we need to move c left. Since the left-greedy algorithm was applied to try to sort the permutation by two stacks in series, we know that at every stage there must have been a smaller entry to the left of c or above c. Our extra allowance may allow us to move the smaller entry from the left of c back to the first stack, but then the entry is above c and still must be moved to the left of c before c can move. Hence this does not help move c left in any way. If the problem was that the smaller entry was above c, the left-greedy algorithm would have moved it left when it was possible and this move does not require our circular allowance. Thus there is no way to move c left and thus we cannot bring p_i in the stacks using two stacks in a circular series.

Hence no permutation can be sorted by two stacks in a circular series that could not have been sorted by two stacks in series.

◇

Since adding another possible move did not help at all when sorting by two stacks in series, it may be surprising to learn that once there are more than two stacks in series the addition of the circular move makes an immense difference. In fact, every permutation can be sorted by three or more stacks in a circular series. The difference can be seen in part by observing that the class of permutations that are sortable by t stacks in series is closed and therefore categorized by pattern avoidance. Thus if there is even a single pattern that must be avoided for a permutation to be sortable, by the previously mentioned Marcus-Tardos Theorem we know the number of such permutations of length n is bounded by c^n where c is a constant dependent only on the basis of patterns to be avoided.

Proposition 3.2 *For any integer $t > 2$, any permutation of any given length can be sorted by t stacks in a circular series.*

This proposition will follow from the following lemma.

Lemma 3.3 *Given a permutation p and three stacks in a circular series, we can always maneuver the entries that are currently in the stacks to all be in any one stack.*

Proof: This proof is done using induction on the number of entries in the stacks. It is clear when there is only one entry in the stacks. Suppose it is true when we have no more than k entries in the stacks and we will prove it when there are $k+1$ entries.

Let p_{k+1} be the $(k+1)^{st}$ entry and suppose p_{k+1} is the i^{th} largest of these $k+1$ entries.

Suppose we are trying to get the $k+1$ entries all in the first stack. Before p_{k+1} enters the stacks, put the k entries already there so that they are all in the first stack using the induction hypothesis.

Then move the $i-1$ smallest entries to the second stack. This can also be done using the induction hypothesis since there are less than $k+1$ of these entries

and they can move as if there are no other entries in the stacks since each is smaller than any of the remaining entries.

With the smaller elements out of the way, p_{k+1} can freely enter the first stack. Once again the $i-1$ smallest entries can move as though there are no other entries in the stack and so by the induction hypothesis, we can move them all back to the first stack.

To get the $k+1$ entries in the second stack, the process is the same as before. Before letting p_{k+1} enter the stacks, move the k entries already there to the second stack. Then move the $i-1$ smallest entries to the third stack. Now p_{k+1} can enter the first stack and move to the second stack. Finally the $i-1$ smallest entries can move back to the second stack.

If we are trying to get the $k+1$ entries into the third stack, we need a slight variation on this method. Once again, before letting p_{k+1} enter the stacks, put the k original entries in the third stack. Next, bring p_{k+1} into the first stack and over to the second stack. Now move the $i-1$ smallest entries to the first stack. This allows to move p_{k+1} to the third stack. Lastly bring the $i-1$ smallest entries back to the third stack.

$\diamond$

This lemma gives us all we need to now prove the earlier proposition.

Proof: Since we can put all the entries in one stack, this means that we can put all n entries of p into the stacks and get them all in the last stack. Since the stacks are in descending order, we need merely to empty the third stack into the output to get the identity permutation.

Also note that having t stacks when $t > 3$ does not hurt the process as we can always say that no entry stays in the stacks $3, 4, ..., t-1$. That is, if an entry enters

any of these stacks it keeps moving left until it ends up in stack t. Then we are essentially using only three stacks.

◇

Example 3.4 The permutation 23451 is not sortable by three stacks in series, but we can sort it when we put the stacks in a circular series. Figures 3–3 and 3–4 demonstrate one way this can be done.

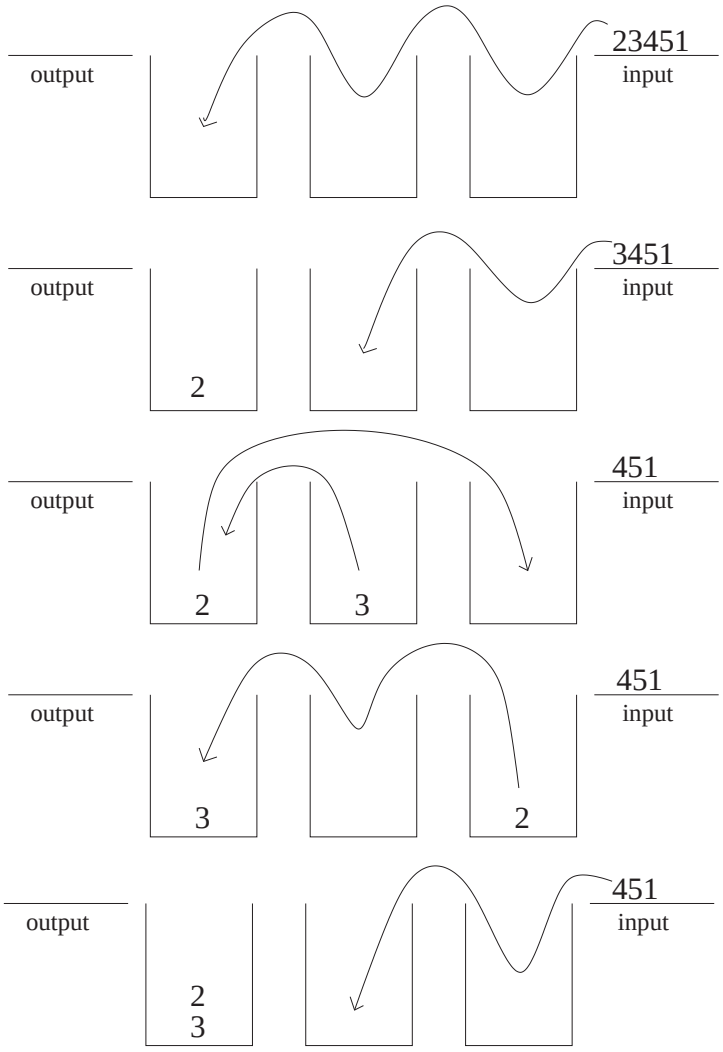

Figure 3–3: Part I of sorting 23451 by three stacks in a circular series.

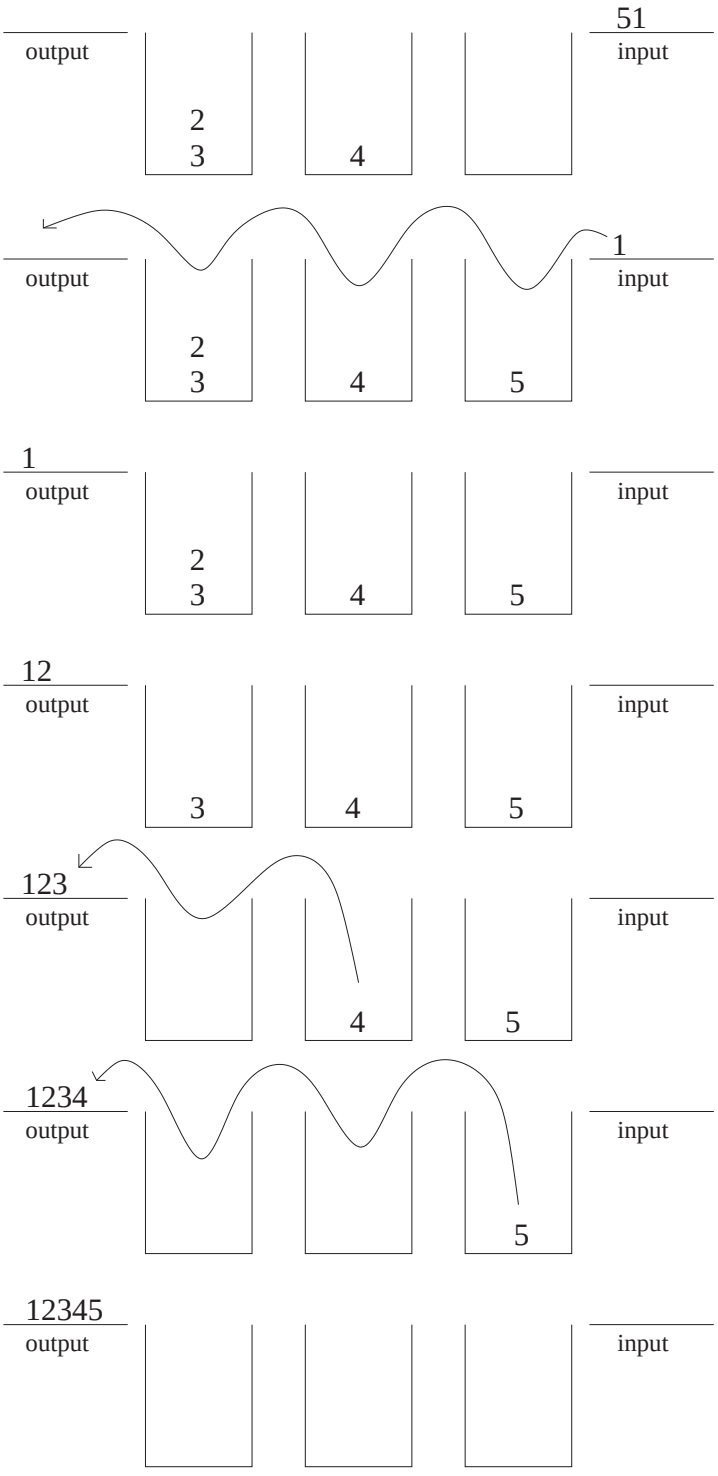

Figure 3–4: Part II of sorting 23451 by three stacks in a circular series.

CHAPTER 4
PARITY AND SORTABLE PERMUTATIONS OF A GIVEN LENGTH

Counting the permutations of length n that are sortable by two or more stacks is difficult. One way to get more information in that direction is to consider parity.

4.1 Parity of the Number of Permutations of Length n which are t-Stack Sortable for $t = 1$ or 2

In the case of permutations of length n that are sortable by just one stack, there are many proofs of exactly when there are an odd number of such permutations. This is due to the fact that these permutations are counted by the Catalan numbers.

Theorem 4.1 *The n^{th} Catalan number is odd exactly when $n = 2^r - 1$ for some natural number r.*

One interesting combinatorial proof by Eğecioğlu [9] uses a collection of lattice paths which are also counted by the Catalan numbers. We present his proof here.

Proof: One type of combinatorial object that the n^{th} Catalan number counts is a set of lattice paths D_n. These lattice paths start at $(0, 0)$ and end at $(2n, 0)$ by using only the steps $(1, 1)$ and $(1, -1)$ and never leave the first quadrant.

Define an involution $\alpha : D_n \rightarrow D_n$ by reflecting the paths across the line $x = n$. Then the only paths of D_n that have not been paired up are those which are symmetric with respect to the line $x = n$.

To complete the proof, we will need to show that there are an odd number of these symmetric paths if and only if $n = 2^r - 1$ for some natural number r.

Note that every lattice path in D_n takes exactly $2n$ steps. If w is a symmetric path of D_n, then w can be broken down in such a way that $w = w_1 u \bar{u} w_2$ where w_1

and w_2 are paths that take $n-1$ steps each. Then u would be either the step $(1,1)$ or the step $(1,-1)$, and $\bar{u}$ would be the other type of step.

We use this decomposition to define another involution β. If w_1 (and thus w_2) is not a path of $D_{\frac{n-1}{2}}$, then we know that w_1 stops above the x-axis. Therefore in this case it is safe to the swap steps u and $\bar{u}$ to get another symmetric path of D_n. We define β in the following way.

$$\beta(w) = \begin{cases} w_1\bar{u}uw_2 & \text{if } w_1 \notin D_{\frac{n-1}{2}} \\ w & \text{else.} \end{cases}$$

Now if n is even, then $D_{\frac{n-1}{2}}$ does not exist and so the involution β completes the pairing. Hence $|D_n|$ is even when n is even.

We use induction to prove the remainder of the theorem. The theorem is clearly true when $n = 1$. Suppose it is true for any natural number less than n. When proving this is true for n, we may assume n is odd.

The only unpaired paths of D_n are the symmetric paths $w = w_1u\bar{u}w_2$ where $w_1 \in D_{\frac{n-1}{2}}$. Notice that w is completely determined by w_1 since u must be the step $(1,1)$ so that the path can remain in the first quadrant and w_2 simply takes the opposite steps of w_1 in the reverse order.

Thus $f : D_n \to D_{\frac{n-1}{2}}$ where $f(w) = w_1$ is a bijection. Hence $|D_n| \equiv |D_{\frac{n-1}{2}}| \pmod 2$.

If $|D_n|$ is odd, then we have $|D_{\frac{n-1}{2}}|$ is also odd and thus $\frac{n-1}{2} = 2^r - 1$ for some natural number r. Hence $n = 2^{r+1} - 1$.

Likewise, if $n = 2^r - 1$ for some natural number r, then $\frac{n-1}{2} = 2^{r-1} - 1$ and so $|D_{\frac{n-1}{2}}|$ is odd and thus $|D_n|$ is odd as well.

Since $C_n = |D_n|$ this completes the proof.

$\diamond$

The number of times that the number of 1-stack sortable permutations of a given length is odd is infinite. Interestingly, if we consider the function

$$\delta_{C_n} = \begin{cases} 0 & \text{if } C_n \text{ is even} \\ 1 & \text{if } C_n \text{ is odd} \end{cases}$$

we can see that the density of δ_{C_n} is zero.

In the case of 2-stack sortable permutations of length n, it has been determined combinatorially by Bóna [4] exactly which values of n have $W_2(n)$ odd. Starting with $n_0 = 1$, a number m can be reached by some combination of the following steps:

1. $n_{i+1} = 2n_i - 1$ or
2. $n_{i+1} = 4n_i + 1$

if and only if the value of $W_2(m)$ is odd.

In fact, Bóna [4] showed that exactly F_{k+1} of the values n less than 2^k are such that $W_2(n)$ is odd. Note that F_{k+1} denotes the $(k+1)^{st}$ Fibonacci number.

If we consider the function

$$\delta_{W_2(n)} = \begin{cases} 0 & \text{if } W_2(n) \text{ is even} \\ 1 & \text{if } W_2(n) \text{ is odd} \end{cases}$$

the density of $\delta_{W_2(n)}$ is again zero. This is so because the asymptotic growth of the Fibonacci numbers is roughly $(1.61)^n$.

Guibert has computed the number of 3-stack sortable permutations of length n for $n \in 1, 2, 3, ..., 9$. He found that the only two cases where this number is odd is when $n = 1$ or $n = 9$. The only cases where the number is odd for 1-stack sortable permutations occur when $n = 2n + 1$, and the only cases when the number is odd for 2-stack sortable permutations occur when $n = 4n + 1$. These facts suggest that perhaps the only cases when the number is odd for 3-stack sortable permutations occur when $n = 8n + 1$.

4.2 Parity of the Permutations of Length n which are Sortable by Two Stacks in Series

The parity of permutations of length n that are sortable by two stacks in series is not difficult to find from the aforementioned formula in Theorem 1.28 for the number of such permutations,

$$S_n(1342) = \frac{7n^2 - 3n - 2}{2} \cdot (-1)^{n-1} + 3\sum_{i=2}^{n} 2^{i+1} \cdot \frac{(2i-4)!}{i!(i-2)!} \cdot \binom{n-i+2}{2} \cdot (-1)^{n-1}.$$

Theorem 4.2 *The number of permutations of length n which are sortable by two stacks in series is odd exactly when n is congruent to zero or one modulo four.*

Proof: The first term is odd exactly when n is congruent to zero or one modulo four. Each term of the summation can be seen to be even due to the relatively high power of two. Hence the number of permutations of length n that are sortable by two stacks in series is odd if and only if n is congruent to zero or one modulo four.

◇

Here we show a semi-combinatorial proof of this theorem.

Proof: As was mentioned in Chapter 1, permutations which are sortable by two stacks in series are related to $\beta(0,1)$ trees. Recall that the permutations of length n which are sortable by two stacks in series are in bijection with plane forests of $\beta(0,1)$ trees on a total of n vertices. This was shown by Atkinson, Murphy, and Ruškuc [1]. What we will show is that there is an odd number of plane forests of $\beta(0,1)$ trees on n vertices exactly when n is congruent to zero or one modulo four.

The number of $\beta(0,1)$ trees on n vertices is given by

$$3 \cdot 2^{n-2} \frac{(2n-2)!}{(n-1)!(n+1)!}$$

as was proved by Cori, Jacquard, and Schaeffer [7].

This proof is only semi-combinatorial due to the fact that we use this formula to see that the number of $\beta(0,1)$ trees on n vertices is even exactly when $n > 3$. We define f to be an involution on $\beta(0,1)$ trees that takes one $\beta(0,1)$ tree on k vertices to another $\beta(0,1)$ tree on k vertices. Further we may assume that f has no fixed points when $k > 3$. In the cases when $k = 1$ and $k = 2$, there is just one $\beta(0,1)$ tree for each, so f will fix these two trees. Finally in the case when $k = 3$, we have three $\beta(0,1)$ trees as shown in Figure 4–1

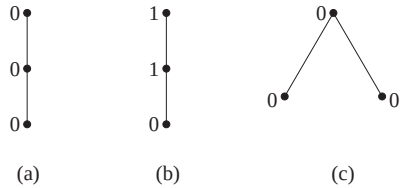

Figure 4–1: The three $\beta(0,1)$ trees on three vertices.

In this case we will have f take tree (a) to tree (b) and fix the trees of type (c).

Now we consider plane forests of $\beta(0,1)$ trees with a total of k vertices. We pair most of these forests with other such forests by the following steps.

First, pair forests by applying f to each tree. The remaining unpaired forests have some combination of only single vertex trees, trees with two vertices, or the three vertex tree (c).

The forests that remain are exactly those which contain at most one type of tree with three vertices, at most one type of tree with two vertices, and at most one type of tree with one vertex. What we are really looking at are ordered partitions of n into parts no larger than three.

Let a_n be the number of such partitions of n. It is clear to see that $a_0 = 1$, $a_1 = 1$, $a_2 = 2$, and $a_3 = 4$. By using a recursion on the number of these partitions, we will prove the following claim.

Claim: The number of ordered partitions of n into parts that are no larger than three is odd exactly when n is congruent to zero or one modulo four.

Proof of Claim: We will use induction on n. We have shown the claim to be true when $n \leq 3$. Suppose it is true for any whole number smaller than n. Then consider the ordered partitions of n into parts no larger than three.

When n is at least three, we know that the first part can be one, two, or three. Then the remainder is the same type of partition of $n-1$, $n-2$, or $n-3$ respectively.

Hence $a_n = a_{n-1} + a_{n-2} + a_{n-3}$ when $n \geq 3$. Note that this recursion produces the Tribonacci numbers when $a_0 = 0$, $a_1 = 0$, and $a_2 = 1$. Besides the two initial zeros, the sequence for the Tribonacci numbers is the same as the sequence for number of partitions of n into parts no larger than three.

If $n \equiv 0 \pmod 4$, then by the induction hypothesis, only a_{n-3} is odd while a_{n-1} and a_{n-2} are even, hence a_n is odd.

If $n \equiv 1 \pmod 4$, then again using the induction hypothesis, this time only a_{n-1} is odd while a_{n-2} and a_{n-3} are even and so a_n is odd.

If $n \equiv 2 \pmod 4$, then only a_{n-3} is even which means a_n is even as well.

Finally, if $n \equiv 3 \pmod 4$, now only a_{n-1} is even and thus a_n is also even.

Hence we have proved our claim which also proves our theorem since it has already been shown that there are an even number of the other $\beta(0,1)$ plane forests on n vertices regardless of the value of n.

$\diamond$

As in the case of 1-stack sortable permutations, there are infinitely many times that the number of permutations of a given length which are sortable by two stacks series is odd. Recall that the number of permutations of length n that are sortable by two stacks in series is the same as the number of permutations that avoid 1342.

In contrast to the previous cases, if we define

$$\delta_{S_n(1342)} = \begin{cases} 0 & \text{if } S_n(1342) \text{ is even} \\ 1 & \text{if } S_n(1342) \text{ is odd} \end{cases}$$

the density of $\delta_{S_n(1342)}$ not zero. In fact, since $\delta_{S_n(1342)}$ is odd exactly when $n \equiv 0$ or $1 \pmod 4$, the density of $\delta_{S_n(1342)}$ is exactly one half.

This also means that the number of permutations of length n that avoid 1342 is odd for exactly half the values of n. This is first instance where the number of permutations a given length that avoid a pattern is known to have an equal number of odd and even cases.

CHAPTER 5
PERMUTATION RECONSTRUCTION

5.1 Introduction

The idea of graph reconstruction began with a question of Ulam [21] in his problem book.

Conjecture 5.1 *(Ulam) Let A and B be sets with n elements each where $n \geq 3$. Also suppose that a metric ρ_A is given on A with the property that $\rho_A(x, y)$ is equal to either one or two for any two distinct elements x and y in A. A metric ρ_B with the same property is given on B. Now suppose that there exists a bijection f between the $(n-1)-$element subsets of A and the $(n-1)-$element subsets of B such that if $f(A_i) = B_i$ then A_i is isometric to B_i. Then A is isometric to B.*

If we then suppose that A and B are unlabeled simple graphs and define the metric $\rho_A(x, y)$ (and similarly $\rho_B(x, y)$) to be such that

$$\rho_A(x, y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ are adjacent vertices of } A \\ 2 & \text{if } x \text{ and } y \text{ are nonadjacent vertices of } A \ , \end{cases}$$

we get an equivalent graph theory conjecture.

Let A and B be any two unlabeled simple graphs with n vertices where $n > 3$. Let $A_1, A_2, ..., A_n$ be all the minors that result when in each possible way you remove a single vertex and all its incident edges from the graph A. Suppose there is a way to label the minors $B_1, B_2, ..., B_n$ that result when in all possible ways you remove a single vertex and all its incident edges from the graph B such that A_i is isomorphic to B_i for each $i \in \{1, 2, 3, ..., n\}$. The conjecture then states that this is enough to show that A is isomorphic to B.

This conjecture can be stated alternatively to say that any unlabeled graph with more than three vertices can be reconstructed (up to isomorphism) from the set of minors that result from the removal of a single vertex and all its incident edges. One variation on this problem is trying reconstruct a graph by looking at the minors that result when a single edge is removed instead of a vertex. Another is to fix a number k and see if graphs can be reconstructed if one considers the removal of each set of k vertices and their incident edges to form the minors.

Since Ulam made his conjecture, many partial results have been found in graph theory that are consistent with this. However, the complete problem remains unsolved.

We now consider this problem applied to permutations instead of graphs.

Definition 5.2 *Let p be a permutation of length n. In all possible ways, delete k entries of p. Then renumber (retaining the order) the obtained $\binom{n}{k}$ strings as permutations so that their entries are from 1 through $n-k$. We will refer to each of these renumbered strings as* $(n-k)$-minors of p. *Call the multiset of these permutations the* $(n-k)$-minor multiset of p *and denote it $M_k(p)$.*

Example 5.3 Consider the permutation $p = 142536$. If we remove two entries of p in all possible ways, we get the multiset:

$\{1425, 1423, 1453, 1253, 4253, 1426, 1456, 1256, 4256, 1436, 1236, 4236, 1536, 4536,$
$2536\}$

After renumbering and reordering, we have

$M_2(142536) = \{1234^3, 1243, 1324^5, 1342, 1423, 2143, 2314, 3124, 3142\}$

Let N_k be the smallest number such that if given $M_k(p)$, where p is a permutation of length $n \geq N_k$, we can figure out what p must have been. That is, each permutation of length $n \geq N_k$ gives a unique multiset of $(n-k)$-minors, but there is a pair of permutations p and q each of length $N_k - 1$ where $M_k(p) = M_k(q)$.

Note that although graph reconstruction has been seriously studied, it could be that the results for permutation reconstruction are completely different.

One reason to think that this may be the case can be seen by looking at posets ordered by minors of both graphs and permutations.

Definition 5.4 *A* minor of a graph *can be obtained by either contracting an edge (which makes the two vertices that were joined by the edge into a single vertex) or by deleting an edge.*

Note that we can get the minors described in the graph reconstruction problem by contracting an edge incident to the vertex to be deleted and some other vertex v and then deleting edges that make v adjacent to vertices that it was not adjacent to before the contraction.

Consider a poset of unlabeled, simple graphs P_G obtained by the order $\leq_G$ being such that $G_1 \leq_G G_2$ if and only if G_1 is a minor of G_2. Likewise let the poset P_P of permutations be obtained by applying the order $\leq_P$ where $p \leq_P q$ if and only if the permutation p is a minor of the permutation q.

Definition 5.5 *An* antichain *of either P_G or P_P is a collection of objects where no object is a minor of another object.*

It has been proved by Robertson and Seymour [14] that P_G does not have an infinite antichain. Since there is no infinite antichain, the graphs of P_G are said to have a *well-quasi ordering*.

However, P_P does not have a well-quasi ordering as there are examples of infinite antichains of the poset P_P. One such example is the set of minimal permutations which are not sortable by two stacks in series given in Chapter 1. Another example can be found in Chapter 16 of A Walk Through Combinatorics by Bóna [6].

This difference in possibilities in the minors could make solving the corresponding reconstruction problems with permutations more promising than it

has been with graph theory. In fact, in the next sections, we present proofs that $N_1 = 5$, $N_2 = 6$, as well as some partial results for larger values of k.

5.2 The Case when $k = 1$

Proposition 5.6 *Let p be a permutation of length $n \geq 5$. Then $M_1(p)$ is unique to p. That is, $N_1 \leq 5$.*

Before we prove this, we need a definition.

Definition 5.7 *A permutation $p = p_1p_2...p_n$ is said to contain a* factor *$q = q_1q_2...q_k$ if p contains q as a contiguous subword.*

The following proof of Proposition 5.6 is due to Bruce Sagan [15].

Proof: Let p be a permutation of length $n \geq 5$. We will first show that we can determine which position in p the 1 is in from $M_1(p)$.

If 1 is in the i^{th} position in p, then consider the position of 1 in the $(n-1)$-minors of p. We have

$$\begin{aligned} &n - i \text{ times, 1 is in the } i^{th} \text{ position and} \\ &i - 1 \text{ times, 1 is in the } (i-1)^{st} \text{ position.} \end{aligned} \tag{5.1}$$

Also, one time the position of 1 in an $(n-1)$-minor is determined by 2 when the original 1 of p is the entry deleted.

Thus we have three possibilities.

1. The 1 is in position i where p does not contain 12 or 21 as a factor. This happens if and only if there is a position $j \neq i, i-1$ such that one of the $(n-1)$-minors of p has 1 in position j.
2. The 1 is in position i where p contains the factor 12. This case happens if and only if the positions i and $i-1$ are the only places where any of the $(n-1)$-minors have a 1 and at least $n-2$ of them, which is more than half of the $(n-1)$-minors since $n \geq 5$, also contain the factor 12. (The $n-2$ times are when neither the original 1 or 2 was eliminated.)

3. The 1 is in position i where p contains the factor 21. This case happens if and only if the positions i and $i-1$ are the only places where any of the $(n-1)$-minors have a 1 and at least $n-2$ of them also contain the factor 21.

Now we can determine which position i is from these possibilities and Equation 5.1 and thus we know where the 1 is in p.

To complete the proof, we need to find which minor p' was produced by eliminating the original 1 of p. Then we can simply insert the 1 into the i^{th} position of p' and increase all the other entries by one to get the permutation p.

We will consider the three cases listed above.

1. In the case when p did not contain 12 or 21 as a factor, p' is the $(n-1)$-minor where 1 is not in position i or $i-1$.
2. In the case when p contains the factor 12, we can see that p' is the permutation with shortest factor of the form $123...m$. This is true since eliminating an entry from the factor will decrease the length of it by one. Note that if the original p contains $123...(m+1)$, then eliminating any of the entries $1, 2, ..., m+1$ will result in such a minor, but since the entries are all adjacent, these minors will all be the same, so it will not matter which one of these minors we pick.
3. Finally when p contains the factor 21, we can see that p' is the permutation with shortest factor of the form $m(m-1)(m-2)...21$ by the same reasoning as in case 2.

Hence the proposition is proved.

$\diamond$

Theorem 5.8 $N_1 = 5$.

Proof: As a result of the previous proposition, what remains to be shown is that there is a pair of permutations p and q each of length four such that

$M_1(p) = M_1(q)$.

If we let $p = 2413$ and $q = 3142$ we can check that

$M_1(p) = \{132, 213, 231, 312\} = M_1(q)$.

$\diamond$

5.3 The Case when $k = 2$

Proposition 5.9 *Let p be a permutation of length $n \geq 6$. Then $M_2(p)$ is unique to p.*

We used a computer program to show that $M_2(p)$ is unique to p if the length of p is six or seven.

Proof: We may now consider the case when the length of p is at least eight. We will use induction on the entries of p.

First consider the position of 2 relative to the position of 1 in p. In the $\binom{n-2}{2}$ cases when neither the original 1 or 2 of p is eliminated, we get that in the resulting $(n-2)$-minors, the 1 is on the same side of 2 as in p.

Since $\binom{n-2}{2} > \frac{\binom{n}{2}}{2}$ when $n > 7$, we can determine which side of 1 the 2 is on in p from the $(n-2)$-minors of p.

Now set aside $\binom{n-2}{2}$ of the $(n-2)$-minors where 1 is on the same side of 2 as it is in p. The remaining $(n-2)$-minors determine where 3 is relative to 1 and 2.

Notice when exactly one of 1 or 2 is eliminated and 3 is not, the 3 acts as a 2 in the resulting $(n-2)$-minors. The final cases occur when two of 1, 2 and 3 are eliminated and 3 acts as the 1 or is not included at all.

If in $0 \leq k \leq 3$ of these $(n-2)$-minors, 1 is to the left of 2, then 3 must be to the left of both 1 and 2.

If in $n-3 \leq k \leq n$ of these $(n-2)$-minors, 1 is to the left of 2, then 3 must be between 1 and 2.

If in $2n-6 \leq k \leq 2n-3$ (all but at most three) of these $(n-2)$-minors, 1 is to the left of 2, then 3 must be to the right of both 1 and 2.

Since $n \geq 8$, all of these cases are distinct.

We can now determine where 1, 2, and 3 are relative to one another.

Suppose we can determine where $1, 2, 3, ..., i$ are relative to one another. Then we will show how to determine where $i+1$ is relative to $1, 2, 3, ..., i$.

We use another induction argument here. First we will show how to find where $i+1$ is relative to 1, 2, and 3.

Consider where 1 is relative to $i-1$ in the $(n-2)$-minors of p. These can all be found by where $1, 2, 3, ..., i$ are relative to each other except for the following cases:

$\binom{i-1}{2}$ times this is determined by where 1 is relative to $i+1$,

$i-2$ times this relative order is determined by where 2 is relative to $i+1$, and

one time this is determined by where 3 is relative to $i+1$.

Since we know where 1, 2, and 3 are relative to one another, this determines where $i+1$ is relative to 1, 2, and 3.

Now suppose we know where $i+1$ is relative to $1, 2, ..., k-1$. To find where $i+1$ is relative to k, consider where $k-2$ is relative to $i-1$ in the $(n-2)$-minors of p. Except in the case where these two entries come from k and $i+1$, we know where $k-2$ and $i-1$ should be relative to one another in the $(n-2)$-minors of p. Simply look at the remaining minors to determine where k is relative to $i+1$ in p.

Since we can determine where all the entries are relative to one another, we can find p from its $(n-2)$-minors.

$\diamond$

Theorem 5.10 $N_2 = 6$.

Proof: Given the previous proposition, all we need to do is show is that there is a pair of permutations p and q each of length five such that $M_2(p) = M_2(q)$.

If we let $p = 13524$ and $q = 14253$ we can check that

$M_2(p) = \{123^3, 132^4, 213, 231, 312\} = M_2(q)$.

$\diamond$

5.4 The Case when $k = 3$

In this section we will use similar techniques as shown in the last section to prove that $N_3 = 7, 10, 11, 12,$ or 13.

Proposition 5.11 *Let p be a permutation of length $n \geq 13$. Then $M_3(p)$ is unique to p.*

Note: We can again use a computer program to show that $M_3(p)$ is unique to p if the length of $p = 7$ or 8, but the numbers are too large for this program when we look at longer permutations. This in combination with the following proof and the later example show that $N_3 = 7, 10, 11, 12,$ or 13.

Proof: We may now consider the case when the length of p is at least thirteen. We will use induction on the entries of p.

First consider the position of 2 relative to the position of 1 in p. In the $\binom{n-2}{3}$ cases when neither the original 1 or 2 of p is eliminated, we get that in the resulting $(n-3)$-minors, the 1 is on the same side of 2 as in p.

Since $\binom{n-2}{3} > \frac{\binom{n}{3}}{2}$ when $n > 12$, we can determine which side of 1 the 2 is on in p from the $(n-3)$-minors of p.

Notice when exactly one of 1 or 2 is eliminated and 3 is not, the 3 acts as a 2 in the resulting $(n-2)$-minors. Note that there are $n-2$ times when both 1 and 2 are eliminated and $2\binom{n-3}{1}$ times when 3 is eliminated and exactly one of 1, 2 is eliminated. Hence there are $3n-8$ times when the 3 does not act as the 2 in the $(n-3)$-minors that result when at least one of 1, 2 is eliminated.

Now set aside $\binom{n-2}{3}$ of the $(n-3)$-minors where 1 is on the same side of 2 as it is in p. The remaining $(n-3)$-minors determine where 3 is relative to 1 and 2.

If in $0 \leq k \leq 3n-8$ of these $(n-3)$-minors, 1 is to the left of 2, then 3 must be to the left of both 1 and 2.

If in $\binom{n-3}{2} \leq k \leq \binom{n-3}{2} + 3n - 8$ of these $(n-3)$-minors, 1 is to the left of 2, then 3 must be between 1 and 2.

If in $2\binom{n-3}{2} \leq k \leq 2\binom{n-3}{2} + 3n - 8$ of these $(n-3)$-minors, 1 is to the left of 2, then 3 must be to the right of both 1 and 2.

Since $n \geq 13$, all of these cases are distinct.

We can now determine where 1, 2, and 3 are relative to one another.

Once again, from the total multiset of $(n-3)$-minors of p, set aside $\binom{n-3}{3} + 3\binom{n-3}{2}$ of these minors that have 1, 2, and 3 in the same order as p does. We do this since there are $\binom{n-3}{3} + 3\binom{n-3}{2}$ minors where at most one of 1, 2, and 3 is eliminated.

In the $(n-3)$-minors that result from at least two of 1, 2, and 3 being eliminated, the 4 acts as the 2 except when three of 1, 2, 3, and 4 are eliminated.

If in $0 \leq k \leq 4$ of these $(n-3)$-minors, 1 is to the left of 2, then 4 must be to the left of 1, 2, and 3.

If in $n-4 \leq k \leq n$ of these $(n-3)$-minors, 1 is to the left of 2, then 4 must be to the left of exactly two of 1, 2, and 3.

If in $2(n-4) \leq k \leq 2n-4$ of these $(n-3)$-minors, 1 is to the left of 2, then 4 must be to the left of exactly one of 1, 2, and 3.

If in $3(n-4) \leq k \leq 3n-8$ of these $(n-3)$-minors, 1 is to the left of 2, then 4 must be to the right of 1, 2, and 3.

Since $n \geq 13$, all of these cases are distinct.

We can determine where 1, 2, 3, and 4 are relative to one another.

Suppose we can determine where $1, 2, 3, ..., i$ are relative to one another. Then we will show how to determine where $i+1$ is relative to $1, 2, 3, ..., i$.

We use another induction argument here. First we will show how to find where $i+1$ is relative to 1, 2, 3, and 4.

Consider where 1 is relative to $i-2$ in the $(n-2)$-minors of p. These can all be found by where $1, 2, 3, ..., i$ are relative to each other except for the following cases:

$\binom{i-1}{3}$ times this relative order is determined by where 1 is relative to $i+1$,

$\binom{i-2}{2}$ times this is determined by where 2 is relative to $i+1$,

$\binom{i-3}{1}$ times this is determined by where 3 is relative to $i+1$, and

one time this is determined by where 4 is relative to $i+1$.

Since we know where 1, 2, 3, and 4 are relative to one another, this determines where $i+1$ is relative to 1, 2, 3, and 4.

Now suppose we know where $i+1$ is relative to $1, 2, ..., k-1$. To get where $i+1$ is relative to k, consider where $k-3$ is relative to $i-2$ in the $(n-2)$-minors of p. Except in the case where these two entries come from k and $i+1$, we know where $k-3$ and $i-2$ should be relative to one another in the $(n-3)$-minors of p because of the induction hypotheses. Now simply look at the remaining minors to determine where k is relative to $i+1$ in p.

Since we can determine where all the entries are relative to one another, we can find p from its $(n-3)$-minors.

$\diamond$

Theorem 5.12 $N_3 > 6$.

Proof: We simply need to show that there is a pair of permutations p and q each of length six such that $M_3(p) = M_3(q)$.

If we let $p = 135246$ and $q = 142536$ we can check that
$M_3(p) = \left\{123^{10}, 132^4, 213^4, 231, 312\right\} = M_3(q)$.

$\diamond$

5.5 Further Results and Directions

First it is good to note that if we have two permutations whose minor multisets are the same when removing $k-1$ entries at a time, then the minor multisets that result from removing k entries at a time will also be the same as is stated in the following lemma.

Lemma 5.13 *If $M_{k-1}(p) = M_{k-1}(q)$, then $M_k(p) = M_k(q)$.*

Before we prove this lemma, we will introduce the idea of removing entries from permutations in the multiset $M_k(p)$. Consider the multiset $M'_k(p)$ to be the multiset that it is formed by removing one at a time each entry of each permutation in $M_k(p)$ and, as before, the remaining entries of each permutation are appropriately renumbered.

Example 5.14 If we let $p = 2413$, then we have already seen that $M_1(p) = \{132, 213, 231, 312\}$. Removing one entry at a time from each permutation in this multiset and renumbering, we get $M'_1(p) = \{12^6, 21^6\}$.

Proof: (of the lemma)

Let p and q be two permutations such that $M_{k-1}(p) = M_{k-1}(q)$. Then consider the multisets $M'_{k-1}(p)$ and $M'_{k-1}(q)$. By the way that these multisets were formed and the fact that $M_{k-1}(p) = M_{k-1}(q)$, it is clear that $M'_{k-1}(p) = M'_{k-1}(q)$.

Also notice that each $n-k$ element subpermutation (up to renumbering) of p is contained in $n-k+1$ of the $(n-k+1)$-minors of p as there are $n-k+1$ ways to choose the last last entry. Then each $(n-k)$-minor of $M_k(p)$ will be represented exactly $n-k+1$ times in $M'_{k-1}(p)$.

Since $M'_{k-1}(p) = M'_{k-1}(q)$, we know $M_k(p) = M_k(q)$.

$\diamond$

The following proposition is due to Miklós Bóna. This proposition gives us a lower bound on what N_k can be for any value of k.

Proposition 5.15 *(Bóna)* $N_k \geq k + 4$.

Proof: By induction on k.

The result has been shown to be true for $k = 1$ and $k = 2$ in the previous sections. Suppose this proposition is true for $k - 1$ and we will prove it is also true for k.

To do this, we will show that there exist permutations P and Q each of length $k + 3$ such that $M_k(P) = M_k(Q)$. By the induction hypothesis, there exist permutations p and q, both of length $(k - 1) + 3$ such that $M_{k-1}(p) = M_{k-1}(q)$.

Now consider the permutations $1p$ and $1q$ which are obtained by putting a 1 in front of the appropriate permutation and increasing all the other values of the permutation by one.

The 3-minors of $1p$ (and similarly $1q$) can be obtained in one of the two following ways.

First, we consider the 3-minors of $1p$ that are obtained when the 1 of the permutation is removed. Then we are also removing $k - 1$ entries from the permutation p. In other words, we are simply obtaining the 3-minors of p. We already know that the 3-minors formed in such a way will be the same as when 1 and k other entries of $1q$ are eliminated to form 3-minors of $1q$.

Thus we only need consider the second case when 1 is not one of the entries removed from $1p$. The 3-minors formed in this way will all begin with 1 and then be followed by 2-minors of p (incremented up by one). By the previous lemma, we know that $M_k(p) = M_k(q)$. Hence these minors will also match up with those of $1q$ formed this way.

Therefore $M_k(1p) = M_k(1q)$, so $N_k \geq$ (length of $1p$) $+ 1 = k + 4$.

$\diamond$

The results for small k shown in this chapters suggest that perhaps the following may be true.

Conjecture 5.16 $N_k = k + 4$

If one could simply check relatively small values of n for each k in order to determine N_k, it would certainly make this process easier. One possible way to reduce the work would be to find an affirmative answer to the following question.

Question 5.17 *If $M_k(p)$ is unique for all p of length n, does this force $M_k(q)$ to be unique for all q of length $m > n$?*

REFERENCES

[1] M. D. Atkinson, M. M. Murphy, N. Ruškuc, Sorting with two ordered stacks in series. *Theoretical Computer Science,* **289** (2002), 205-223.

[2] M. Bóna, Combinatorics of Permutations. C.R.C. Press, Boca Raton, FL, 2004.

[3] M. Bóna, Exact Enumeration of 1342-avoiding permutations; A close link with labeled trees and planar maps. *Journal of Combinatorial Theory*, Series A **80** (1997), 257-272.

[4] M. Bóna, Parity and stack sortability. Presented at the Third International Conference on Pattern Avoiding Permutations (Gainesville, 2004).

[5] M. Bóna, A survey of stack sorting disciplines. *Electronic Journal of Combinatorics,* **9** (2002-2003), no. 2, Article 1.

[6] M. Bóna, A Walk Through Combinatorics: An Introduction to Enumeration and Graph Theory. World Scientific, River Edge, NJ, 2002.

[7] R. Cori, B. Jacquard, G. Schaeffer, Description trees for some families of planar maps. *Proceedings of the 9th Conference on Formal Power Series and Algebraic Combinatorics,* (Vienna, 1997), 196-208.

[8] S. Dulucq, S. Gire, J. West, Permutations with forbidden subsequences and nonseparable planar maps. *Proceedings of the 5th Conference on Formal Power Series and Algebraic Combinatorics,* (Florence, 1993), *Discrete Math.,* **153** (1996), 85-103.

[9] O. Eğecioğlu, The parity of the Catalan numbers via lattice paths. *Fibonacci Quart.* **21** (1983), no. 1, 65-66.

[10] I. P. Goulden, J. West, Raney paths and a combinatorial relationship between rooted nonseparable planar maps and two-stack-sortable permutations. *Journal of Combinatorial Theory*, Series A **75** (1996), 220-242.

[11] D. E. Knuth, Fundamental Algorithms, The Art of Computer Programming, vol. 1, 2nd ed. Addison-Wesley, Reading, MA 1973.

[12] G. Kreweras, Sur une classe des problèmes liés au treillis des partitions d'entiers. *Cahiers du B.U.R.O.,* **6** (1965), 5-105.

[13] A. Marcus, G. Tardos, Excluded permutation matrices and the Stanley-Wilf conjecture. *J. Combin. Theory* Series A **107** (2004), no. 1, 153-160.

[14] N. Robertson, P. D. Seymour, Graph Minors. XX. Wagner's Conjecture. *Journal of Combinatorial Theory*, Series B **92** (2004), no. 2, 325-357.

[15] Bruce Sagan, personal communication.

[16] R. P. Stanley, Log-concave and unimodal sequences in algebra, combinatorics, and geometry, in "Graph Theory and Its Applications: East and West." *Ann. NY Acad. Sci.,* **576** (1989), 500-535.

[17] Z. Stankova, J. West, A new class of Wilf-equivalent permutations. *J. Algebraic Combinatorics*, **15** (2002), no. 3, 271-290.

[18] R. Smith, Comparing algorithms for sorting with t stacks in series. *Annals of Combinatorics*, **8** (2004), 113-121.

[19] R. E. Tarjan, Sorting using networks of queues and stacks. *Journal of the ACM*, **19** (1972), 341-346

[20] J. W. Tutte, A census of planar maps. *Canadian Journal of Mathematics,* **33** (1963), 249-271.

[21] S. M. Ulam, A Collection of Mathematical Problems. Wiley, New York, NY, 1960.

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

[23] J. West, Sorting twice through a stack. *Theoretical Computer Science,* **117** (1993), 303-313.

[24] D. Zeilberger, A proof of Julian West's conjecture that the number of two-stack-sortable permutations of length n is $2(3n)!/((n+1)!(2n+1)!)$. *Discrete Math.,* **102** (1992), 85-93.

BIOGRAPHICAL SKETCH

I was born in Rochester, NY. I lived there until I moved to Indiana and then Florida to pursue graduate studies. I earned my bachelor's degree from Nazareth College in 1997 and my master's degree from Purdue University in 2000. I came to the University of Florida in 2001 and began studying under the direction of Miklós Bóna in 2002.