WATTS, MICHAEL D., M.A. On Characterizing Pairs of Permutations in Determining their Generated Group. (2013)
Directed by Dr. Greg Bell. 89 pp.

Among the unsolved problems in mathematics listed on Wolfram Mathworld's website is finding a formula for the probability that two permutations chosen at random generate the symmetric group [Wei13]. We show that two permutations of order two cannot generate the symmetric group using the maximum order of permutations.

We also define a new notion of *distance* in a permutation and use a concept of *distance preservation* to compare the generated subgroup with the symmetric group. We find that if a permutation preserves a distance which is not relatively prime to the lengths of the cycles in another permutation, then the pair of permutations will not generate the symmetric group. All commutative permutation subgroups and some imprimitive subgroups generated by pairs of permuations can be described by distance preservation. Using these results we are able to completely classify pairs of permutations involving a transposition which generate the symmetric group.

Lastly, we describe how to form the multiplication table for the symmetric group without performing any multiplications of permutations in the hopes that this description can be utilized in determining generated groups.

ON CHARACTERIZING PAIRS OF PERMUTATIONS IN DETERMINING THEIR GENERATED GROUP

by

Michael D. Watts

A Thesis Submitted to
the Faculty of the Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Arts

Greensboro
2013

Approved by

_____
Committee Chair

APPROVAL PAGE

This thesis has been approved by the following committee of the Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair _____
Greg Bell

Committee Members _____
Igor Erovenko

_____
Cliff Smyth

_____
Date of Acceptance by Committee

_____
Date of Final Oral Examination

ACKNOWLEDGMENTS

I would like to thank Dr. Greg Bell for his advising and guidance through the thesis process, and his assistance with the content herein. I also thank the committe members, Dr. Igor Erovenko and Dr. Cliff Smyth, for their support.

I also wish to thank Dr. Mark Stankus of Cal Poly, San Luis Obispo, for introducing me to the problem and aiding in my initial research.

TABLE OF CONTENTS

Page

LIST OF TABLES v

LIST OF FIGURES vi

CHAPTER

I. INTRODUCTION 1

1.1. History 1
1.2. Basics and Notation 3
1.3. Rudimentary Results 6

II. ON PERMUTATIONS OF ORDER TWO 9

2.1. Maximum Order of a Permutation 10
2.2. Permutations of Order Two 17

III. ON DISTANCES AND IMPRIMITIVE SUBGROUPS 22

3.1. Distance in Permutations 22
3.2. On Common Distance Divisors 27
3.3. On Imprimitive Subgroups 37

IV. PAIRS INVOLVING A TRANSPOSITION 45

V. THE MULTIPLICATION TABLE 49

REFERENCES 57

APPENDIX A. COMPUTING PERMUTATION GROUPS 58

LIST OF TABLES

Page

Table 1. Mutiplication Table for S_3 50

LIST OF FIGURES

Page

Figure 1. Comparison of Upper Bounds on Landau's Function 14

Figure 2. Representation of the Multiplcation Table for S_4 52

CHAPTER I

INTRODUCTION

1.1 History

Generating random permutations has broad applications including the use in cryptography, where an *ideal block cipher* is defined as a random permutation. Being able to generate a random permutation also has great importance to computational group theory [Pak00]. Due to the complexity of cryptography, creating an efficient algorithm to generate random permutations is crucial. One of the more surprising methods for generating random permutations that runs in polynomial time [Pak00] is the heuristic approach known as the product replacement algorithm, designed by Leedham-Green and Soicher [LP01]. The product replacement algorithm is defined for finite groups in general, though this paper's focus is on permutation groups.

The product replacement algorithm works as follows [LP01]: Given a finite group G, let $X = \{x_1, x_2, \ldots, x_k\}$ be a generating set for G. Define a move on the generating set in the following way: Randomly choose a pair (i, j) such that $1 \leq i \neq j \leq k$, then with equal probability replace x_i with either $x_i \cdot x_j$ or $x_j \cdot x_i$. Note that these moves map a generating set to a generating set. Apply these moves a number of times and return a random element of the resulting generating set. This is the desired random element of the group G.

The randomness of the product replacement algorithm depends on the size of the generating set and the number of moves. It turns out that a relatively small number of moves, dependent on the size of the generating set, gives a stastically random element

of the group. The product replacement algorithm has proven to be so efficient that it is used in two of the most common computer algebra packages GAP and MAGMA, despite the fact that we're not entirely sure why it works so well. In order for the algorithm to be effective, is seems necessary that the generating set X be randomly constructed and that it actually generate all of the group. Since this paper's focus is on permutations and the symmetric group, we will focus on a generating set of size two, which is sufficient to generate the symmetric group.

It was conjectured by Netto and later proved by Dixon [Dix69], that the probability that two permutations generate the symmetric group S_n approachs $\frac{3}{4}$ as n approaches infinity. More specifically, the probability that two permutations chosen at random generate either the symmetric group S_n or the alternating group A_n converges to 1 as n approaches infinity. Since the alternating group contains excatly $\frac{1}{2}$ of the permutations of S_n, exactly $\frac{1}{4}$ of the possible pairs of permutations of S_n are a subset of A_n. Thus, given two even permutations at random, there is an almost certainty that the permutations will generate the entire alternating group, whereas given two permutations not both even, there is an almost certainty that the permutations will generate the entire symmetric group, for large values of n.

A formula for the precise probability that two permutations chosen at random generate the symmetric group is an unsolved problem and is notable enough that it is listed as one of the unsolved problems in mathematics on Wolfram Mathworld's website [Wei13]. The purpose of this paper is to characterize some pairs of permutations which do not generate the symmetric group without computing any elements in their generated group.

1.2 Basics and Notation

Let X be any non-empty set. The collection of all bijections on the set X forms a group under function composition with identity being the identity map, denoted *id*. We refer to the bijections as *permutations* and denote composition as multiplication. The inverse elements of the symmetric group are the inverse functions, and we denote the inverse of a permutation $\sigma \in S_X$ as σ^{-1}. Observe that in general, $\sigma\tau$ is not the same as $\tau\sigma$.

Since this paper will only be focused on finite symmetric groups, we can relabel X such that $X = \{1, 2, \ldots, n\}$ where $n = |X|$. We will use the notation S_n to mean the symmetric group acting on a set of n elements, but we will only refer to the elements of the set as natural numbers. For the purposes of this paper, it will be understood when referring to the symmetric group as the group of permutations acting on $\{1, \ldots, n\}$. Since there are exactly $n!$ arrangements of a set of size n, $|S_n| = n!$.

A *cycle* is an ordered list of distinct integers which defines how a permutation permutes each integer of the cycle. The cycle $(a_1 a_2 \ldots a_k)$ represents a permutation which sends a_i to a_{i+1} for $1 \leq i < k$ and sends a_k to a_1. We define the *length* of a cycle to be the number of integers that appear in the cycle. Permutations can be uniquely represented as a multiplication of disjoint cycles of varying lengths, up to order, known as the *cyclic form* of the permutation. Omitting cycle of singular length, e.g. (5) or (n), gives us the *cycle decomposition* of a permutation. For example, the permutation $\sigma = (1\ 3)$ in S_4 is understood to represent the permutation defined by the mappings $\sigma(1) = 3, \sigma(2) = 2, \sigma(3) = 1$, and $\sigma(4) = 4$.

A *transposition* is a permutation which interchanges two elements but fixes the rest. For example, the permutation with cycle decomposition (1 5) is a transposition

in S_5, and every transposition in S_n has cycle decomposition $(a\ b)$ for some $a, b \in \{1, \ldots, n\}$. Every permutation can be written as a product of transpositions, though this product is not unique. However, the number of transpositions that can be used in product to represent a permutation is unique modulo 2. In other words, if a permutation can be written as a product of p transpositions and q transpositions, then $p \equiv q \pmod 2$. We call a permutation *even* if the permutation can be written as a product of an even number of transpositions, and *odd* otherwise. The set of even permutations of S_n forms a subgroup called the *alternating group* and is denoted A_n.

The order of a permutation is equivalent to the least common multiple (lcm) of the lengths of the cycles of the cycle decomposition. If $\sigma \in S_n$ has cyclic form whose cycles has lengths $k_1, k_2, \ldots, k_m$ where $k_1 \leq k_2 \leq \cdots \leq k_m$ (including its 1-cycles), then the list of integers $k_1, k_2, \ldots, k_m$ is called the *cycle type* of σ.

The symmetric group S_n acts on the set $\{1, \ldots, n\}$, so a subgroup G of S_n affords an action on $\{1, \ldots, n\}$. The *orbits* of a subgroup are the equivalence classes of the group action which form a parition on the set $\{1, \ldots, n\}$. We also wish to describe the orbits of a permutation and so we will distinguish between the orbits of a group and the orbits of a permutation by calling the former *group orbits* and the latter the *orbits of a permutation.*

Definition 1.1. The *orbits of a permutation* are the sets corresponding to the cycles of the permutation. In particular, the orbits of a permutation are the orbits of the group generated by the permutation.

Example 1.2. The orbits of the permutation $(1\ 2\ 3)(4\ 5) \in S_6$ are $\{1, 2, 3\}, \{4, 5\}$, and $\{6\}$.

A group action is called *transitive* if there is only one group orbit, and *intransitive* otherwise. Similarly, a subgroup of S_n is called *transitive* if it's group action on $\{1, \ldots, n\}$ is transitive, and *intransitive* otherwise. Note that the symmetric group acts transitively on $\{1, \ldots, n\}$.

If σ and τ are any two permutation of S_n, then σ and $\tau\sigma\tau^{-1}$ are called *conjugates*. Conjugation in S_n can be calculated using the following lemma:

Lemma 1.3. *Let $\sigma, \tau \in S_n$. If $(a_1\ a_2\ \cdots\ a_k)$ is a cycle of σ then*

$$(\tau(a_1)\ \tau(a_2)\ \cdots\ \tau(a_k))$$

is a cycle of $\tau\sigma\tau^{-1}$. In other words, the cyclic form of $\tau\sigma\tau^{-1}$ can be constructed by replacing each a in the cyclic form of σ by $\tau(a)$. In particular, σ and any conjugate of σ by permutations in S_n have the same cycle type.

Proof. Let $\rho = \tau\sigma\tau^{-1}$ and $a \in \{1, \ldots, n\}$. Then $\rho(\tau(a)) = \tau\sigma\tau^{-1}\tau(a) = \tau\sigma(a)$. Then if $(a_1\ a_2\ \cdots\ a_k)$ is a cycle of σ, then ρ maps $\tau(a_i) \mapsto \tau(a_{i+1})$ for $1 < i \leq k$ and $\tau(a_k) \mapsto \tau(a_1)$. Then ρ has cycle $(\tau(a_1)\ \tau(a_2)\ \cdots\ \tau(a_k))$. □

Lastly, we will be using modular arithmetic later in Chapter III. For this we will repeatedly use the following lemma without further mention:

Lemma 1.4. *Let $g, k, d \in \mathbb{N}$ and suppose g divides k and d. Then g divides $d \pmod{k}$, which means that g divides every representative of the equivalence class of d modulo k.*

Proof. Since g divides k and d, then $k = gr$ and $d = gs$ for some $r, s \in \mathbb{N}$. Then for all $t \in \mathbb{Z}$, $d \equiv d + kt \pmod{k}$ and so $d \equiv gs + grt \pmod{k}$, which is precisely every

representative of the equivalence class of d modulo k, $\overline{d}$. Therefore every $x \in \overline{d}$ can be written as $x = gs + grt = g(s + rt)$ and thus g divides x. □

For the purposes of this paper, we mean $\mathbb{N}$ to be the set $\{1, 2, 3, \ldots\}$.

1.3 Rudimentary Results

A few of the pairs of permutations are easily identifiable as not a generating set of the symmetric group. They are well known facts that we will mention and summarize here with little explanation.

Lemma 1.5. *Let $n \geq 2$ and $\sigma, \tau \in S_n$ such that both σ and τ are even. Then the group generated by σ and τ is not all of S_n. More specifically, $\langle \sigma, \tau \rangle \leq A_n$.*

Proof. If σ and τ are both even, they can both be written as a product of an even number of transpositions. Then any product of σ and τ can also be written as an even number of transpositions (the sum of two even integers is even). Thus for any $\rho \in \langle \sigma, \tau \rangle$, $\rho \in A_n$. Therefore $\langle \sigma, \tau \rangle \leq A_n$. □

Lemma 1.6. *Let $n \geq 3$ and $\sigma, \tau \in S_n$ such that $\sigma\tau = \tau\sigma$. Then the group generated by σ and τ is not all of S_n.*

Proof. Since σ and τ commute, then the group $\langle \sigma, \tau \rangle$ is commutative. Consider the permutations (1 2) and (1 3) in S_n. By a quick calculation, (1 2)(1 3) = (1 3 2) and (1 3)(1 2) = (1 2 3). Thus there exist non-commutative permutations in S_n and so S_n is not commutative. Therefore $\langle \sigma, \tau \rangle \neq S_n$. □

It seems difficult to determine whether two permutations commute without computing the product, which we are trying to avoid. We will return to this question in Chapter III.

Lemma 1.7. *Let $\sigma, \tau \in S_n$ such that $\langle \sigma, \tau \rangle$ is not transitive. Then the group generated by σ and τ is not all of S_n.*

Proof. The result is clear from the fact that S_n acts transitively on $\{1, \ldots, n\}$. □

Again, we do not wish to actually construct the group to determine whether the group is all of S_n or not. However, we can deduce transitivity based off the generating set.

Lemma 1.8. *The orbits of a group G generated by a set of permutations $X \subseteq S_n$ are the unions of the orbits of the permutations of X which intersect. In other words, if $\mathcal{O}$ is an orbit of G, and $\mathcal{A}$ is an orbit of some $\sigma \in X$ such that $\mathcal{A} \cap \mathcal{O} \neq \emptyset$, then $\mathcal{A} \subseteq \mathcal{O}$.*

Proof. Let $\mathcal{O}$ be an orbit of a group G generated by a set of permutations $X \subseteq S_n$ and, without loss of generality, suppose $\mathcal{A}$ is an orbit of σ such that $\mathcal{A} \cap \mathcal{O} \neq \emptyset$. Let $a \in \{1, \ldots, n\}$ and $x \in \mathcal{A} \cap \mathcal{O}$. Then $\sigma^r(a) = x$ for some $r \in \mathbb{Z}$ since $x \in \mathcal{A}$. But $\sigma^r \in G$ so then there exists $\rho \in G$ such that $\rho(a) = x$. Then $a \in \mathcal{O}$ since $x \in \mathcal{O}$. Therefore $\mathcal{A} \subseteq \mathcal{O}$. □

Example 1.9. Consider the permutations $(1\ 2)(3\ 4\ 5)(6\ 7)(8\ 9)$ and $(1\ 6)(4\ 5)(3\ 8\ 10)$ in S_{10}. The orbits of the group generated by the permutations are $\{1, 2, 6, 7\}$ and $\{3, 4, 5, 8, 9, 10\}$ which is obtained by combining the cycles of σ and τ which intersect. Therefore these permutations do not generate all of S_n since the group is not transitive.

A summary of the results is as follows:

Proposition 1.10. *Let $\sigma, \tau \in S_n$ for $n \geq 3$. The group generated by σ and τ is not the symmetric group if one of the following is true:*

(1) $\sigma, \tau \in A_n$,

(2) $\tau \in \langle \sigma \rangle$ *or* $\sigma \in \langle \tau \rangle$, *or*

(3) $\langle \sigma, \tau \rangle$ *is not transitive.*

Each of these results come from basic understandings of permutation groups. The following chapters will delve into more complex structures that may not be immediately obvious. Since S_1 and S_2 are easily understood and uninteresting, for the purposes of this paper we will assume that when referring to S_n we mean for $n \geq 3$, unless otherwise stated. We will also further assume that the permutation group $\langle \sigma, \tau \rangle$ is transitive and neither is the identity permutation. Since we do not know, as of yet, an easy way to determine whether two permutations are in the same group generated by a single permutation, we will not assume that (2) of Lemma 1.10 is not the case for a pair of permutations. We will describe how to identify these pairs in Chapter III.

CHAPTER II

ON PERMUTATIONS OF ORDER TWO

Permutations in a group can be seen as words whose letters are composed of the members of the generating set of the group. When the generating set contains only two elements of order 2, all words can be represented as a series of alternating generators, in that if the group is generated by σ and τ each of order 2, then a generic element ρ will always look something like (up to the choice of the first and last permutation)

$$\rho = \sigma\tau\sigma\tau\cdots\sigma\tau$$

since σ^n or τ^n simplifies to either σ, τ or the identity for all $n \in \mathbb{N}$. Given this sequence, we can rewrite it by coupling $\tau\sigma$ pairs as a single permutation giving us the form

$$\rho = \sigma^s(\tau\sigma)^r\tau^t$$

where s and t are either 0 or 1. But $\tau\sigma$ has some maximal order, so there are only so many ways we can write ρ, a generic permutation in $\langle\sigma, \tau\rangle$. It is then our goal to determine the maximum order a permutation may have in S_n so that we give an upper bound on the size of $|\langle\sigma, \tau\rangle|$.

2.1 Maximum Order of a Permutation

The order of a given permutation σ is the least common multiple of the lengths of the cycles in the cycle decomposition of σ. The maximum order of a permutation in S_n, then, is the maximum least common multiple of all possible combinations of lengths. The maximum order was given as a function observed by Landau [Nic97].

Definition 2.1. Landau's function, denoted $g(n)$, is the maximum order of a permutation in S_n. Landau's function is given by

$$g(n) = \max\{lcm(a_1, a_2, \cdots, a_k) \mid a_i \in \mathbb{N} \text{ and } a_1 + a_2 + \cdots + a_k = n\}$$

A formula for the precise value of $g(n)$ is not known and remains a difficult problem [DNZ08]. There are some known upper bounds for $g(n)$ (see the remark at the end of the section) but they are more complicated than the comparison we will be making. We wish to calculate the least common multiple of all possible lengths of cycles which add to n. If $\{a_1, a_2, \ldots, a_k\}$ are the lengths of the cycles of a permutation, then the order of the permutation is

$$g(n) = lcm(a_1, a_2, \ldots, a_k) = \prod_{i=1}^{m} p_i$$

where the p_i's are the distinct primes of the factorizations of the a_j's and

$$p_1 + p_2 + \cdots + p_m \leq a_1 + a_2 + \cdots + a_k = n.$$

Then in determining an upper bound for $g(n)$, it will suffice to consider lengths of cycles which are themselves distinct primes. However, determining all these primes is difficult, so we will resort to averaging odd integers that sum to n.

Lemma 2.2. *Let $n \in \mathbb{N}$, and let X be a set of distinct primes whose sum is less than or equal to n. Then*

$$|X| \leq \left\lfloor \sqrt{n-1} \right\rfloor$$

Proof. Let $n \in \mathbb{N}$ and

$$X = \{a_1, a_2, \ldots, a_k \mid a_1 + a_2 + \cdots + a_k \leq n, a_i \text{ distinct primes}\}$$

be an ordered set where $a_i < a_{i+1}$. In order to maximize $|X|$, it suffices to minimize the values of the a_i's. To minimize the values, it suffices to set a_i to be the ith prime, where i ranges from 1 up to some value k such that $\sum a_i \leq n$. While the a_i's may add up to less than n, the corresponding greatest index k will still maximize $|X|$.

In order to create a closed formula for an upper bound on the maximum index, it suffices setting $a_1 = 2$ and $a_i = 2i-1$ for each $i \geq 2$, thus creating a set of 2 and all the odd numbers greater than or equal to 3. Thus we have that $2+3+5+\cdots+2k-1 \leq n$, or $2 + 1 + 3 + 5 + \cdots + 2k - 1 \leq n + 1$. Using the formula for the sum of the first k odd integers (k^2) we have that

$$\begin{aligned} & 2 + (k)^2 && \leq n + 1 \\ \Rightarrow \quad & k^2 && \leq n - 1 \\ \Rightarrow \quad & k && \leq \sqrt{n-1}. \end{aligned}$$

Thus we have that since k is the greatest possible index of the a_i's, then

$$|X| \leq \sqrt{n-1}$$

Note that $|X|$ must be an integer, thus we can take the floor of the above calculation and get our desired result. □

Now that we have an estimate to the number of distinct primes, we can use their average in determining an upper bound for Landau's function.

Lemma 2.3. *Let $k_n = \lfloor \sqrt{n-1} \rfloor$. Landau's function, $g(n)$, has an upper bound of*

$$g(n) \leq \left(\frac{n}{k_n}\right)^{k_n}.$$

Proof. From the definition of Landau's function, $g(n)$ is the maximum of the least common multiple of integers that add to n. Letting X denote a set of positive integers which add to n, the least common multiple over X is the product of the distinct prime factors of the integers in the set X. The maximum number of distinct prime numbers on this set is bounded above by $\lfloor \sqrt{n-1} \rfloor$, by Lemma 2.2.

Let $k_n = \lfloor \sqrt{n-1} \rfloor$ and let (b_i) be a finite sequence of all the distinct prime factors of the set X, $b_1, b_2, \ldots, b_k$. The sequence (b_i) is at most k_n long since $\sum b_i \leq n$ and so $k \leq k_n$. Then

$$lcm(a_1, a_2, \ldots, a_k) = \prod_{i=1}^{k} b_i.$$

By the arithmetic geometric mean inequality, we have that

$$\begin{aligned}\prod_{i=1}^{k} b_i &\leq \left(\frac{\sum_{i=1}^{k} b_i}{k}\right)^k \\ &\leq \left(\frac{n}{k}\right)^k \\ &\leq \left(\frac{n}{k_n}\right)^{k_n} \quad (*)\end{aligned}$$

The proof of $(*)$ is left at the end of the section in Lemma 2.5. Therefore, $g(n) \leq \left(\frac{n}{k_n}\right)^{k_n}$. □

Remark. The upper bound on $g(n)$ we have introduced is not the lowest upper bound for large values of n. It has been shown that $\log g(n)$ asymptotically approaches $\sqrt{n \log n}$, i.e. $\log g(n) \sim \sqrt{n \log n}$, for large values of n [MNR89]. However, comparing $(n/k_n)^{k_n}$ with the asymptotic upper bound for $n \geq 4$

$$\log g(n) \leq \sqrt{n \log n}\left(1 + \frac{\log\log n - .975}{2 \log n}\right)$$

we find that $(n/k_n)^{k_n}$ is a closer approximation for smaller values of n, and arguably easier to compare with $n!$ using Stirling's formula, which we will describe in the next chapter.

The graph shown in Figure 1 shows the comparison of the two bounds where

$$\begin{aligned}f(n) &= \sqrt{n \log n}\left(1 + \frac{\log\log n - .975}{2 \log n}\right) \\ h(n) &= \log\left(\left(\frac{n}{k_n}\right)^{k_n}\right)\end{aligned}$$

such that $\log g(n) \leq f(n)$ and $\log g(n) \leq h(n)$.

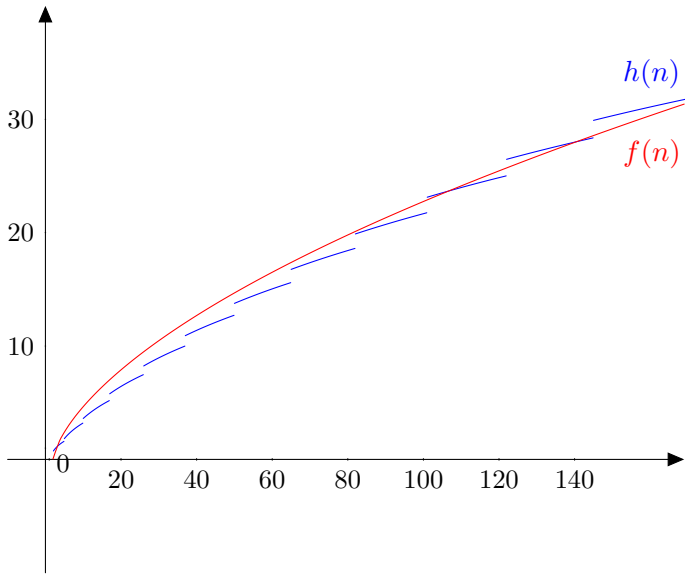

Figure 1. Comparison of Upper Bounds on Landau's Function

The following lemmas are used in the previous proofs, but we leave them at the end of the section since they are not particualrly interesting.

Lemma 2.4. *Fix $a \in \mathbb{R}$, where $a > 0$. Let*

$$a_k = \left(1 + \frac{a}{k}\right)^k.$$

Then the sequence $a_1, a_2, a_3, \ldots$ is increasing and converges to e^a as $k \to \infty$.

Proof. The sequence is well-known to converge to e^a as $k \to \infty$. It remains to show that the sequence is increasing.

Let $f(k)$ be a real valued function such that

$$f(k) = \ln\left(\left(1+\frac{a}{k}\right)^k\right)$$

Note that f is continuous for all $k > 0$. Consider the first and second derivatives of the function on $(0, \infty)$:

$$\begin{aligned} f(k) &= k\ln(1+\frac{a}{k}) \\ f'(k) &= \frac{k}{1+\frac{a}{k}} \cdot \frac{-a}{k^2} + \ln(1+\frac{a}{k}) \\ &= \frac{-a}{k+a} + \ln(1+\frac{a}{k}) \\ f''(k) &= \frac{a}{(k+a)^2} + \frac{-a}{k^2+ka} \\ &= -\frac{a^2}{k(k+a)^2} \end{aligned}$$

The second derivative is always negative for positive k, so the first derivative is always decreasing for positive k. Notice that

$$\lim_{k\to\infty} f'(k) = 0 + \ln(1) = 0$$

and $f'(a) = -\frac{1}{2} + \ln(2) > 0$. Thus since the first derivative is positive at some point, is always decreasing, and is converging to 0, then the first derivative is always positive. Therefore the function $f(k)$ is always increasing on $(0, \infty)$. Thus the sequence $\left(1+\frac{a}{k}\right)^k$ starting at $k = 1$ is increasing and converges to e^a. □

Lemma 2.5. *Let $n \in \mathbb{N}$, $n \geq 4$ and $k_n = \lfloor \sqrt{n-1} \rfloor$. For any positive integer k such that $k \leq k_n$,*

$$\left(\frac{n}{k}\right)^k \leq \left(\frac{n}{k_n}\right)^{k_n}.$$

Proof. Let n and k_n be as in the hypothesis. If $k = k_n$ there's nothing to prove. If $n = 4$, then $k_n = 1$, so there is no smaller such positive integer to compare. If $n = 5$, then $k_n = 2$ and the only smaller positive integer is 1. But $(5/1)^1 = 5 < 6.25 = (5/2)^2$. Thus the hypothesis holds true for $n = 4$ and $n = 5$. Now let $n \geq 6$. Since $k_6 = 2$ and $6/e > 2$, then $\frac{n}{e} > k_n$ for all $n \geq 6$ since $\frac{n}{e}$ increases faster than $\lfloor \sqrt{n-1} \rfloor$. Thus, for all $n \geq 4$ we have that $k_n < \frac{n}{e}$.

Let k be a positive integer such that $k < k_n$. Then $k_n = k + a$ for some $a \geq 1$. So $k < k + a = k_n < \frac{n}{e}$. Thus

$$\begin{aligned}
& & k + a &< \tfrac{n}{e} \\
\Rightarrow & & (k+a)^a &< \left(\tfrac{n}{e}\right)^a \quad \text{(since } a \geq 1\text{)} \\
\Rightarrow & & \tfrac{n^a}{(k+a)^a} &> e^a \\
& & &> \left(1 + \tfrac{a}{k}\right)^k \quad \text{(by Lemma 2.4)} \\
\Rightarrow & & \tfrac{n^a}{(k+a)^a} &> \left(\tfrac{k+a}{k}\right)^k \\
\Rightarrow & & \tfrac{n^a \cdot n^k}{(k+a)^a (k+a)^k} &> \tfrac{n^k}{k^k} \\
\Rightarrow & & \left(\tfrac{n}{k+a}\right)^{k+a} &> \left(\tfrac{n}{k}\right)^k \\
\Rightarrow & & \left(\tfrac{n}{k}\right)^k &< \left(\tfrac{n}{k_n}\right)^{k_n}
\end{aligned}$$

□

2.2 Permutations of Order Two

We now return our attention to the question of how many ways we can uniquely represent a permutation in a group generated by two permutations. As mentioned in the beginning of this chapter, if ρ is in a group generated by two elements σ, τ of order 2, then ρ has the form (up to the choice of the first and last permutations)

$$\rho = \sigma(\tau\sigma)^r\tau$$

Then the number of ways to uniquely represent permutations in $\langle \sigma, \tau \rangle$ is done by multiplying the choices for the first, last, and middle permutations. There are 2 choices for the first and last permutations and at most $g(n)$ choices for the middle portion, thus $4g(n)$ choices all together. We will first show that $4g(n)$ is strictly less than $n!$ utilizing Stirling's approximation for $n!$ and then formalize the proof that two permutations of order two cannot generate the symmetric group.

Lemma 2.6. Stirling's formula [Rob55]. *For large values of n,*

$$n! \sim \sqrt{2\pi n} n^n e^{-n}$$

where $\sim$ denotes asymptotic convergence. Moreover, for $n \in \mathbb{N}$,

$$\sqrt{2\pi n} n^n e^{-n+1/(12n+1)} < n! < \sqrt{2\pi n} n^n e^{-n+1/(12n)}$$

Using the bounds on $n!$ given by Stirling's formula will be useful in comparing $4g(n)$ with $n!$.

Lemma 2.7. *Let $n \geq 4$ and let $g(n)$ denote Landau's function, that is $g(n)$ is the maximum order of a permutation in S_n. Then*

$$4g(n) < n!$$

Proof. By Lemma 2.3, $g(n)$ is bounded above by

$$\left(\frac{n}{k_n}\right)^{k_n}$$

where $k_n = \lfloor n - 1 \rfloor$. Then

$$4g(n) \leq 4\left(\frac{n}{k_n}\right)^{k_n}$$

Note that k_n is a non-decreasing function of n, and by direct calculation $k_4 = 1, k_5 = 2$ and $k_9 = 2$. Thus $k_n = 2$ for $5 \leq n \leq 9$.

Case $(n = 4)$: If $n = 4$ then $k_n = 1$ and $4(n/k_n)^{k_n} = 4 \cdot 4 = 16 < 24 = 4!$.

Case $(5 \leq n \leq 9)$: If $5 \leq n \leq 9$ then $k_n = 2$ and

$$4\left(\frac{n}{k_n}\right)^{k_n} = 4\left(\frac{n}{2}\right)^2 = n^2 < n!.$$

Case $(n \geq 10)$: If $n \geq 10$, we have that $k_n \geq k_{10} = 3 > e$. We also have the following upper bound on k_n:

$$k_n = \lfloor \sqrt{n-1} \rfloor < n$$

for $n \geq 10$. Thus, for $n \geq 10$, $n/e > 1, 4 < \sqrt{2\pi n}$ and

$$\begin{aligned} 4\left(\frac{n}{k_n}\right)^{k_n} &\leq 4\left(\frac{n}{e}\right)^{k_n} \\ &< 4\left(\frac{n}{e}\right)^{n} \\ &< \sqrt{2\pi n}\left(\frac{n}{e}\right)^{n} \\ &< n! \end{aligned}$$

by Stirling's Formula. Therefore for $n \geq 4$ we have that $4g(n) \leq 4\left(\frac{n}{k_n}\right)^{k_n} < n!$. □

Using this result we can proceed to prove our desired hypothesis and goal of this chapter.

Proposition 2.8. *For $n \geq 4$, a group generated by a pair of permutations of order 2 is not S_n.*

Proof. Let $n \geq 4$ and $\sigma, \tau \in S_n$ be two permutations of order 2. Consider a permutation ρ in the group generated by σ and τ, $\rho \in \langle \sigma, \tau \rangle$. Any permutation in $\langle \sigma, \tau \rangle$ can be represented by a product of a finite sequence of alternating σ's and τ's since $\sigma^2 = \tau^2 = id$. Then ρ begins and ends with either σ or τ, and so ρ has the following form:

$$\rho = \sigma^s \tau\sigma\tau\sigma \cdots \tau\sigma\tau^t$$

where s and t are either 1 or 0 which effectively lets ρ begin and end with either σ or τ. We can rewrite β by coupling the $\tau\sigma$ pairs in the middle:

$$\beta = \sigma^s (\tau\sigma)^r \tau^t$$

where r is a non-negative integer. Moreover, it suffices to take r to be less than the order of $\tau\sigma$, since otherwise we would be repeating elements of $\langle\sigma\tau\rangle$. In other words, r is at most $g(n)$, the maximum order of an element of S_n.

Since every permutation in $\langle\sigma,\tau\rangle$ can be written as ρ, we can count the combinations of choices for s, r, and t. There are 2 choices for s and t, and at most $g(n)$ choices for r. Thus the maximum number of unique ways to write ρ or the size of the group $\langle\sigma,\tau\rangle$, is

$$|\langle\sigma,\tau\rangle| \leq \text{choices of } s \cdot \text{choices of } r \cdot \text{choices of } t \leq 2 \cdot g(n) \cdot 2 = 4g(n).$$

By Lemma 2.7, $4g(n) < n!$, so $|\langle\sigma,\tau\rangle| < n! = |S_n|$. Therefore the group generated by σ and τ cannot be all of S_n. Since σ, τ were taken arbitrarily, no such pair of permutations of order 2 can generate S_n for $n \geq 4$. □

Remark. An alternative method to proving that two permutations of order 2 cannot generate the symmetric group is to look at the group presentation. If σ and τ are two such permutations, then $\sigma^2 = 1$ and $\tau^2 = 1$. If the group is not transitive, then $\langle\sigma,\tau\rangle \neq S_n$. If the group is transitive, then the product $\sigma\tau$ is an n-cycle for n odd and thus $(\sigma\tau)^n = 1$. Then $\langle\sigma,\tau\rangle$ has presentation

$$\langle\sigma,\tau \mid \sigma^2 = \tau^2 = (\sigma\tau)^n = 1\rangle$$

which is a presentation of the Dihedral group acting on n vertices, D_{2n} [DF04, p. 28]. We know that D_{2n} is a proper subgroup of S_n for $n \geq 4$ and thus no such group generated by a pair of permutations of order 2 can generate S_n for $n \geq 4$.

Identifying a permutation of order 2 (and thus a pair of permutations of order 2) *prima facie* is simple in that the cycle decomposition is a product of disjoint transpositions.

Example 2.9. Consider the permutations $\sigma = (1\ 2)(3\ 4)(5\ 6)$ and $\tau = (2\ 3)(4\ 5)(6\ 7)$ in S_7. Then $\langle \sigma, \tau \rangle \neq S_7$ since $|\sigma| = |\tau| = 2$.

CHAPTER III

ON DISTANCES AND IMPRIMITIVE SUBGROUPS

One identifying characteristic of the cycle decomposition of a permutation is the *distance* between two integers, where we mean distance to be how many *moves*, or applications of the permutation, it takes to get from one integer to another in the cycle decomposition. Using the word *distance* is a little abuse of the term in that the direction in which you observe the distance between two integers matters. For example, in the cycle (1 2 3) you might observe the distance to be 2 between integers 1 and 3 if you move to the right. But as you move to the left you might observe the distance to be 1. However there is a relationship between the two observed distances modulo 3.

We will use modular arithmetic to describe the properties exhibited by this notion of *distance* in permutations and show that some pairs of permutations preserve a sort of greatest common divisor of distances.

3.1 Distance in Permutations

Definition 3.1. Let τ be a permutation in S_n. Let the map $d_\tau : \{1, \ldots, n\} \times \{1, \ldots, n\} \to \mathbb{N} \cup \{0, \infty\}$ be defined by, for $a, b \in \{1, \ldots, n\}$,

$$d_\tau(a,b) = \begin{cases} \min\{d \in \mathbb{N} \cup \{0\} \mid \tau^d(a) = b\} & \text{if such a } d \text{ exists} \\ \infty & \text{otherwise} \end{cases}$$

We call $d_\tau(a,b)$ the distance in τ from a to b.

Since distances in permutations are defined to be integers or infinite, it will be understood when identifying distances to mean such. We will also define what it means to divide in the set $\mathbb{N} \cup \{0, \infty\}$ by the usual definition of divisibility and that nothing divides infinity, including infinity itself.

Remark. Calling d_τ a distance function alludes to the idea that d is a metric. This would be a false claim since in general $d_\tau(a,b) \neq d_\tau(b,a)$. However, as we will see later, d_τ has properties that are similar to a metric and it would not be too abusive of the term distance.

It should be noted that $d = d_\tau(a,b)$ is either infinity or less than the length k of the cycle which contains both a and b since $\tau^{d+rk}(a) = \tau^d(a)$ for all $r \in \mathbb{Z}$. This is due to the fact that τ restricted to the cycle c of length k containing a is a k-cycle of order k, and so $\tau|_c^{rk} = id|_c$. Then we can refer to distances in τ using modulo arithmetic, and indeed the distance function d_τ has metric-like properties using modulo arithmetic.

Proposition 3.2. *Let d_τ be the distance function of $\tau \in S_n$. Then for all $a, b, c \in \{1, \ldots, n\}$*

Regardless of choice of a and b

(i) $d_\tau(a,a) = 0$*;*

(ii) $d_\tau(a,b) > 0$ *if* $a \neq b$*;*

If a, b, and c are in the same cycle of length k in τ, then

(iii) $d_\tau(a,b) \equiv -d_\tau(b,a) \pmod{k}$*;*

(iv) $d_\tau(a,b) \equiv d_\tau(a,c) + d_\tau(c,b) \pmod{k}$*;*

If a, b are not in the same cycle in τ, then

(v) $d_\tau(a,b) = d_\tau(b,a)$;

(vi) $d_\tau(a,b) = d_\tau(a,c)$ *or* $d_\tau(a,b) = d_\tau(c,b)$.

Proof. Let $a, b, c \in \{1, \ldots, n\}$.

(i) Since $\tau^0 = id$, then $\tau^0(a) = a$ for all $a \in \{1, \ldots, n\}$. Thus $d_\tau(a,a) = 0$.

(ii) By definition of d_τ, $d_\tau(a,b) \geq 0$. If $a \neq b$, then $\tau^0(a) = a \neq b$. Thus $d_\tau(a,b) > 0$ if $a \neq b$.

(iv) Let a, b, c be in the same cycle of length k. Let $d = d_\tau(a,b)$, $d_1 = d_\tau(a,c)$ and $d_2 = d_\tau(c,b)$. Then $d, d_1, d_2 < \infty$ since a, b, c are in the same cycle. Then $\tau^d(a) = b, \tau^{d_1}(a) = c$, and $\tau^{d_2}(c) = b$. Thus

$$\begin{aligned} \tau^{d_2+d_1}(a) &= \tau^{d_2} \circ \tau^{d_1}(a) \\ &= \tau^{d_2}(c) \\ &= b \\ &= \tau^d(a) \end{aligned}$$

Therefore $\tau^{d_2+d_1-d}(a) = a$. But $\tau^r(a) = a$ if and only if $r \equiv 0 \pmod{k}$. Thus $d_2 + d_1 - d \equiv 0 \pmod{k}$ and therefore $d \equiv d_1 + d_2 \pmod{k}$.

(iii) If a, b are in the same cycle of length k, then using the result from (iv)

$$\begin{aligned} d_\tau(a,b) &\equiv d_\tau(a,b) + d_\tau(b,a) + d_\tau(a,b) \pmod{k} \\ 0 &\equiv d_\tau(b,a) + d_\tau(a,b) \pmod{k} \\ d_\tau(a,b) &\equiv -d_\tau(b,a) \pmod{k} \end{aligned}$$

(v) If a, b are not in the same cycle, then $d_\tau(a, b) = \infty = d_\tau(b, a)$.

(vi) If a, b are not in the same cycle, for all c, either a and c are not in the same cycle or b and c are not in the same cycle, or both. Then $d_\tau(a, b) = \infty$ and either $d_\tau(a, c) = \infty$ or $d_\tau(c, b) = \infty$, or both. Then either $d_\tau(a, c) = d_\tau(a, b)$ or $d_\tau(a, c) = d_\tau(b, c)$.

□

Remark. When we mention a particular distance in a permutation, we are suggesting that such a distance exists in the first place. Therefore, any time we say that d is a distance in $\tau \in S_n$, we mean that there exists some $a, b \in \{1, \ldots, n\}$ such that $d_\tau(a, b) = d$. Since distances must be less than the length of the cycle which it is being observed in, then distance d in a permutation $\tau \in S_n$ is always less than the length of the largest cycle of τ. That is, if $k_1, k_2, \ldots, k_m$ is the cycle type of τ, letting

$$k = \max\{k_1, k_2, \ldots, k_m\}$$

then a distance d in τ is such that $0 \leq d < k$.

Example 3.3. Consider the permutation $\tau = (1\ 2\ 3\ 4)(5\ 6\ 7)$ in S_7. Then

$$d_\tau(5, 7) = 2$$
$$d_\tau(2, 5) = \infty$$
$$d_\tau(7, 5) = 1 \equiv -2 \pmod 3 \equiv -d_\tau(5, 7) \pmod 3$$
$$d_\tau(1, 3) = 2 \equiv 3 + 3 \pmod 4 \equiv d_\tau(1, 4) + d_\tau(4, 3) \pmod 4$$

Lemma 3.4. *Let τ be a permutation in S_n. Consider $a \in \{1, \ldots, n\}$ which belongs to a cycle in τ of length k. Then for all $r, s \in \mathbb{Z}$*

(1) $d_\tau(a, \tau^r(a)) \equiv r \pmod{k}$;

(2) $d_\tau(\tau^s(a), \tau^r(a)) \equiv r - s \pmod{k}$.

Proof. Let $r \in \mathbb{Z}$ and $a \in \{1, \ldots, n\}$ which belongs to a cycle of length k in τ.

(1) Note that $\tau^r(a) = \tau^r(a)$. Therefore by the definition of d_τ, $d_\tau(a, \tau^r(a)) \equiv r \pmod{k}$.

(2) Note that a and $\tau^r(a)$ are in the same cycle in τ for all $r \in \mathbb{Z}$. Using the triangle inequality on d_τ and the result from (1) we have that

$$\begin{aligned} d_\tau(\tau^s(a), \tau^r(a)) &\equiv d_\tau(\tau^s(a), a) + d_\tau(a, \tau^r(a)) \pmod{k} \\ &\equiv -d_\tau(a, \tau^s(a)) + d_\tau(a, \tau^r(a)) \pmod{k} \\ &\equiv -s + r \pmod{k} \end{aligned}$$

□

We can use this distance function to express how a permutation permutes the integers of another permutation. If both permutations preserve some notion of distance then the whole group generated by the permutations will as well, giving us a means of comparing the generated subgroup with the entire symmetric group. The next section will describe this preservation.

3.2 On Common Distance Divisors

It is well known that the symmetric group is generated by the pair of permutations $(1\ 2\ \cdots\ n)$ and $(1\ 2)$. To see this observe that conjugating $(1\ 2)$ by the $n-$cycle produces all transpositions of the form $(i\ i+1)$, which in turn generate all transpositions of S_n. Since every permutation can be written as a product of transpositions, the entire symmetric group is thus generated. The question is what property of the transposition $(1\ 2)$ is so special? Why not choose $(1\ 3)$ or $(5\ n)$? The answer is that determining a transposition to pair with the n-cycle that will generate the symmetric group depends on the *distance* in $(1\ 2\ \cdots\ n)$ between the integers in the transposition.

Definition 3.5. Let $\tau \in S_n$ and A be a set of finite distances in τ. If d divides every distance in A then we call d a *common distance divisor in* τ of the set A.

Observe that saying that a set has a common distance divisor implies that there are no infinite distances in the set.

Definition 3.6. Let $\sigma, \tau \in S_n$. If there exists distance d in τ that divides $d_\tau(\sigma(a), \sigma\tau^d(a))$ for all $a \in \{1, \ldots, n\}$, then we say that *distance d in τ is preserved by σ*. In other words, d is a common distance divisor of the set

$$\{d_\tau(\sigma(a), \sigma\tau^d(a)) \mid a \in \{1, \ldots, n\}\}.$$

Recall that saying there exists a distance d in τ means that d is less than the largest length of cycles of τ. Otherwise, distance preservation would be a completely uninteresting concept since if $m = |\tau|$ then $d_\tau(\sigma(a), \sigma\tau^m(a)) = 0$ for all a, which m divides.

Example 3.7. Let $\tau = (1\ 2\ 3\ 4\ 5\ 6)$ and $\sigma = (1\ 2\ 4\ 5)$ in S_6. Then

$$d_\tau(\sigma(1), \sigma\tau^3(1)) = d_\tau(2, 5) = 3$$
$$d_\tau(\sigma(2), \sigma\tau^3(2)) = d_\tau(4, 1) = 3$$
$$d_\tau(\sigma(3), \sigma\tau^3(3)) = d_\tau(3, 6) = 3$$
$$d_\tau(\sigma(4), \sigma\tau^3(4)) = d_\tau(5, 2) = 3$$
$$d_\tau(\sigma(5), \sigma\tau^3(5)) = d_\tau(1, 4) = 3$$
$$d_\tau(\sigma(6), \sigma\tau^3(6)) = d_\tau(6, 3) = 3$$

Thus 3 divides $d_\tau(\sigma(a), \sigma\tau^3(a))$ for all $a \in \{1, \ldots, 6\}$. Then distance 3 in τ is preserved by σ.

Example 3.8. Let τ be any permutation in S_n. Then for all distances d in τ and $a \in \{1, \ldots, n\}$, $d_\tau(\tau(a), \tau\tau^d(a)) = d$. Thus distance d in τ is preserved by τ.

The reason we use the phrase *preserved by* σ is that the distance in τ between a and $\tau^d(a)$ is precisely d (modulo the length of the cycle of a). Then if σ moves integers of distance d in τ such that the resultant distance is divisible by d, it makes sense to talk about some notion of distance preservation by σ.

Proposition 3.9. *Let $\sigma, \tau \in S_n$. If there exists distance $d > 0$ such that $d_\tau(\sigma(a), \sigma\tau^d(a)) = d$ for all $a \in \{1, \ldots, n\}$ and d is relatively prime to $|\tau|$, then $\langle \sigma, \tau \rangle \neq S_n$. In particular, $\langle \sigma, \tau \rangle$ is a commutative subgroup.*

Proof. Let $m = |\tau|$ and suppose there exists some distance $d > 0$ such that $d_\tau(\sigma(a), \sigma\tau^d(a)) = d$ for all $a \in \{1, \ldots, n\}$ and d is relatively prime to $|\tau| = m$. Then by definition, there

exists $r \in \mathbb{Z}$ such that $dr \equiv 1 \pmod{m}$ and so $\tau^{dr} = \tau$. Thus for all $a \in \{1, \ldots, n\}$,

$$\begin{aligned}
& \tau^d \sigma(a) && = \sigma \tau^d(a) \\
\Rightarrow \quad & \tau^d \sigma && = \sigma \tau^d \\
\Rightarrow \quad & \tau^{2d} \sigma && = \tau^d \sigma \tau^d \\
\Rightarrow \quad & \tau^{2d} \sigma && = \sigma \tau^d \tau^d = \sigma \tau^{2d} \\
& && \vdots \\
\Rightarrow \quad & \tau^{rd} \sigma && = \sigma \tau^{rd} \quad \text{for all } r \in \mathbb{N}.
\end{aligned}$$

But $\tau^{rd} = \tau$ for some $r \in \mathbb{Z}$, so $\tau\sigma = \sigma\tau$. Therefore σ and τ commute. Then the group generated by σ and τ is a commutative group. But S_n is not commutative, therefore $\langle \sigma, \tau \rangle \neq S_n$. □

Corollary 3.10. *Let $\sigma, \tau \in S_n$. The group $\langle \sigma, \tau \rangle$ is commutative if and only if there exists distance $d > 0$ such that $d_\tau(\sigma(a), \sigma\tau^d(a)) = d$ for all $a \in \{1, \ldots, n\}$ and d is relatively prime to $|\tau|$.*

Proof. If such a distance exists as in the hypothesis, then the previous proposition shows that $\langle \sigma, \tau \rangle$ is commutative. If $\langle \sigma, \tau \rangle$ is commutative, then σ and τ commute. Suppose $\tau(a) = a$ for some $a \in \{1, \ldots, n\}$. Since $\langle \sigma, \tau \rangle$ is transitive, we require

$\sigma(b) = a$ for some $b \neq a$. Then

$$\begin{aligned} & \tau(a) = a \\ \Rightarrow \quad & \tau\sigma(b) = a \\ \Rightarrow \quad & \sigma\tau(b) = a \\ \Rightarrow \quad & \tau(b) = \sigma^{-1}(a) \\ \Rightarrow \quad & \tau(b) = b \end{aligned}$$

Again, since $\langle \sigma, \tau \rangle$ is transitive, we require $\sigma(c) = b$ for some $c \neq b$. Moreover, if $c = a$ then $(a\ b)$ would be a cycle of σ which would make $\langle \sigma, \tau \rangle$ intranstive since τ fixes both a and b, which is a contradiction of our implied assumption that $\langle \sigma, \tau \rangle$ is transitive. Therefore $c \notin \{b, a\}$ and so similarly $\tau(c) = c$.

Continuing in this fashion we see that τ fixes every integer in $\{1, \ldots, n\}$, thus $\tau = id$ which is a contradiction by our implied assumption that $\tau \neq id$. Therefore every cycle of τ is of length greater than 1. Then there exists $d > 0$ such that d is less than the length of every cycle of τ, so that $\tau^d(a) \neq a$ for all $a \in \{1, \ldots, n\}$. Since $\langle \sigma, \tau \rangle$ is commutative, $\tau^d\sigma(a) = \sigma\tau^d(a)$ for all $a \in \{1, \ldots, n\}$. Therefore $d_\tau(\sigma(a), \sigma\tau^d(a)) = d$ for all $a \in \{1, \ldots, n\}$. In particular, when $d = 1$, then d is relatively prime to $|\tau|$. $\square$

Example 3.11. If $\sigma \in \langle \tau \rangle$, then σ and τ commute. Moreover, $\sigma = \tau^r$ for some $r \in \mathbb{Z}$. Let $a \in \{1, \ldots, n\}$ and k be the length of the cycle in τ containing a. Then

$$\begin{aligned} d_\tau(\sigma(a), \sigma\tau^d(a)) &= d_\tau(\tau^r(a), \tau^r\tau^d(a)) \\ &\equiv r + d - r \pmod{k} \\ &\equiv d \pmod{k} \end{aligned}$$

for any $d \in \mathbb{Z}$. In particular, for any $d > 0$ less than the length of every cycle of τ, $d_\tau(\sigma(a), \sigma\tau^d(a)) = d$ for all $a \in \{1, \ldots, n\}$.

A nice result, then, of using distances is that we have completely classified all commutative subgroups of S_n that are generated by two permutations. However, in the previous lemmas, we required the distances to all be equal. We can extend our results by focusing on distance divisors.

Lemma 3.12. *Let $\tau, \sigma \in S_n$ and $k_1, k_2, \ldots, k_j$ be the non-singular lengths of the cycles of τ. If there exists distance $d > 1$ such that distance d in τ is preserved by σ and $g = \gcd(d, k_1, k_2, \ldots, k_j) > 1$, then g divides $d_\tau(\sigma(a), \sigma\tau^{dr}(a))$ for all $r \in \mathbb{Z}$ and $a \in \{1, \ldots, n\}$. In other words, g is a common distance divisor of the set*

$$\{d_\tau(\sigma(a), \sigma\tau^{dr}(a)) \mid a \in \{1, \ldots, n\}, r \in \mathbb{Z}\}.$$

In particular, if $g = \gcd(d, k_1, \ldots, k_j)$ and g divides $d_\tau(\sigma(a), \sigma\tau^d(a))$ for all $a \in \{1, \ldots, n\}$ for some distance $d > 1$ in τ, then g divides $d_\tau(\sigma(a), \sigma\tau^{dr}(a))$ for all $r \in \mathbb{Z}$.

Proof. Let d and g be as in the hypothesis. Then g divides $d_\tau(\sigma(a), \sigma\tau^d(a))$ for all $a \in \{1, \ldots, n\}$.

(Induction on r.) In the case that $r = 0$,

$$d_\tau(\sigma(a), \sigma\tau^{dr}(a)) = d_\tau(\sigma(a), \sigma\tau^0(a)) = d_\tau(\sigma(a), \sigma(a)) = 0$$

which g clearly divides. If $r = 1$ then $d_\tau(\sigma(a), \sigma\tau^{dr}(a)) = d_\tau(\sigma(a), \sigma\tau^d(a))$ which is divisible by g.

Suppose g divides $d_\tau(\sigma(a), \sigma\tau^{dr}(a))$ up to some $r \in \mathbb{N}$ and let $b = \tau^{dr}(a)$. Note that $\sigma(a)$ and $\sigma\tau^{dr}(a)$ are in the same cycle of τ since g divides the distance between them (hence finite). Moreover, g divides $d_\tau(\sigma(b), \sigma\tau^{d}(b))$ by hypothesis, that is g divides $d_\tau(\sigma\tau^{dr}(a), \sigma\tau^{d}\tau^{dr}(a)) = d_\tau(\sigma\tau^{dr}(a), \sigma\tau^{d(r+1)}(a))$. So $\sigma(a), \sigma\tau^{dr}(a)$ and $\sigma\tau^{d(r+1)}$ are in the same cycle of τ. Let k be the length of this cycle. Then for some $m_1, m_2 \in \mathbb{Z}$,

$$\begin{aligned} d_\tau(\sigma(a), \sigma\tau^{d(r+1)}(a)) &\equiv d_\tau(\sigma(a), \sigma\tau^{dr}(a)) + d_\tau(\sigma\tau^{dr}(a), \sigma\tau^{d(r+1)}(a)) \pmod{k} \\ &\equiv gm_1 + gm_2 \pmod{k} \\ &\equiv g(m_1 + m_2) \pmod{k} \end{aligned}$$

If $k = 1$ then $d_\tau(\sigma(a), \sigma\tau^{d(r+1)}(a)) = 0$ which is divisible by g. Otherwise, g divides k and so g divides $d_\tau(\sigma(a), \sigma\tau^{d(r+1)}(a))$.

For negative values of r, letting $b = \tau^{dr}(a)$ we have that $\tau^{-dr}(b) = a$ and $\sigma(b), \sigma\tau^{-dr}(b)$ are in the same cycle since $-dr > 0$ and we already showed that this is finite for positive multiples of d. Then

$$d_\tau(\sigma(a), \sigma\tau^{dr}(a)) = d_\tau(\sigma(\tau^{-dr}(b)), \sigma(b)) \equiv -d_\tau(\sigma(b), \sigma\tau^{-dr}(b)) \pmod{k}$$

where k is the length of the cycle in τ containing $\sigma(b)$. Thus since $-r > 0$, the case where r is negative reduces to the case for positive values. Therefore g divides $d_\tau(\sigma(a), \sigma\tau^{dr}(a))$ for all $r \in \mathbb{Z}$. □

Proposition 3.13. *Let $\tau, \sigma \in S_n$ and $k_1, k_2, \ldots, k_j$ be the non-singular lengths of the cycles of τ. Suppose there exists distance $d > 1$ such that distance d in τ is preserved by σ and $\gcd(d, k_1, k_2, \ldots, k_j) > 1$. Then the group generated by σ and τ is not all of S_n. In other words, if there exists distance $d > 1$ such that d is a common distance*

divisor of the set

$$\{d_\tau(\sigma(a), \sigma\tau^d(a)) \mid a \in \{1, \ldots, n\}\}$$

and the greatest common divisor of d *and the non-singular lengths of the cycles of* τ *is greater than* 1*, then* $\langle \sigma, \tau \rangle \neq S_n$*.*

Proof. Let $\sigma, \tau \in S_n$ and d be as in the hypothesis. Let $g = \gcd(d, k_1, k_2, \ldots, k_j) > 1$. Then g divides d and g divides the non-singular lengths of the cycles of τ. Note that since $\tau \neq id$, there exists some cycle of τ with length greater than 1. Consider an element $\rho \in \langle \sigma, \tau \rangle$ such that for $\alpha_i \in \{\sigma, \tau\}$,

$$\rho = \alpha_1 \alpha_2 \cdots \alpha_m$$

(Induction on m.) If $\rho = \alpha_1 = \tau$ then for $a \in \{1, \ldots, n\}$, letting k be the length of the cycle in τ containing a,

$$\begin{aligned} d_\tau(\rho(a), \rho\tau^d(a)) &= d_\tau(\tau(a), \tau^{d+1}(a)) \\ &\equiv (d+1) - 1 \pmod{k} \quad \text{(Lemma 3.4)} \\ &\equiv d \pmod{k} \end{aligned}$$

which is either 0 if $k = 1$ or is divisible by g since g divides both d and k. Either way, g divides $d_\tau(\alpha_1(a), \alpha_1\tau^d(a))$.

If $\rho = \alpha_1 = \sigma$, then for $a \in \{1, \ldots, n\}$, g divides d and so $d_\tau(\sigma(a), \sigma\tau^d(a))$ is divisible by g. Thus for $m = 1$ we have that g divides $d_\tau(\rho(a), \rho\tau^d(a))$ for all $a \in \{1, \ldots, n\}$.

Let $\rho = \alpha_1\alpha_2\cdots\alpha_{m+1}$ and $\beta = \alpha_1\alpha_2\cdots\alpha_m$ and suppose g divides $d_\tau(\beta(a), \beta\tau^d(a))$ for all $a \in \{1, \ldots, n\}$. If $\alpha_{m+1} = \tau$ then

$$\begin{aligned} d_\tau(\rho(a), \rho\tau^d(a)) &= d_\tau(\beta\tau(a), \beta\tau\tau^d(a)) \\ &= d_\tau(\beta(\tau(a)), \beta\tau^d(\tau(a))) \end{aligned}$$

which is divisible by g by the induction hypothesis.

Now suppose $\alpha_{m+1} = \sigma$. Since d divides $d_\tau(\sigma(a), \sigma\tau^d(a))$, there exists $r \in \mathbb{Z}$ such that $\tau^{dr}\sigma(a) = \sigma\tau^d(a)$. Then

$$\begin{aligned} d_\tau(\rho(a), \rho\tau^d(a)) &= d_\tau(\beta\sigma(a), \beta\sigma\tau^d(a)) \\ &= d_\tau(\beta(\sigma(a)), \beta\tau^{dr}(\sigma(a)) \end{aligned}$$

By Lemma 3.12, g divides $d_\tau(\beta(\sigma(a)), \beta\tau^{dr}(\sigma(a))$. Therefore, by induction, g divides $d_\tau(\rho(a), \rho\tau^d(a))$ for every $\rho \in \langle\sigma, \tau\rangle$.

Since d is a distance in τ, d must be less than the length of some cycle of τ, and since $d > 1$ then there is a cycle of length k such that $1 < d < k$. Then there exists $a \in \{1, \ldots, n\}$ such that a is in the cycle of length k and $\tau(a) = b \neq a$. Let $c \in \{1, \ldots, n\}$ such that $\tau^d(a) = c$. Consider the permutation $\gamma = (a\ b) \in S_n$. Then

$$d_\tau(\gamma(a), \gamma\tau^d(a)) = d_\tau(b, \gamma(c)).$$

Case $\gamma(c) = b$: In this case, $c = a$ and so $\tau^d(a) = a$. Then d is a multiple of the length of the cycle containing a and b, which is a contradiction by choice of k.

Case $\gamma(c) = a$:

$$\begin{aligned} d_\tau(\gamma(a), \gamma\tau^d(a)) &= d_\tau(b, \gamma(c)) \\ &= d_\tau(b, a) \\ &\equiv -d_\tau(a, b) \pmod{k} \\ &\equiv -1 \pmod{k} \end{aligned}$$

which is not divisible by g since g and k are not relatively prime. Therefore $\gamma \notin \langle \sigma, \tau \rangle$.

Case $\gamma(c) = c$: Then $c \neq a$ and $c \neq b$. Thus

$$\begin{aligned} d_\tau(\gamma(a), \gamma\tau^d(a)) &= d_\tau(b, \gamma(c)) \\ &= d_\tau(b, c) \\ &= d_\tau(\tau(a), \tau^d(a)) \\ &\equiv d - 1 \pmod{k} \end{aligned}$$

which is not divisible by g since g divides d and k. Therefore $\gamma \notin \langle \sigma, \tau \rangle$.

Therefore, since γ cannot be in the group generated by σ and τ, then $\langle \sigma, \tau \rangle \neq S_n$. □

Example 3.14. From the Example 3.7, distance 3 is preserved in (1 2 3 4 5 6) by (1 2 4 5). Thus $\langle (1\ 2\ 3\ 4\ 5\ 6), (1\ 2\ 4\ 5) \rangle \neq S_n$.

Corollary 3.15. *Let $\sigma, \tau \in S_n$. If there exists distance $d > 1$ in τ such that d divides $d_\tau(a, \sigma(a))$ for all $a \in \{1, \ldots, n\}$ and $\gcd(d, n) > 1$, then $\langle \sigma, \tau \rangle \neq S_n$.*

Proof. Let d be as in the hypothesis and let $g = \gcd(d, n) > 1$. Since $d_\tau(a, \sigma(a))$ is finite, then $a, \sigma(a), \sigma^2(a), \cdots$ are all in the same cycle of τ. Then in order for $\langle \sigma, \tau \rangle$

to be transitive, τ must be an n-cycle, so that is there is only one cycle of τ of length n.

Using the triangle equality and the fact that everything is in the same cycle in τ,

$$d_\tau(\sigma(a), \sigma\tau^g(a)) \equiv d_\tau(\sigma(a), a) + d_\tau(a, \tau^g(a)) + d_\tau(\tau^g(a), \sigma\tau^g(a)) \pmod{n}$$

Since g divides each of the three terms on the right hand side of the equality, g divides $d_\tau(\sigma(a), \sigma\tau^g(a))$ since g divides n. Then $g > 1$ is a common distance divisor of the set

$$\{d_\tau(\sigma(a), \sigma\tau^g(a)) \mid a \in \{1, \ldots, n\}\}.$$

Moreover, $d < n$ since all distances in τ must be less than the length of the cycles of τ, in this case length n. Then $1 < g < n$ and so g is not a multiple of n. Therefore, by the previous proposition, $\langle \sigma, \tau \rangle \neq S_n$. □

Example 3.16. Letting $\tau = (1\ 2\ 3\ 4\ 5\ 6)$ and $\sigma = (1\ 3)$ in S_6,

$$\begin{aligned} d_\tau(1, \sigma(1)) &= d_\tau(1, 3) = 2 \\ d_\tau(3, \sigma(3)) &= d_\tau(3, 1) = 4 \\ d_\tau(a, \sigma(a)) &= d_\tau(a, a) = 0 \quad \forall\ a \in \{2, 4, 5, 6\} \end{aligned}$$

Then 2 divides $d_\tau(a, \sigma(a))$ for all $a \in \{1, \ldots, n\}$ and so $\langle \sigma, \tau \rangle \neq S_n$.

The requirement that the greatest common divisor of the preserved distance and the lengths of the cycles be greater than 1 is necessary.

Example 3.17. Let $\tau = (1\ 3\ 2\ 5\ 4\ 7\ 4)$ and $\sigma = (3\ 4)(5\ 6\ 7)$. Then distance 2 is preserved in τ by σ but $\langle \sigma, \tau \rangle = S_n$ and clearly 2 is relatively prime to 7, the length of the cycle in τ.

It should be noted that a pair of permutations which satisfy Proposition 3.13 do not necessarily satisfy Corollary 3.15. In Example 3.14, 3 is preserved by σ but $d_\tau(1, \sigma(1)) = d_\tau(1, 2) = 1$, so the corollary does not hold. Regardless, σ and τ can be viewed as preserving a set of integers of distance d or g apart. The next section will describe this relation of set preservation.

In our description of distance preservation, we required all the distances to be finite. Nothing about the proofs of the lemmas and propositions in this section relied heavily on this requirement. Some evidence suggests the following conjecture.

Conjecture 3.18. *Let $\sigma, \tau \in S_n$ and $k_1, \ldots, k_m$ be the lengths of the non-singular lengths of the cycles of τ. If there exists a distance $d > 1$ in τ such that $d_\tau(\sigma(a), \sigma\tau^d(a))$ is infinite or divisible by d for all $a \in \{1, \ldots, n\}$ and $\gcd(d, k_1, \ldots, k_m) > 1$, then $\langle \sigma, \tau \rangle \neq S_n$.*

3.3 On Imprimitive Subgroups

The symmetric group as a whole does not preserve any significant property of groups by the observation that the symmetric group contains every possible permutation. Thus finding any property of a subgroup which is preserved is sufficient to distinguish the subgroup from the entire symmetric group. Intransitive groups, for example, preserve the non-trivial orbits of the group action, and orbits are nothing more than a partition of the set $\{1, \ldots, n\}$. We will then define what it means for permutations to preserve a partition and relate this preservation to distances.

Definition 3.19. A transitive permutation group G acting on a set X is called *primitive* if G does not preserve any nontrivial partition of X. If G does preserve a nontrivial partition of X then G is called *imprimitive*.

A nontrivial partition is a partition that is not the set of all singletons nor the entire set.

Example 3.20. Let $\sigma = (1\ 2\ 3\ 4) \in S_4$. Then $\sigma(\{1,3\}) = \{2,4\}$ and $\sigma(\{2,4\}) = \{1,3\}$. Thus $\langle\sigma\rangle$ preserves the partition $\{\{1,3\},\{2,4\}\}$ of $\{1,2,3,4\}$.

Lemma 3.21. *Every primitive group is transitive, but not every transitive group is primitive.*

Proof. By definition, primitive groups are transitive. However, the group generated by $(1\ 2\ 3\ 4) \in S_4$ is clearly transitive but imprimitive by Example 3.20. □

Note that S_n is clearly a primitive subgroup since given any nontrivial partition of $\{1,\ldots,n\}$ we can always construct a transposition which does not preserve the partition by simply sending one integer of a nonsingleton part to an integer of another part, thereby fixing every other integer. Then imprimitivity is a sufficient property to distinguish proper subgroups of the symmetric group. However, the number of ways to partition a set of n elements, known as Bell numbers and denoted B_n, is difficult to count and has order of growth faster than any exponential [Jak11], that is

$$\lim_{n\to\infty} \frac{B_n}{a^n} = \infty \quad (a > 0).$$

Thus it would seem rather time consuming to check every partition against the group generated by two permutations. The previous section on distances describes a portion of the imprimitive subgroups of S_n generated by two elements.

Lemma 3.22. *Let $\tau \in S_n$ and $d \in \mathbb{N}$. The set*

$$X = \{X_1, X_2, \ldots, X_n\}; \quad X_i = \{i, \tau^d(i), \tau^{2d}(i), \ldots\}$$

has the property that for each $X_i, X_j \in X$, $X_i \cap X_j = \emptyset$ or $X_i = X_j$ and $\bigcup X_i = X$. In other words, after removing duplicates, X is a partition on the set $\{1, \ldots, n\}$.

Proof. Let $\tau \in S_n$ and $d \in \mathbb{N}$. Consider $i \in X_i$ and let k be the length of the cycle in τ containing i. For all $r \in \mathbb{Z}$, $\tau^{rd}(i) = \tau^{rd+sk}(i)$ for all $s \in \mathbb{Z}$ since $\tau^s k(i) = i$. Then we can take s large enough so that $rd + skd > 0$. Then $r + sk > 0$ and

$$\tau^{rd}(i) = \tau^{rd+skd}(i) = \tau^{d(r+sk)}(i) \in X_i$$

Thus $\tau^{rd}(i) \in X_i$ for all $r \in \mathbb{Z}$.

Note that each X_i is a subset of a single cycle of τ. Clearly, $\bigcup X_i = \{1, \ldots, n\}$ since for all $i \in \{1, \ldots, n\}$, $i \in X_i$. It remains to show that the X_i are pairwise disjoint or equal. Then consider $X_i, X_j \in X$ and suppose $X_i \cap X_j \neq \emptyset$. If $a \in X_i \cap X_j$ then $a = \tau^{dr}(i) = \tau^{ds}(j)$ for some $r, s \in \mathbb{N}$. Then $\tau^{d(r-s)}(i) = j$ and $\tau^{d(s-r)}(j) = i$. Therefore $j \in X_i$ and $i \in X_j$, which means that $X_i = X_j$ since X_i and X_j are *generated* by i and j respectively. Therefore X forms a partition of $\{1, \ldots, n\}$. □

Proposition 3.23. *Let $\sigma, \tau \in S_n$ and $k_1, \ldots, k_m$ be the lengths of cycles of τ. If there exists a distance $d > 1$ which is preserved in τ by σ such that $\gcd(d, k_i) = g$ for each $1 \leq i \leq m$, then $\langle \sigma, \tau \rangle$ is imprimitive.*

Proof. Assume there exists a distance $d > 1$ which is preserved in τ by σ and $g = \gcd(d, k_i) > 1$ for each cycle length k_i in τ. Note that $\gcd(d, k_1, \ldots, k_m) = g$. Let

$X = \{X_1, X_2, \ldots, X_n\}$ be the partition of $\{1, \ldots, n\}$ given in the previous lemma for $d \in \mathbb{N}$. Then

$$\begin{aligned}\tau(X_i) &= \{\tau(i), \tau(\tau^d(i)), \tau(\tau^{2d}(i)), \ldots\} \\ &= \{\tau(i), \tau^d(\tau(i)), \tau^{2d}(\tau(i)), \ldots\} \\ &= X_{\tau(i)}\end{aligned}$$

Thus the partition X is preserved by τ.

Let $x \in \sigma(X_i)$ and k_i denote the length of the cycle in τ that contains i. Then $x = \sigma\tau^{jd}(i)$ for some $j \in \mathbb{N}$. By Lemma 3.12, g divides $d_\tau(\sigma(i), \sigma\tau^{jd}(i))$ and thus there exists $r_j \in \mathbb{N}$ such that $\tau^{gr_j}\sigma(i) = \sigma\tau^{jd}(i)$. Since $\gcd(d, k_{\sigma(i)}) = g$, there exists $p, q \in \mathbb{Z}$ such that $dp + k_{\sigma(i)}q = g$ and so there exists $p, q \in \mathbb{Z}$ such that $dp + k_{\sigma(i)}q = gr_j$. Then

$$\sigma\tau^{jd}(i) = \tau^{gr_j}\sigma(i) = \tau^{dp}\sigma(i) \in X_{\sigma(i)}$$

Therefore $\sigma(X_i) \subseteq X_{\sigma(i)}$ and thus, since σ is a bijection, $|X_i| \leq |X_{\sigma(i)}|$. Similarly, since i was taken arbitrarily, $\sigma(X_{\sigma(i)}) \subseteq X_{\sigma^2(i)}$ and thus $|X_{\sigma(i)}| \leq |X_{\sigma^2(i)}|$. But $\sigma^p(i) = i$ for some $p \in \mathbb{N}$, and so continuing this process we get the chain of inequalities:

$$|X_i| \leq |X_{\sigma(i)}| \leq |X_{\sigma^2(i)}| \leq \cdots \leq |X_{\sigma^{p-1}(i)}| \leq |X_i|$$

Therefore, $|X_i| = |X_{\sigma(i)}|$ and so $\sigma(X_i) = X_{\sigma(i)}$. Thus the partition X is preserved by σ.

Therefore, since both σ and τ preserve the partition X, for all $\rho \in \langle \sigma, \tau \rangle$, ρ preserves X. It remains to show that the partition is non-trivial.

If $X = \{\{1, \ldots, n\}\}$, then for $a \in \{1, \ldots, n\}$, $\tau^{rd}(a) = \tau(a)$ for some $r \in \mathbb{Z}$ by construction of the partition X. Then $d_\tau(a, \tau^{rd}(a)) = d_\tau(a, \tau(a)) = 1$, which d does not divide. This contradicts Lemma 3.12. If $X = \{\{1\}, \{2\}, \ldots\}$, then $\tau^d(i) = i$ for all $i \in \{1, \ldots, n\}$. Thus d is a multiple of $|\tau|$, which is a contradiction since d by definition as a distance must be less than the largest length of the cycles of τ. Therefore the partition X is nontrivial and thus G is imprimitive. $\square$

Remark. Notice that in the previous proof that the sizes of the sets in the partition of X are equal so long as $\langle \sigma, \tau \rangle$ is transitive. This is a requirement for a group to be imprimitive as we will see.

We've shown that some groups generated by two permutations which preserve some non-trivial distance is an imprimitive group, but not every imprimitive group preserves some non-trivial distance. For example, letting $\tau = (1\ 2\ 3\ 4)(5\ 6\ 7\ 8)$ and $\sigma = (2\ 4\ 6\ 8)$, we see that σ and τ preserve the partition $\{\{1, 3, 5, 7\}, \{2, 4, 6, 8\}\}$ and $\langle \sigma, \tau \rangle$ is clearly transitive. The only non-trivial distance which could be preserved is 2 since 2 is the only common divisor of the lengths of the cycles in τ. But $d_\tau(\sigma(4), \sigma\tau^2(4)) = \infty$, so unless Conjecture 3.18 holds the result will not be clear. However, if we force τ to be an n-cycle, then we get the following proposition.

Proposition 3.24. *Let $\sigma, \tau \in S_n$ such that τ is an n-cycle and let $G = \langle \sigma, \tau \rangle$. If G is imprimitive then there exists distance $d > 1$ which is preserved in τ by σ.*

Proof. Let $\langle \sigma, \tau \rangle = G$ be imprimitive which preserves the nontrivial partion $X = \{X_1, X_2, \ldots, X_m\}$. Since G is transitive and generated by σ and τ, for every $X_i, X_j \in$

X there exists $\rho \in \langle \sigma, \tau \rangle$ such that $\rho(X_i) = X_j$. Since permutations are bijections, the size of each X_i must be equal. Then $|X_i|$ divides n. Letting $d = n/|X_i| > 1$ we see that $|X| = d$ since $\bigcup X_i = \{1, \ldots, n\}$ and $|X_i| = |X_j|$ for all i, j and X_i, X_j are pairwise disjoint.

Since τ acts on the partition X and is an n-cycle, τ can be seen as a d-cycle in S_X, so that $\tau^d(X_i) = X_i$ for all $X_i \in X$ and is the smallest such positive integer with this property on X. Let $i \in X_i$. Consider the set

$$\mathcal{A} = \{i, \tau^d(i), \tau^{2d}(i), \ldots, \tau^n(i) = i\}.$$

Since τ is an n-cycle, then $\tau^r(i) \neq i$ for all $r < n$. Therefore $|\mathcal{A}| = n/d$. Also, $\tau^d(a) \in X_i$ for all $a \in X_i$, hence $\tau^{rd}(a) \in X_i$ for all $r \in \mathbb{Z}$. Therefore, $\mathcal{A} \subseteq X_i$. But $|X_i| = n/d = |\mathcal{A}|$, and so $\mathcal{A} = X_i$. Therefore, for all $i \in \{1, \ldots, n\}$, letting $X_i \in X$ such that $i \in X_i$,

$$X_i = \{i, \tau^d(i), \tau^{2d}(i), \ldots, \tau^n(i) = i\}.$$

Thus for all $a, b \in X_i$, $i = \tau^{rd}(a)$ and $i = \tau^{sd}(b)$ for some $r, s \in \mathbb{N}$ such that $rd, sd < n$. Without loss of generality, suppose $r > s$. Then $0 \leq dr - ds < n$ and so $\tau^{rd-sd}(a) = b$ implies that $d_\tau(a, b)$ is divisible by d.

Let $a \in \{1, \ldots, n\}$ and $X_a \in X$ such that $a \in X_a$. Then $\tau^d(a) \in X_a$ as well. Let $X_b = \sigma(X_a)$. Then $\sigma(a)$ and $\sigma(\tau^d(a))$ are both in X_b. Thus d divides $d_\tau(\sigma(a), \sigma\tau^d(a))$ for all $a \in \{1, \ldots, n\}$. Therefore distance $d > 1$ is preserved in τ by σ. □

Corollary 3.25. *If G is an imprimitive subgroup of S_n which preserves a partition $X = \{X_1, X_2, \ldots, X_m\}$, then $|X_i| = |X_j|$ for all $X_i, X_j \in X$.*

Proof. From the proof of the previous proposition, since G is imprimitive, and thus transitive, there exists $\rho \in G$ such that $\rho(X_i) = X_j$ for every pairing $X_i, X_j \in X$. Since ρ is a bijection, we get that $|X_i| = |X_j|$ for all $X_i, X_j \in X$. □

Corollary 3.26. *There are no imprimitive subgroups of S_p for p prime.*

Proof. If G is an imprimitive subgroup of S_p, then the partition that G preserves has elements whose size divides p, from Corollary 3.25. Since p is prime, the size of the elements of the partition are either 1 or p. Then the elements of the partition are precisely either singletons or the entire set $\{1, 2, \ldots, p\}$, neither of which is nontrivial. Thus G cannot be imprimitive. □

Corollary 3.27. *Let $\sigma, \tau \in S_n$ such that τ is an n-cycle and let $G = \langle \sigma, \tau \rangle$. Then G is imprimitive if and only if there exists distance $d > 1$ which is preserved in τ by σ such that* $\gcd(d, n) = d$.

Proof. If G is imprimitive, then by the proof of Proposition 3.24, G has distance $d > 1$, which is preserved in τ by σ and d divides n. Thus $\gcd(d, n) = d$. If distance $d > 1$ is preserved in τ by σ such that $\gcd(d, n) = d$, then since τ is an n-cycle, Proposition 3.23 shows that $\langle \sigma, \tau \rangle = G$ is imprimitive. □

Unfortunately, distance preservation does not describe all imprimitive subgroups.

Example 3.28. Let $\tau = (1\ 2\ 3\ 4)(5\ 6\ 7\ 8)$ and $\sigma = (1\ 3\ 5\ 7)(2\ 4\ 6\ 8)$ in S_8. Then σ and τ both preserve the partition $\{\{1, 3, 5, 7\}, \{2, 4, 6, 8\}\}$, so that $\langle \sigma, \tau \rangle$ is imprimitive. The only possible distance $d > 1$ which could be preserved in τ by σ is either 2 or 3 since distances must be less than the length of the cycles in τ. However,

$$d_\tau(\sigma(1), \sigma\tau^2(1)) = d_\tau(3, 5) = \infty$$

and

$$d_\tau(\sigma(1), \sigma\tau^3(1)) = d_\tau(3, 6) = \infty.$$

Example 3.29. Let $\tau = (1\ 2\ 3\ 4)(5\ 6)$ and $\sigma = (1\ 5\ 3\ 6)(2\ 4)$ in S_6. Then σ and τ both preserve the partition $\{\{1,3\},\{2,4\},\{5,6\}\}$, so that $\langle\sigma,\tau\rangle$ is imprimitive. If $d > 1$ and is preserved in τ by σ, then d is either 2 or 3. However,

$$d_\tau(\sigma(1), \sigma\tau^2(1)) = d_\tau(5, 6) = 1$$

and

$$d_\tau(\sigma(5), \sigma\tau^3(5)) = d_\tau(3, 1) = 2.$$

CHAPTER IV

PAIRS INVOLVING A TRANSPOSITION

Using the combined results of the previous chapters we can completely determine whether a pair of permutations, one of which is a transposition, will generate the symmetric group or not.

Proposition 4.1. *Let $\sigma \in S_n$ be a transposition with cycle decomposition $(p\ q)$ and $\tau \in S_n$ be a n-cycle. If $d_\tau(p,q)$ is relatively prime to n, then $\langle \sigma, \tau \rangle = S_n$.*

Proof. Let $d = d_\tau(p,q)$ and suppose d is relatively prime to n. Then there exists $r \in \mathbb{N}$ such that $dr \equiv 1 \pmod{n}$. We can write $\sigma = (a_1\ a_{1+d})$ and $\tau = (a_1\ a_2\ \ldots\ a_n)$ where $a_1 = p$ and $a_{1+d} = q$. Then it will suffice to show that $(a_1, a_2) \in \langle \sigma, \tau \rangle$ since it is a matter of relabelling the pair $(1\ 2)$ and $(1\ 2\ \ldots\ n)$ which we know generates S_n. We will track the indices of the a_i and we mean a_{x+y} to mean $a_{x+y \pmod{n}}$, which makes sense since the a_i are in the n-cycle τ.

Consider conjugation of σ by powers of τ. Then for any $t \in \mathbb{Z}$,

$$\tau^t \sigma \tau^{-t} = (\tau^t(a_1)\ \tau^t(a_{1+d})) = (a_{1+t}\ a_{1+t+d}) \quad \text{(Lemma 1.3)}$$

and so $d_\tau(a_{1+t}, a_{1+t+d}) = d$. Thus, by conjugating, we get that the set

$$\mathcal{A} = \{(a_i\ a_j) \in S_n \mid j \equiv i + d \pmod{n}\}$$

is a subset of $\langle \sigma, \tau \rangle$. Then

$$\begin{aligned} & (a_1\ a_{1+d})(a_{1+2d}\ a_{1+d})(a_1\ a_{1+d}) &&= (a_1, a_{1+2d}) \in \langle \sigma, \tau \rangle \\ \Rightarrow\ & (a_1\ a_{1+2d})(a_{1+3d}\ a_{1+2d})(a_1\ a_{1+2d}) &&= (a_1\ a_{1+3d}) \in \langle \sigma, \tau \rangle \\ & &&\vdots \\ \Rightarrow\ & &&(a_1\ a_{1+td}) \in \langle \sigma, \tau \rangle \end{aligned}$$

for all $t \in \mathbb{N}$. But $rd \equiv 1 \pmod{n}$ for some $r \in \mathbb{N}$, so $(a_1\ a_{1+rd}) = (a_1\ a_{1+1}) = (a_1\ a_2) \in \langle \sigma, \tau \rangle$. Therefore we have shown the sufficient condition that $\langle \sigma, \tau \rangle = S_n$. □

Example 4.2. Consider the permutations $\sigma = (3\ 6)$ and $\tau = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8)$ where $d_\tau(3, 6) = 3$. Then $\langle \sigma, \tau \rangle = S_n$ since 3 and 8 are relatively prime.

Example 4.3. Consider S_p where p is prime. Then every pair of permutations where one is a transposition and the other is a p-cycle generates all of S_p.

Corollary 4.4. *For $n \geq 3$, let $\sigma, \tau \in S_n$ where $\sigma = (p\ q)$ is a transposition and τ an n-cycle. Then $\langle \sigma, \tau \rangle = S_n$ if and only if $d_\tau(p, q)$ is relatively prime to n.*

Proof. If $d_\tau(p, q)$ is relatively prime to n, then by Proposition 4.1, $\langle \sigma, \tau \rangle = S_n$. If $d_\tau(a, b)$ is not relatively prime to n, let $g = \gcd(n, d_\tau(a, b)) > 1$. Then $d_\tau(a, \sigma(a)) = 0$ for all $a \neq p, q$ and $d_\tau(p, q) \equiv -d_\tau(q, p) \pmod{n}$ and so g divides $d_\tau(a, \sigma(a))$ for all $a \in \{1, \ldots, n\}$. Therefore, by Proposition 3.15 we have that $\langle \sigma, \tau \rangle \neq S_n$. □

The only instances where a transposition σ and a permutation τ in S_n is transitive is if τ is composed of at most 2 cycles and σ connects the cycles of τ. Whether $\langle \sigma, \tau \rangle$ is all of S_n is a matter of the size of the cycles of $\langle \sigma, \tau \rangle$.

Proposition 4.5. *For $n \geq 3$, let $\sigma \in S_n$ be a transposition and $\tau \in S_n$ that is composed of exactly two cycles c_1 and c_2. Then $\langle \sigma, \tau \rangle = S_n$ if and only if $\langle \sigma, \tau \rangle$ is transitive and the lengths of the cycles c_1 and c_2 are relatively prime.*

Proof. Let $(p\ q)$ be the cycle decomposition of σ. If p and q lie in the same cycle, then $\langle \sigma, \tau \rangle$ is intransitive. Then let k_1 and k_2 be the lengths of the two cycles c_1 and c_2. We can write $\sigma = (a_1\ b_1)$ and $\tau = (a_1\ \ldots\ a_{k_1})(b_1\ \ldots\ b_{k_2})$ where $a_1 = p$ and $b_1 = q$. Consider the product of σ and τ:

$$\begin{aligned}\sigma\tau &= (a_1\ b_1)(a_1\ \ldots\ a_{k_1})(b_1\ \ldots\ b_{k_2}) \\ &= (a_1\ a_2\ a_3\ \ldots\ a_{k_1}\ b_1\ b_2\ b_3\ \ldots\ b_{k_2})\end{aligned}$$

which is an n-cycle. Let $\rho = \sigma\tau$. By observation, $d_\rho(a_1, b_1) = k_1$ and $d_\rho(b_1, a_1) = k_2$. Let d be the greatest common divisor of k_1 and k_2. Then there exists $r, s \in \mathbb{Z}$ such that $k_1 r + k_2 s = d$. But $k_1 + k_2 = n$ and so

$$\begin{aligned}d &= k_1 r - k_1 s + k_1 s + k_2 s \\ &= k_1(r - s) + s(k_1 + k_2) \\ &= k_1(r - s) + sn\end{aligned}$$

Thus d is also the greatest common divisor of $d_\rho(p, q)$ and n. Since $\langle \sigma, \tau \rangle = \langle \sigma, \sigma\tau \rangle = \langle \sigma, \rho \rangle$, then by Corollary 4.4, $\langle \sigma, \tau \rangle = S_n$ if and only if $d = 1$ if and only if the lengths of the cycles of τ are relatively prime. □

We have thus completely characterized pairs of permuations involving at least one transposition. The summary of these characterizations is provided in the following proposition.

Proposition 4.6. *Let $\sigma = (a, b) \in S_n$ be a transposition and $\tau \in S_n$. Then the group generated by σ and τ is all of S_n if and only if $\langle \sigma, \tau \rangle$ is transitive and one of the following is true:*

- *τ is an n-cycle and $d_\tau(a, b)$ is relatively prime to n;*
- *τ is composed of two cycles of relatively prime lengths.*

CHAPTER V

THE MULTIPLICATION TABLE

Using a mutliplication table for the symmetric group, we can look for patterns in the table and hopefully use the pattern to see if we can determine that through a series of multiplications that some permutations will be *unreachable*.

There is a natural lexicographical ordering of the permutations of the symmetric group using the arrangement described by the permutations. For example, in S_3 the permutations can be written as their mappings

$$id = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}; \quad (1\ 2) = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}; \quad (3\ 1\ 2) = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}.$$

Then 1 2 3 defines the identity mapping, 2 1 3 defines (1 2), and 2 3 1 defines (3 1 2). Using these arrangements, we can give a lexicographical ordering of the permutations so that for $\sigma, \tau \in S_n$ with arrangements $a_1\ a_2\ \cdots\ a_n$ and $b_1\ b_2\ \cdots\ b_n$, respectively, $\sigma < \tau$ if there exists $i \in \{1, \ldots, n\}$ such that $a_i < b_i$ and $a_j = b_j$ for all $j < i$, and $\sigma = \tau$ if $a_i = b_i$ for all i.

For example, for S_3 we have the following permutations in order:

$$\begin{array}{lll} 1\ 2\ 3 = id; & 1\ 3\ 2 = (2\ 3); & 2\ 1\ 3 = (1\ 2); \\ 2\ 3\ 1 = (1\ 2\ 3); & 3\ 1\ 2 = (1\ 3\ 2); & 3\ 2\ 1 = (1\ 3) \end{array}$$

We can then index the permutations from 1 to $n!$ in the lexicographical ordering so that $id = 1$ and so forth.

We create this ordering in order to implement a multiplication table. The multiplication table for S_3 is shown in Table 1.

Table 1. Mutiplication Table for S_3

$\circ$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	1	5	6	3	4
3	3	4	1	2	6	5
4	4	3	6	5	1	2
5	5	6	2	1	4	3
6	6	5	4	3	2	1

A few patterns become visible from observing this table. The top-left 2×2 block is actually S_2 and each 2×2 block below it is equivalent to S_2 by relabelling.

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \sim \begin{bmatrix} 3 & 4 \\ 4 & 3 \end{bmatrix} \sim \begin{bmatrix} 5 & 6 \\ 6 & 5 \end{bmatrix}$$

Furthermore, each 1×2 block in the multiplication table is a copy of one of the 1×2 blocks from the first column of 1×2 blocks.

$\circ$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	1	5	6	3	4
3	3	4	1	2	6	5
4	4	3	6	5	1	2
5	5	6	2	1	4	3
6	6	5	4	3	2	1

Also, the order of the 1×2 blocks from the 2×2 blocks of the first column maintain their order from top to bottom among each column of 1×2 blocks. Notice how the darker gray blocks always appear above the lighter gray in the table below, where each shade comes from the same 2×2 block.

$\circ$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	1	5	6	3	4
3	3	4	1	2	6	5
4	4	3	6	5	1	2
5	5	6	2	1	4	3
6	6	5	4	3	2	1

Observing each column of the 1×2 blocks, we see that it is merely a rearrangement of the first column of 1×2 blocks. More importantly, this rearrangement of blocks is not random. If we assign a value to each 2×2 block in sequential order so that each 1×2 block in the 2×2 block has the same value, we can rewrite the multiplication table focusing only on these blocks.

$\circ$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	1	5	6	3	4
3	3	4	1	2	6	5
4	4	3	6	5	1	2
5	5	6	2	1	4	3
6	6	5	4	3	2	1

$\sim$

$\circ$	1	2	3	4	5	6
1	1		2		3	
2	1		3		2	
3	2		1		3	
4	2		3		1	
5	3		1		2	
6	3		2		1	

Reading across the rows of the multiplication table, we see that the rows form exactly the permutations of S_3 in lexicographical order. This pattern can be observed for any n, that is for any symmetric group. Observe the multiplication table in Fig-

ure 2 for S_4, using shades of grey in place of values for visibility.

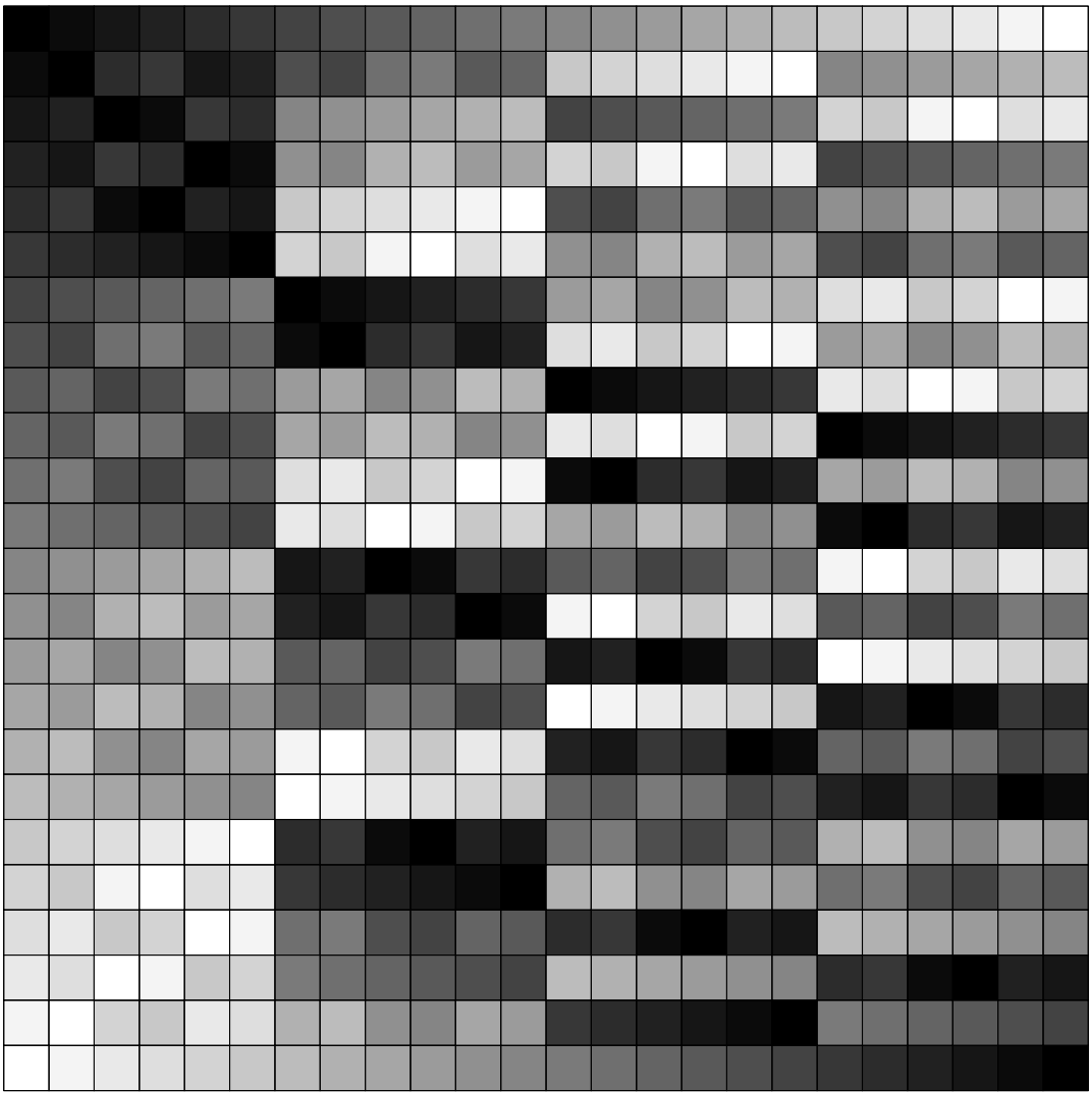

Figure 2. Representation of the Multiplcation Table for S_4

Using these observations, we can construct the multiplication table without ever actually multiplying the permutations. The algorithm to do so is presented below.

Algorithm 5.1 (Multiplication Table for the Symmetric Group)**.** *Constructing the multiplication table for S_n.*

Input: n, the size of the set that the symmetric group acts on

Output: the multiplication table for S_n

(1) If $n = 1$, return the 1×1 table consisting of 1; the table for S_1.

(2) Let M be a matrix of size $n! \times n$ and $M(i, j)$ denote the entry in the ith row and jth column. Each $M(i, j)$ is in itself a $1 \times (n-1)!$ block.

(3) For each $i \in \{1, \ldots, (n-1)!\}$:

  (a) Set $M(i, 1)$ to be the ith row of the multiplication table for S_{n-1}, which is itself a $1 \times (n-1)!$ block.

(4) For each k such that $(n-1)! < k < n!$:

  (a) Let $q, r \in \mathbb{N}$ be the unique values such that $k - 1 = q(n-1)! + r$ and $r < (n-1)!$.

  (b) Set $M(k, 1) = M(r+1, 1) + q(n-1)!$, where $M(i, j) + x$ is defined by adding x to each entry of the $1 \times (n-1)!$ block that is $M(i, j)$.

(5) For each pair $(i, j) \in \{1, \ldots, n!\} \times \{2, \ldots, n\}$:

  (a) Let p be the ith permutation in lexicographical order.

  (b) Let $a_1\ a_2\ \cdots\ a_n$ be the arrangement of p.

  (c) Let k be the number of times a_j appear in the jth slot of the first i permutations in lexicographical order.

(d) Set $M(i,j) = M((a_j - 1)(n-1)! + k, 1)$.

(6) Return M as a $n! \times n!$ table, where the ith row and jth column represent the ith permutaion multiplied on the left by the jth permutation, in lexicographical order.

Let us use the above algorithm to recreate the multiplication table for S_3. The first three steps are straightforward in that we set the top-left 2×2 block to the equivalent numbering of S_2:

1	2		
2	1		

For step 4, we use the division algorithm on each row number by dividing $k-1$ by $(n-1)! = 2$:

$$\begin{aligned}
3-1 &= 1(2)+0 \;\rightarrow\; M(3,1) = M(0+1,1)+1(2)\\
4-1 &= 1(2)+1 \;\rightarrow\; M(4,1) = M(1+1,1)+1(2)\\
5-1 &= 2(2)+0 \;\rightarrow\; M(5,1) = M(0+1,1)+2(2)\\
6-1 &= 2(2)+1 \;\rightarrow\; M(6,1) = M(1+1,1)+2(2)
\end{aligned}$$

Then adding 2 to each entry of $M(1,1)$ will produce $M(3,1)$ and adding 4 to each entry of $M(2,1)$ will produce $M(6,1)$, and so on. After replacing these rows we will have completely filled the first column of the matrix that corresponds to the 1×2

blocks which make up the multiplication table. The rest of the 1×2 blocks will be copies of the blocks from the first column.

1	2		
2	1		
3	4		
4	3		
5	6		
6	5		

For entry $M(1,2)$ we consider the arrangement of the first permutation of S_3: 1 2 3. There are no other permutations before it in order, so the value of the second slot, 2, appears exactly once in the second slot of the first 1 permutations. Then

$$M(1,2) = M((2-1)2+1,1) = M(3,1).$$

For entry $M(3,3)$ we consider the arrangement of the third permutation, 2 1 3. The value of the third slot, 3, appears exactly twice in the first 3 permutations. Then

$$M(3,3) = M((3-1)2+2,1) = M(6,1).$$

Continuing this process we get all of the multiplication table for S_3:

1	2	3	4	5	6
2	1	5	6	3	4
3	4	1	2	6	5
4	3	6	5	1	2
5	6	2	1	4	3
6	5	4	3	2	1

Perhaps this description of the multiplication table is helpful in determining whether two permutations will generate the symmetric group. It does at least exclude us from requiring to multiply permutations directly in order to determine whether their generated group is all of S_n. However, filling out and tracking the multiplication table may be as difficult as computing the generated group directly.

REFERENCES

[DF04] David S. Dummit and Richard M. Foote, *Abstract algebra*, third ed., John Wiley & Sons Inc., Hoboken, NJ, 2004. MR 2286236 (2007h:00003)

[Dix69] John D. Dixon, *The probability of generating the symmetric group*, Math. Z. **110** (1969), 199–205. MR 0251758 (40 #4985)

[DNZ08] Marc Deléglise, Jean-Louis Nicolas, and Paul Zimmermann, *Landau's function for one million billions*, J. Théor. Nombres Bordeaux **20** (2008), no. 3, 625–671. MR 2523311 (2010e:11119)

[Jak11] Rafael Jakimczuk, *Integer sequences, functions of slow increase, and the Bell numbers*, J. Integer Seq. **14** (2011), no. 5, Article 11.5.8, 11. MR 2802060 (2012e:11045)

[LP01] Alexander Lubotzky and Igor Pak, *The product replacement algorithm and Kazhdan's property (T)*, J. Amer. Math. Soc. **14** (2001), no. 2, 347–363 (electronic). MR 1815215 (2003d:60012)

[MNR89] Jean-Pierre Massias, Jean-Louis Nicolas, and Guy Robin, *Effective bounds for the maximal order of an element in the symmetric group*, Math. Comp. **53** (1989), no. 188, 665–678. MR 979940 (90e:11139)

[Nic97] Jean-Louis Nicolas, *On Landau's function $g(n)$*, The mathematics of Paul Erdős, I, Algorithms Combin., vol. 13, Springer, Berlin, 1997, pp. 228–240. MR 1425188 (98b:11096)

[Pak00] Igor Pak, *The product replacement algorithm is polynomial*, 41st Annual Symposium on Foundations of Computer Science (Redondo Beach, CA, 2000), IEEE Comput. Soc. Press, Los Alamitos, CA, 2000, pp. 476–485. MR 1931844

[Rob55] Herbert Robbins, *A remark on Stirling's formula*, Amer. Math. Monthly **62** (1955), 26–29. MR 0069328 (16,1020e)

[Wei13] Eric W. Weisstein, *Unsolved problems. MathWorld – A Wolfram Web Resource*, `http://mathworld.wolfram.com/UnsolvedProblems.html`, 2013, [Online; accessed 25-March-2013].

APPENDIX A

COMPUTING PERMUTATION GROUPS

A useful online tool for quickly calculating permutation subgroups generated by a particular set of permutations is given on the site `http://wims.unice.fr/wims/en_tool~algebra~permgroup.html`. In order to generate all the groups generated by two permutations, the Java program in this appendix is given. For speed and ease of coding, S_n is seen as acting on the set $\{0, 1, \ldots, n-1\}$.

A.1 Sn.java

```
/**
 * Representation of a mathematical Symmetric Group.
 *
 * @author Mike Watts
 * @version Mar 25, 2013
 */
public class Sn {
    private long size; //The cardinality of the Symmetric Group.
    private int n; //The maximum integer plus one allowed for the permutations.

    /**
     * Constructor for the Symmetric Group with a given 'n' value.
     *
     * @param n The maximum integer plus one allowed for the permutations.
     */
    public Sn(int n) {
```

```
        this.n = n;
        size = factorial(n);

    }

    /**
     * Get the 'n' value for this Symmetric Group.
     *
     * @return the 'n' value for this Symmetric Group.
     */
    public int getN() {
        return n;
    }

    /**
     * Get the cardinality of this Symmetric Group.
     *
     * @return the cardinality of this Symmetric Group.
     */
    public long getSize() {
        return size;
    }

    /**
     * Get the indexing value using lexicographical ordering.
     *
     * @param p the permutation to index.
     * @return the indexing value for this permutation.
     */
```

```
public int index(Perm p) {
    int index = 0; //The index number of this permutation.
    byte[] perm = new byte[n]; //The integer mapping given by the permutation.

    //Copy the array of the permutation since modifications are to be made in the indexing
        process.
    System.arraycopy(p.getArray(), 0, perm, 0, n );

    for(int i = 0; i < n ; i++) {
        index+= perm[i]*factorial(n - i)/(n - i);

        for(int j = i+1; j < n; j++) {
            if(perm[j] > perm[i]) {
                perm[j]--;
            }
        }
    }
    return index;
}

/**
 * Get the permutation given an index number.
 *
 * @param index The index number for the permutation.
 * @return The permutation given by the index number.
 */
public Perm permOf(int index) {
    byte[] perm = new byte[n]; //The mapping of the integers for the permutation.
```

```
        //Reverse the process of indexing.
        for(int i = 0; i < n; i++) {
            perm[i] = (byte) (index*(n-i)/factorial(n-i));
            index-= perm[i]*factorial(n-i)/(n-i);
        }
        for(int i = n-1; i > 0; i--) {
            for(int j = i-1; j > -1; j--) {
                if(perm[i] >= perm[j]) perm[i]++;
            }
        }
        return new Perm(perm);
    }

    /**
     * Compute the factorial on the given number.
     *
     * @param num The number to perform the factorial operation on.
     * @return the factorial of the number given.
     */
    public long factorial(int num) {
        long fact = 1;
        switch(num) {
            case 0:
                fact = 1;
                break;
            case 1:
                fact = 1;
                break;
            case 2:
```

```
        fact = 2;
        break;
    case 3:
        fact = 6;
        break;
    case 4:
        fact = 24;
        break;
    case 5:
        fact = 120;
        break;
    case 6:
        fact = 720;
        break;
    case 7:
        fact = 5040;
        break;
    case 8:
        fact = 40320;
        break;
    case 9:
        fact = 362880;
        break;
    case 10:
        fact = 3628800;
        break;
    case 11:
        fact = 39916800;
        break;
```

```
            case 12:
                fact = 479001600;
                break;
            case 13:
                fact = 6227020800L;
                break;
            case 14:
                fact = 87178291200L;
                break;
            case 15:
                fact = 1307674368000L;
                break;
            default:
                for(int i = 2; i <= num; i++)
                {
                    fact*= i;
                }
                break;
        }
        return fact;
    }

    /**
     * Determine whether the indexed permutation is even.
     *
     * @param index the index number of the permutation.
     * @return true if the permutation is even, false otherwise.
     */
    public boolean isEven(int index) {
```

```
    if(permOf(index).isEven()) return true;
    else return false;
}

/**
 * Determine whether the permutation is even.
 *
 * @param perm the permutation.
 * @return true if the permutation is even, false otherwise.
 */
public boolean isEven(Perm perm) {
    return perm.isEven();
}

/**
 * Multiply two permutations given by their index numbers.
 *
 * @param p The indexed permutation on the left of the operation
 * @param q The indexed permutation on the right of the operation.
 * @return The product of the two permutations given.
 */
public int mult(int p, int q) {
    return index(permOf(p).multiply(permOf(q)));
}


/**
 * Prints out the multiplication table for the current symmetric group.
 */
```

```
public void printTable() {
    int m; //The product of the current multiplication.

    //Find the maximum number of digits, for spacing purposes.
    int max = 10;
    while(size/max >= 10)
    {
        max*=10;
    }

    //For each pair of integers, multiply the two and print out their result.
    for(int i = 0; i < size; i++) {
        for(int j = 0; j < size; j++) {
            m = mult(i,j);
            System.out.print(m + " ");

            //Add spacers to align columns.
            for(int alpha = max; alpha/(m+1) > 0 && alpha > 1; alpha/= 10)
            {
                System.out.print(" ");
            }
        }
        System.out.println("");
    }
}

/**
 * Print out the permutations and their index numbers.
 */
```

```
public void printPerms() {
    //For each permutation, print out the index number and the standard string
        representation.
    for(int i = 0; i < size; i++) {
        System.out.println(i + " -> " + permOf(i));
    }
}

/**
 * Checks to see if we can predetermine if two permutations are or are not going to generate
    the Symmetric Group.
 *
 * @param indexp One of the two permutations
 * @param indexq The other of two permutations.
 * @return -1 if the pairs do not generate the Symmetric Group, 1 is the pairs do generate
    the Symmetric Group, and 0 if it cannot be determined.
 */
public byte preCheck(int indexp, int indexq) {
    //Convert the index numbers into permutations.
    Perm p = permOf(indexp);
    Perm q = permOf(indexq);

    //Check if both permutations are even.
    if(isEven(indexp) && isEven(indexq)) return -1;

    //Check if intranstive.
    boolean[] num = new boolean[n], nump = new boolean[n], numq = new boolean[n];
    num[0] = true;
    boolean done = false;
```

```
boolean added;
int iter;
int count = 1;
while(!done) {
    added = false;
    for(iter = 0; iter < n; iter++) {
        if(num[iter]) {
            while(!nump[iter]) {
                if(!num[iter]) {
                    count++;
                    num[iter] = true;
                }
                nump[iter] = true;
                iter = p.getArray()[iter];
            }
        }
    }
    for(iter = 0; iter < n; iter++) {
        if(num[iter]) {
            while(!numq[iter]) {
                if(!num[iter]) {
                    count++;
                    num[iter] = true;
                }
                numq[iter] = true;
                iter = q.getArray()[iter];
                added = true;
            }
        }
```

```
            }
            if(!added && count != n) {
                return -1;
            }
            if(count == n) done = true;
        }

        //Check to see if the group is commutative.
        if(mult(indexp, indexq) == mult(indexq, indexp)) {
            return -1;
        }

        //If none of the checks passed, then it is unknown if the two permutations generate the
            Symmetric Group.
        return 0;
    }

    /**
     * Check to see if distance greater than 1 is preserved.
     *
     * @param p One of the two permutations.
     * @param q The other of the two permutations.
     * @return True if there exists distance greater than 1 which is preserved in p by q, False
        otherwise
     */
    public boolean distPres(Perm p, Perm q) {
        byte[] p1 = p.getArray();

        byte[] p2 = q.getArray();
```

```
boolean distPres = false;
byte gcd = gcd(p.cycletype());

if(gcd == 1) {
    return false;
}
int max = p.maxLength();
for(byte d = 2; d < max && !distPres; d++) {
    byte[] test = {gcd, (byte) d};
    if(gcd(test) != 1) {
        boolean currDist = true;


        for(byte i = 0; i < n; i++) {
            byte a = i;
            byte b = i;
            for(int pow = 1; pow <= d; pow++) {
                b = p1[b];
            }
            byte dist = p.distance(p2[a], p2[b]);
            //System.out.println("distance of " + dist + " for " + i + " in " + p);
            if(dist == -1 || dist % d != 0) {
                currDist = false;
            }
        }
        if(currDist) {
            distPres = true;
```

```
            }
        }
    }
    return distPres;
}

/**
 * Greatest Common Divisor between two integers.
 */
public byte gcd(byte a, byte b) {
    if (b == 0) return a;
    return gcd(b, (byte)(a % b));
}

/**
 * Greatest Common Divisor of a set of integers.
 *
 * @param nums Array as a set of integers
 * @return Greatest common divisor of the set of integers in nums
 */
public byte gcd(byte[] nums) {
    byte gcd = nums[0];
    for(int i = 1; i < nums.length; i++) {
        gcd = gcd(gcd, nums[i]);
    }
    return gcd;
}

/**
```

```
     * Multiply two permutations
     *
     * @param p The permutation on the left of the operation.
     * @param q The permutation on the right of the permutation
     * @return The product of the two permutations.
     */
    public Perm mult(Perm p, Perm q) {
        return p.multiply(q);
    }
}
```

A.2 Group.java

```
import java.util.LinkedList;

/**
 * A mathematical group of permutations.
 *
 * @author Mike Watts
 * @version Nov 13, 2007
 */
public class Group {
    private boolean[] group; //The permutations in this group.
    private long size; //A holder for where the next empty spot is.
    private Sn Sym; //The Symmetric Group that this group is a subgroup of.

    /**
     * Construct a new group of permutations generated by two permutations.
     *
```

```
     * @param indexp A permutation in the group.
     * @param indexq A permutation in the group.
     * @param Sym The Symmetric Group that this group is a subgroup of.
     */
    public Group(int indexp, int indexq, Sn Sym) {
        this.Sym = Sym;

        //If one of the permutations is the identity or if it's equal to the other, find a distinct
            permutation.
        if(indexp == indexq) {
            indexq = Sym.mult(indexp, indexp);
        }
        else if(indexp == 0) {
            indexp = Sym.mult(indexq, indexq);
        }
        else if(indexq == 0) {
            indexq = Sym.mult(indexp,indexp);
        }

        //Initialize the group with the identity permutation.
        group = new boolean[(int) Sym.getSize()];
        size = 1;
        group[0] = true;

        //If both permutations are not the identity, then build the group.
        if(indexq != 0 && indexp != 0)
        {
            //Build the group based on the two permutations.
            buildGroup(indexp, indexq);
```

```
    }
    //If both permutations are the identity, then the group is of size 1.
    else
    {
        group[indexp] = true;
        group[indexq] = true;
        if(indexp == indexq)
            size = 1;
        else
            size = 2;
    }

}

/**
 * Construct a new group of permutations generated by two permutations.
 *
 * @param p A permutation in the group.
 * @param q A permutation in the group.
 * @param Sym The Symmetric Group that this group is a subgroup of.
 */
public Group(Perm p, Perm q, Sn Sym) {
    this.Sym = Sym;

    //If one of the permutations is the identity or if it's equal to the other, find a distinct
        permutation.
    if(p.equals(q)) {
        q = Sym.mult(p, p);
    }
```

```
else if(p.isIdentity()) {
    p = Sym.mult(q, q);
}
else if(q.isIdentity()) {
    q = Sym.mult(p, p);
}

//Initialize the group with the identity permutation.
group = new boolean[(int) Sym.getSize()];
size = 1;
group[0] = true;

//If both permutations are not the identity, then build the group.
if(!q.isIdentity() && !p.isIdentity())
{
    //Build the group based on the two permutations.
    buildGroup(p, q);
}
//If both permutations are the identity, then the group is of size 1.
else
{
    group[Sym.index(p)] = true;
    group[Sym.index(q)] = true;
    if(p.equals(q))
        size = 1;
    else
        size = 2;
}
```

```
}

/**
 * Search for a permutation in the group
 *
 * @param indexq The permutation to find.
 * @return true if the permutation is in the group, false otherwise.
 */
public boolean isIn(int index) {
    return group[index];
}

/**
 * Search for a permutation in the group
 *
 * @param perm The permutation to find.
 * @return true if the permutation is in the group, false otherwise.
 */
public boolean isIn(Perm perm) {
    return group[Sym.index(perm)];
}

/**
 * Get the cardinality of this group.
 *
 * @return the cardinality of this group.
 */
public long getSize() {
    return size;
```

```
}

/**
 * Build a group generated by two permutations.
 *
 * @param indexp The index of a permutation.
 * @param indexq The index of a permutation.
 */
public void buildGroup(int indexp, int indexq) {
    //Add the permutations to the group.
    group[indexp] = true;
    group[indexq] = true;
    size = 3;

    LinkedList<Integer> list = new LinkedList<Integer>();
    list.add(indexq);

    //Multiply each permutation in the group by the previous permutations.
    while(!list.isEmpty() && size <= Sym.getSize()/2) {
        int current = list.remove();
        addPerm(list, Sym.mult(current, indexp));
        addPerm(list, Sym.mult(current, indexq));
        addPerm(list, Sym.mult(indexp, current));
        addPerm(list, Sym.mult(indexq, current));
    }

    //If the group is more than half of Sn, then it is Sn.
    if(size > Sym.getSize()/2)
    {
```

```
            size = Sym.getSize();
        }
    }

    /**
     * Add a permutation to this group.
     *
     * @param list A list of permutations to multiply.
     * @param perm The permutation to add to this group.
     */
    private void addPerm(LinkedList<Integer> list, int perm)
    {
        if(!group[perm]) {
            group[perm] = true;
            size++;
            list.add(perm);
        }
    }

    /**
     * Build a group generated by two permutations.
     *
     * @param indexp The index of a permutation.
     * @param indexq The index of a permutation.
     */
    public void buildGroup(Perm p, Perm q) {
        //Add the permutations to the group.
        group[Sym.index(p)] = true;
        group[Sym.index(q)] = true;
```

```
        size = 3;

        LinkedList<Perm> list = new LinkedList<Perm>();
        list.add(q);

        //Multiply each permutation in the group by the previous permutations.
        while(!list.isEmpty() && size <= Sym.getSize()/2) {
            Perm current = list.remove();
            addPerm(list, Sym.mult(current, p));
            addPerm(list, Sym.mult(current, q));
            addPerm(list, Sym.mult(p, current));
            addPerm(list, Sym.mult(q, current));
        }

        //If the group is more than half of Sn, then it is Sn.
        if(size > Sym.getSize()/2)
        {
            size = Sym.getSize();
        }
    }

    /**
     * Add a permutation to this group.
     *
     * @param list A list of permutations to multiply.
     * @param perm The permutation to add to this group.
     */
    private void addPerm(LinkedList<Perm> list, Perm perm)
    {
```

```
        int index = Sym.index(perm);
        if(!group[index]) {
            group[index] = true;
            size++;
            list.add(perm);
        }
    }

    public boolean isSn() {
        return this.size == Sym.getSize();
    }

    /**
     * Get a string representation of the Group in set notation.
     *
     * @return A string representation of this group.
     */
    public String toString() {
        if(size == Sym.getSize())
            return "Sn";

        String result = "{";

        for(int i = 0; i < size; i++) {
            if(group[i] && i != size-1)
                result+= Sym.permOf(i) + ", ";
            else if(group[i])
                result+= Sym.permOf(i);
        }
```

```
        result+= "}";
        return result;
    }
}
```

A.3 Perm.java

```
/**
 * Representation of a mathematical permutation of the symmetric group.
 *
 * @author Mike Watts
 * @version Mar 25, 2013
 */
public class Perm {
    private byte[] perm; //Mapping of the integers 0,1,...,n−1 for this permutation in Sn.
    private boolean isIdent;

    /**
     * Constructor for a permutation.
     *
     * @param perm the mapping of the integers defined by this permutation.
     */
    public Perm(byte[] perm) {
        this.perm = perm;
        isIdent = checkIdent();
    }

    /**
```

```
 * Multiply (compose) two permutations in the order (this * other).
 *
 * @param other The other permutation to multiply by this permutation.
 * @return The result product of the two permutations
 */
public Perm multiply(Perm other) {
    byte[] result = new byte[this.perm.length]; //The product of the two permutations.

    //For each integer, compose the other permutation into this one.
    for(byte i = 0; i < this.perm.length; i++) {
        result[i] = this.perm[other.perm[i]];
    }
    return new Perm(result);
}

/**
 * Get the mapping of the integers for this permutation.
 *
 * @return the array which represents the mapping of the integers.
 */
public byte[] getArray() {
    return perm;
}

/**
 * Get the cyclic form of the permutation.
 *
 * @return the standard string representation of this permutation.
 */
```

```
public String toString() {
    String result = ""; //The resultant string.
    boolean[] checked = new boolean[perm.length]; //Checks to see which integer has been
        added to the resultant string.
    boolean start = true; //Check to see if adding a new cycle.

    //If this permutation is the identity permutation, return the word 'id'.
    if(maxLength() == 1) return "id";

    //Parse each integer to add to the result string.
    for(byte i = 0; i< perm.length; i++) {
        //If the integer not been checked and does it not map to itself, add it to the result
            string.
        if(!checked[i] && perm[i]!=i) {
            //If a new cycle is started, add a '('.
            if(start) {
                result+= "(";
            }
            //Add the integer and set it as checked.
            result+= i;
            checked[i] = true;

            //If at the end of a cycle, then close it with a ')'.
            if(checked[perm[i]]) {
                result+= ")";
                i = perm[i];
                start = true;
            }
            //If not at the end of a cycle, add a ','.
```

```
            else {
                result+= ",";
                i = perm[i];
                i--;
                start = false;
            }
        }
    }
    return result;
}

/**
 * Check if this permutation is equal to another.
 *
 * @param other The other permutation to compare this one to.
 * @return true if the two permutations are the same, false otherwise.
 */
public boolean equals(Perm other) {
    boolean isEqual = true; //Determines if this permutation is equal to other.

    //For each integer, check to see if the two permutation map to the same integer.
    for(byte i = 0; i < this.perm.length && isEqual; i++) {
        isEqual = this.perm[i] == other.perm[i];
    }
    return isEqual;
}

/**
 * Check to see if this permutation is even.
```

```
     *
     * @return true if this permutation is even, false otherwise.
     */
    public boolean isEven() {
        int count = length(); //The number of integers which do not map to themselves

        return (this.toString().length() - count) % 2 == 0;
    }

    /**
     * Check to see if this permutation is the identity.
     *
     * @return true if this permutation is the identity, false otherwise.
     */
    public boolean isIdentity() {
        return isIdent;
    }

    /**
     * Check to see if this permutation is the identity.
     *
     * @return true if this permutation is the identity, false otherwise.
     */
    private boolean checkIdent() {
        int count = 0; //Counts the number of similarities
        int n = perm.length;

        for(int i = 0; i < n; i++) {
            if(perm[i] == i) {
```

```
            count++;
        }
        else {
            break;
        }
    }

    return (count == n);
}

/**
 * Get the length of the cycle with maximum length.
 *
 * @return the maximum length of a cycle of this permutation.
 */
public int maxLength() {
    int max = 1; //The current recorded maximum cycle length.
    int cycleLength; //The current cycles length.
    boolean[] num = new boolean[perm.length]; //Check for whether an integer has been
        checked.

    //For each integer, find it's cycle and record it's length.
    for(byte i = 0; i < perm.length; i++) {
        cycleLength = 0;
        //Pass through each integer in this cycle.
        while(!num[i]) {
            num[i] = true;
            i = perm[i];
            cycleLength++;
```

```
        }
        //If the current cycle's length is greater than the maximum, set the maximum to the
            current cycle's length.
        if(cycleLength > max) {
            max = cycleLength;
        }
    }
    return max;
}



/**
 * Get the inverse permutation of this permutation.
 *
 * @return the inverse permutation.
 */
public Perm inverse() {
    byte[] inv = new byte[perm.length]; //The inverse permutation mapping.\

    //For each integer, reverse map it.
    for(byte i = 0; i < perm.length; i++) {
        inv[perm[i]] = i;
    }
    return new Perm(inv);
}

/**
 * Get the cycle type of this permutation.
 *
```

```
     * @return the cycle type of this permutation returned as an array.
     */
    public byte[] cycletype() {
        byte[] cycletype = new byte[perm.length];
        byte index = 0;
        boolean[] nums = new boolean[perm.length];
        for(int i = 0; i < perm.length; i++) {
            if(!nums[i]) {
                nums[i] = true;
                byte count = 1;
                for(int j = perm[i]; i != j; j = perm[j])
                {
                    nums[j] = true;
                    count++;
                }
                cycletype[index] = count;
                index++;
            }
        }

        return cycletype;
    }

    /**
     * Get the distance between two integers in this permutation.
     *
     * @param a a number between 0 and (n−1).
     * @param b a number between 0 and (n−1).
     * @return the distance in this permutation between a and b.
```

```
 */
public byte distance(byte a, byte b) {
    byte count = 0;

    while(a != b && count <= perm.length) {
        count++;
        a = perm[a];
    }
    if(count > perm.length) {
        return -1;
    }
    return count;
}

/**
 * Get the value that this permutation maps k to.
 */
public byte of(byte k) {
    return perm[k];
}

/**
 * Get the order of this permutation.
 */
public int order() {
    Perm m = this.multiply(this);
    int order = 1;
    while(!this.equals(m)) {
        order++;
```

```
            m = m.multiply(this);
        }
        return order;
    }
}
```