The Minnesota Journal of Undergraduate Mathematics

Properties and Calculations of Constructive Orderings of $\mathbb{Z}/n\mathbb{Z}$

Zackary Baker

The King's University

The Minnesota Journal of Undergraduate Mathematics

Volume 4 (2018-2019 Academic Year)

Sponsored by
School of Mathematics
University of Minnesota
Minneapolis, MN 55455

The Minnesota Journal of Undergraduate Mathematics

Volume 4 (2018-2019 Academic Year)

Properties and Calculations of Constructive Orderings of $\mathbb{Z}/n\mathbb{Z}$

Zackary Baker

The King's University

Abstract. A *sequencing* of a finite group G of order n is a sequence $g_1, g_2, ..., g_n$ of the elements of G whose set of partial products $\{g_1 g_2 \cdots g_i | 1 \leq i \leq n\}$ contains every element of the group G. In this paper, we study this in the particular case of additive groups modulo n, replacing the partial products with partial sums. We make and prove several observations about these sequencings, and calculate how many there are for $n \leq 16$.

To do this, we define an operator called the *dagger* on a sequence, and collect the results of this operator into what we call the *dagger set* of the group. We then analyze several properties of this set and collect computational data on it.

1. Introduction

This paper proves results on *sequenceable groups* in the specific case of the additive group $\mathbb{Z}/n\mathbb{Z}$ for positive integers n. If any readers of this paper do not know the definition of a group, but understand the structure of $\mathbb{Z}/n\mathbb{Z}$, they can read and understand the majority of the content of this paper.

The idea of a sequenceable group was first defined in 1961 by B. Gordon[2]:

Definition 1.1. A finite group G of order n is *sequenceable* if the elements of G can be arranged in a sequence $g_1, g_2, ..., g_n$ such that the partial products

$$\prod_{i=1}^{k} g_i,$$

are distinct for $1 \leq k \leq n$.

Here, we reframe this definition into language that is easy to work with in the case of the additive group $\mathbb{Z}/n\mathbb{Z}$. To begin, let $\prec$ be a strict total order on G. If an additive group G of size n is ordered so that $g_1 \prec g_2 \prec \cdots \prec g_n$, we denote the kth partial sum, $1 \leq k \leq n$, by

* *Corresponding author*

$g_k\dagger$. That is,

$$g_k\dagger := \sum_{i=1}^{k} g_i.$$

The notation $g_k\dagger$ is inspired by the fact that a partial sum can be loosely viewed as an additive version of a factorial: instead of multiplying a positive integer with all integers smaller than it, we are adding an integer with all integers smaller than it under $\prec$. One might wish to call $g_k\dagger$ a sort of additive factorial. Now that we have the notion of these partial sums, or daggers, as we will call them, we wish to talk about them collectively:

Definition 1.2. Given an additive group G with some strict total order $\prec$, the *dagger set* of G with respect to $\prec$ is defined to be $D_{\prec}(G) := \{g\dagger_{\prec} \mid g \in G\}$.

We can now define a sequenceable additive group using this language:

Definition 1.3. A finite additive group G is *sequenceable* if and only if there exists an ordering $\prec$ such that $D_{\prec}(G) = G$.

In this paper we do not only wish to determine whether or not a group is sequenceable, but rather to know more about the orderings $\prec$ such that $D_{\prec}(G) = G$. In some literature, a sequence $g_1, g_2, ..., g_n$ on a finite group G which can be used to show that G is sequenceable is called a *sequencing*. We will use the term *constructive ordering* (of G) to describe an ordering $\prec$ such that $D_{\prec}(G) = G$.

Some known results on sequenceable additive groups $\mathbb{Z}/n\mathbb{Z}$ are:

Proposition 1.4. [2]

(a) *An ordering $\prec$ such that $D_{\prec}(\mathbb{Z}/n\mathbb{Z}) = \mathbb{Z}/n\mathbb{Z}$ always has its least element equal to 0.*
(b) *There exists an ordering $\prec$ such that $D_{\prec}(\mathbb{Z}/n\mathbb{Z}) = \mathbb{Z}/n\mathbb{Z}$ if and only if n is even. (Note that in contrast, not every group of even order is sequenceable; the Klein 4 group is an example of an even order group that is not sequenceable.)*
(c) (Dagger Uniqueness Property) *For any $g_i, g_j \in \mathbb{Z}/n\mathbb{Z}$, if $g_i\dagger = g_j\dagger$ for $i \neq j$, then $D_{\prec}(\mathbb{Z}/n\mathbb{Z}) \neq \mathbb{Z}/n\mathbb{Z}$.*

Though these results are well-known for sequenceable groups, we state and prove Proposition 1.4(a) and the forward direction of Proposition 1.4(b) here. This is because the proofs in the literature need more mathematical background than we are assuming all readers of this paper have. Here, we write the proofs using elementary notions.

Proof of Proposition 1.4(a):

Proof. This proof is by contradiction. Assume that 0 is not the least element of the ordering $\prec$. Then $g_i = 0$ for some i such that $2 \leq i \leq n$. Then $g_i\dagger = 0 + g_{i-1}\dagger = g_{i-1}\dagger$, so an element is duplicated and the dagger set is smaller than the group. Thus, $D_{\prec}(\mathbb{Z}/n\mathbb{Z})$ cannot equal $\mathbb{Z}/n\mathbb{Z}$, and so a contradiction is reached. □

Remark. Here, we remind ourselves of the definition of triangular numbers, and a formula for them, as they will be useful in the following proof and throughout this paper. If $\prec$ is any order on $\mathbb{Z}/n\mathbb{Z}$, and the elements of $\mathbb{Z}/n\mathbb{Z}$ can be written as $g_1 \prec g_2 \prec \cdots \prec g_n$, then we always have

$$g_n\dagger = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}.$$

This is the same as the $(n-1)^{\text{st}}$ triangular number, which is denoted T_{n-1}. It is well known that if n is odd then $T_{n-1} \equiv 0 \pmod{n}$, and if n is even, $T_{n-1} \equiv \frac{n}{2} \pmod{n}$, and will be shown here. From the above definition, $T_n = \frac{n(n+1)}{2}$. If n is even, then $\frac{n}{2}$ in an integer, meaning it is in $\mathbb{Z}/n\mathbb{Z}$. We can rewrite T_n as $\frac{n^2+n}{2}$, or $\frac{n^2}{2}+\frac{n}{2} = n\frac{n}{2}+\frac{n}{2}$. Taking this expression mod n clearly shows that $T_n \equiv \frac{n}{2} \pmod{n}$. If n is odd, then $n+1$ is even, and $\frac{n+1}{2}$ is an integer. Then, since $T_n = n \cdot \frac{n+1}{2}$, it is clear to see that $T_n \equiv 0 \pmod{n}$.

Partial proof of Proposition 1.4(b): Here, we prove the contrapositive of the forward direction of this "if and only if" statement. That is, we prove that if n is odd, then we have $D_\prec(\mathbb{Z}/n\mathbb{Z}) \neq \mathbb{Z}/n\mathbb{Z}$ for any ordering $\prec$.

Proof. We prove this by showing that $D_\prec(\mathbb{Z}/n\mathbb{Z})$ will have fewer than n elements whenever n is odd. By Proposition 1.4(a) we know that 0 must be the first element of this ordering; i.e. $g_1 = 0$, and $g_1\dagger = 0$. Also, as remarked above, $g_n\dagger = T_{n-1} \equiv 0 \pmod{n}$. Thus $D_\prec(\mathbb{Z}/n\mathbb{Z})$ is always smaller than $\mathbb{Z}/n\mathbb{Z}$ when n is odd. □

When we first began writing this paper, we were not only interested in sequenceable groups, but also in groups where the dagger set was equal to a subgroup of the original group. We quickly observed that the dagger set will never give a proper subgroup of $\mathbb{Z}/n\mathbb{Z}$. We state and prove this fact below, and in the remainder of this paper only focus on the case where the dagger set is equal to the entire group.

Proposition 1.5. *If $D_\prec(\mathbb{Z}/n\mathbb{Z})$ is a proper subset of $\mathbb{Z}/n\mathbb{Z}$, then it is not a subgroup of $\mathbb{Z}/n\mathbb{Z}$.*

Proof. This proof is by contradiction. Assume that there exists an order $\prec$ such that $D_\prec(\mathbb{Z}/n\mathbb{Z})$ is a subgroup of $\mathbb{Z}/n\mathbb{Z}$. Since $\mathbb{Z}/n\mathbb{Z}$ is cyclic, we have that $D_\prec(\mathbb{Z}/n\mathbb{Z}) = \langle k \rangle$ for some nontrivial divisor k of n. Then every element of $D_\prec(\mathbb{Z}/n\mathbb{Z})$ is a multiple of k. Let $g_1 \prec g_2 \prec \cdots \prec g_n$ be the order of the elements of $\mathbb{Z}/n\mathbb{Z}$ under $\prec$. For some $i \in \{1,...,n\}$, $g_i = 1$. If $i = 1$ then $g_i\dagger = 1 \notin \langle k \rangle$, so assume $i > 1$. Then, either $g_{i-1}\dagger$ or $g_i\dagger$ will not be in $D_\prec(\mathbb{Z}/n\mathbb{Z})$, since they differ by 1 and thus cannot both be multiples of k, so a contradiction is reached. Thus, there are no orderings on $\mathbb{Z}/n\mathbb{Z}$ such that $D_\prec(\mathbb{Z}/n\mathbb{Z})$ is a proper subgroup of $\mathbb{Z}/n\mathbb{Z}$ for any $n \in \mathbb{N}$. □

2. Properties of constructive orderings

In this section, we prove several properties of constructive orderings for $\mathbb{Z}/n\mathbb{Z}$. As observed in the previous section, constructive orderings only exist when n is even. Therefore, throughout this section and the rest of the paper, we will assume that n is even.

2.1. **The natural order.** If an ordering $\prec$ is defined by $0 \prec 1 \prec \cdots \prec n-1$, then we call $\prec$ the natural order, and denote it $<$. In this section we consider whether the natural order is ever a constructive ordering, and if so, when it is or is not.

As an example we compute $D_<(\mathbb{Z}/n\mathbb{Z})$ for $n \leq 16$, and determine whether or not it is a constructive ordering.

n	$D_<(\mathbb{Z}/n\mathbb{Z})$	**Is $<$ a constructive ordering for $\mathbb{Z}/n\mathbb{Z}$?**
2	{0,1}	✓
4	{0,1,2,3}	✓
6	{0,1,3,4}	×
8	{0,1,2,3,4,5,6,7}	✓
10	{0,1,3,5,6,8}	×
12	{0,1,3,4,6,7,9,10}	×
14	{0,1,3,6,7,8,10,13}	×
16	{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}	✓

Table 1. Dagger subsets of $\mathbb{Z}/n\mathbb{Z}$, $n \leq 16$ using the natural order

In the small set of examples, we do see that the natural order is a constructive ordering for $n = 2, 4, 8$ and 16. After computing a number of other examples, we were led to the following conclusion:

Theorem 2.1. *For $<$ the natural order, $D_<(\mathbb{Z}/n\mathbb{Z}) = \mathbb{Z}/n\mathbb{Z}$ if and only if $n = 2^k$ for some positive integer k.*

Proof. We will prove the forward direction by contrapositive. Let $n = (2\ell+1)\cdot x$, where $\ell \in \mathbb{N}$ and $x \geq 2$ is a power of 2, and let c be the smallest positive integer such that $cx > \ell$. If there exist $g_i, g_j \in \mathbb{Z}/n\mathbb{Z}$ such that $g_i\dagger = g_j\dagger$, then $<$ is not a constructive ordering. Choose $g_i = cx - \ell - 1$ and $g_j = cx + \ell$. We must confirm that g_i and g_j are reduced modulo n. We begin by showing that $g_i \geq 0$. We begin with the inequality $cx > \ell$. This inequality is the same as $cx \geq \ell + 1$. Rearranging this equation, we have that $cx - \ell - 1 \geq 0$. Next, we will show that $g_j < n$. Since c is defined to be the smallest integer such that $cx > \ell$, we know that $(c-1)x \leq \ell$. From this, we obtain $(c-1)x + \ell \leq 2\ell$, from which we can derive that $(c-1)x + \ell < 2\ell x$. Adding x to both sides gives $cx + \ell < 2\ell x + x$. Rearranging the right hand side results in the inequality $cx + \ell < x(2\ell+1)$, which we can rewrite as $cx + \ell < n$. Finally, since $cx - \ell - 1$ is clearly less than $cx + \ell$, we know that $0 \leq g_i < g_j < n$. Therefore, g_i and g_j are reduced modulo n.

Calculating $g_i\dagger$ and $g_j\dagger$, we get

$$g_i\dagger = 0 + 1 + 2 + \cdots + (cx - \ell - 1)$$

and

$$g_j\dagger = 0 + 1 + 2 + \cdots + (cx - \ell) + (cx - \ell + 1) + \cdots + (cx - 1) + cx + (cx + 1) + \cdots + (cx + \ell).$$

Thus,

$$g_j\dagger - g_i\dagger = (cx + \ell) + \cdots + (cx + 1) + cx + (cx - 1) + \cdots + (cx - \ell + 1) + (cx - \ell).$$

There are $2\ell+1$ terms in this series, and the average of the series is cx. Thus,

$$g_j\dagger - g_i\dagger = (2\ell+1)\cdot cx = c\cdot(2\ell+1)x = c\cdot n \equiv 0 \pmod{n}.$$

This means that $g_j\dagger = g_i\dagger$, and so by Proposition 1.4(c), the natural ordering is not a constructive ordering for $\mathbb{Z}/n\mathbb{Z}$.

Now we will prove the reverse direction. This proof will be by induction on k. For the base case of $k=1$, the group in question is $\mathbb{Z}/2\mathbb{Z}$. The natural order for $\mathbb{Z}/2\mathbb{Z}$ is (0,1). Calculating the dagger set for this ordering gives $0\dagger = 0$ and $1\dagger = 0+1 = 1$, so we have $D_<(\mathbb{Z}/2\mathbb{Z}) = \mathbb{Z}/2\mathbb{Z}$, and the base case holds. Assume that for some positive integer m, we have that $D_<(\mathbb{Z}/2^m\mathbb{Z}) = \mathbb{Z}/2^m\mathbb{Z}$. Now consider $D_<(\mathbb{Z}/2^{m+1}\mathbb{Z})$. Define

$$D_1 := \{0\dagger, 1\dagger, \ldots (2^m-1)\dagger\}$$

and

$$D_2 := \{2^m\dagger, (2^m+1)\dagger, \ldots, (2^{m+1}-1)\dagger\}.$$

Here, D_1 and D_2 are subsets of $D_<(\mathbb{Z}/2^{m+1}\mathbb{Z})$. Since the partial sums $0\dagger, 1\dagger, \ldots, (2^m-1)\dagger$ are all unique modulo 2^m, by the assumption that $D_<(\mathbb{Z}/2^m\mathbb{Z}) = \mathbb{Z}/2^m\mathbb{Z}$, it follows that these are also all unique modulo 2^{m+1}. Thus there are no repeat elements in D_1, a subset of $D_<(\mathbb{Z}/2^{m+1}\mathbb{Z})$.

Thus, to prove that $D_<(\mathbb{Z}/2^{m+1}\mathbb{Z}) = \mathbb{Z}/2^{m+1}\mathbb{Z}$ we must prove that every dagger in D_2 evaluates to a unique element in $\mathbb{Z}/2^{m+1}\mathbb{Z}$ and also that $D_1 \cap D_2 = \emptyset$, since this would imply that $\left|D_<(\mathbb{Z}/2^{m+1}\mathbb{Z})\right| = 2^{m+1}$ and thus the dagger set is equal to the whole group.

First, we show that $2^m\dagger, (2^m+1)\dagger, \ldots, (2^{m+1}-1)\dagger$ are all unique mod 2^{m+1}. Each of these expressions can be written as $(2^m+c)\dagger$ where $0 \le c \le 2^m-1$. We observe that

$$\begin{aligned}
(2^m+c)\dagger &= (2^m+c)+(2^m+c-1)+\cdots+(2^m+2)+(2^m+1)+2^m \\
&\quad+(2^m-1)+(2^m-2)+\cdots+(2^m-(c-1))+(2^m-c) \\
&\quad+(2^m-(c+1))+(2^m-(c+2))+\cdots+1 \\
&= (2^m+c)+(2^m+c-1)+\cdots+(2^m+2)+(2^m+1)+2^m \\
&\quad+(2^m-c)+(2^m-(c-1))+\cdots+(2^m-2)+(2^m-1) \\
&\quad+(2^m-(c+1))+(2^m-(c+2))+\cdots+1.
\end{aligned}$$

If we look at the last three lines of this equation, we see that there is a lot of simplification that we can do, since we're adding, for example, (2^m+c) with (2^m-c) in the line below it. The sum of these two quantities adds to 2^{m+1}. The same cancellation is possible with several other pairs of terms. Thus we have

$$\begin{aligned}
(2^m+c)\dagger &= \left(2^{m+1}+2^{m+1}+\cdots+2^{m+1}\right)+2^m \\
&\quad+(2^m-(c+1))+(2^m-(c+2))+\cdots+1 \\
&\equiv 2^m+(2^m-(c+1))+(2^m-(c+2))+\cdots+1 \pmod{2^{m+1}} \\
&= 2^m+(2^m-(c+1))\,\dagger.
\end{aligned}$$

Thus, since each element in D_2 can be written as $2^m + (2^m - (c+1))\dagger$ for a unique integer c, and we know that the quantities $(2^m - (c+1))\dagger$ are all unique mod 2^{m+1}, then clearly the quantities $2^m + (2^m - (c+1))\dagger$ are also all unique mod 2^{m+1}. Therefore, D_2 contains 2^m distinct elements.

Now to show that $D_1 \cap D_2 = \emptyset$ we observe that every element in D_2 differs from an element in D_1 by 2^m. Since $0\dagger, 1\dagger, \ldots, (2^m-1)\dagger$ are all unique mod 2^m, no two of these daggers differ by 2^m, or else they would not be unique. Thus we must have that $D_1 \cap D_2 = \emptyset$. Therefore $D_<(\mathbb{Z}/2^{m+1}\mathbb{Z}) = \mathbb{Z}/2^{m+1}\mathbb{Z}$, and the proof is established by induction. □

2.2. **Results on constructive orderings.** In this section we make several observations about constructive orderings and prove them. Throughout this section, we assume n is an even, positive integer.

Proposition 2.2. *If $n > 2$, $\frac{n}{2}$ cannot be the greatest element in a constructive ordering.*

Proof. This will be a proof by contradiction. Assume $\frac{n}{2}$ is the greatest element of $\mathbb{Z}/n\mathbb{Z}$ under a constructive ordering $\prec$. Then, since the dagger of the largest element of $\mathbb{Z}/n\mathbb{Z}$ is T_{n-1}, $\frac{n}{2}\dagger = T_{n-1} \equiv \frac{n}{2} \pmod{n}$. Let g be the element directly preceding $\frac{n}{2}$ under the order $\prec$. Then, $g\dagger = \frac{n}{2}\dagger - \frac{n}{2} = \frac{n}{2} - \frac{n}{2} = 0$. But since 0 must be the first element under this order by Proposition 1.4(a), we have two separate elements with daggers equal to 0. Therefore, by Proposition 1.4(c), $D_\prec(\mathbb{Z}/n\mathbb{Z}) \neq \mathbb{Z}/n\mathbb{Z}$. □

Proposition 2.3. *The value $\frac{n}{2}$ cannot be the second smallest element in a constructive ordering.*

Proof. By Proposition 1.4(a) we know that the least element in a constructive ordering is 0. If $\frac{n}{2}$ is the second least element, then $\frac{n}{2}\dagger = \frac{n}{2} + 0 = \frac{n}{2}$. However, we know that the dagger of the greatest element of $\mathbb{Z}/n\mathbb{Z}$ is $\frac{n}{2}$ (as stated in the remark in the introduction). Thus, if the second least element is $\frac{n}{2}$, then $\frac{n}{2}\dagger$ equals $g_n\dagger$, so by Proposition 1.4(c), $\prec$ is not a constructive ordering. □

Proposition 2.4. *Let $g_1 \prec \cdots \prec g_n$ be an ordering such that $g_i + g_{i+1} + \cdots + g_{i+j} = n$ for some $i, j \in \{2, \ldots, n-1\}, j > i$ such that $i + j \leq n$. Then $\prec$ is not a constructive ordering.*

Proof. Clearly,

$$g_{i+j}\dagger = g_{i+j} + g_{i+j-1} + \cdots + g_{i+1} + g_i + g_{i-1}\dagger = n + g_{i-1}\dagger.$$

Then $g_{i+j}\dagger \equiv g_{i-1}\dagger \pmod{n}$, and so by Proposition 1.4(c), $\prec$ is not a constructive ordering. □

The next propositions concern more than one ordering, and their relationship to each other in terms of whether or not two related orderings can both be constructive orderings. In these proofs, it will be necessary to distinguish whether or not the dagger is being taken with respect to an ordering $\prec_1$ or an ordering $\prec_2$. As such, we will use a subscript to clear up this ambiguity, using the notation $\dagger_{\prec_1}$ or $\dagger_{\prec_2}$ to clarify which ordering is being used.

Proposition 2.5. *Let $\prec_1$ be the ordering $0 \prec_1 g_2 \prec_1 g_3 \prec_1 g_4 \prec_1 \cdots \prec_1 g_n$ and $\prec_2$ be the ordering $0 \prec_2 g_3 \prec_2 g_2 \prec_2 g_4 \prec_2 \cdots \prec_2 g_n$; this is the first order with the second and third elements swapped. If $\prec_1$ is a constructive ordering then $\prec_2$ is not.*

Proof. The first three daggers under $\prec_1$ are $0\dagger_{\prec_1} = 0$, $g_2\dagger_{\prec_1} = g_2$ and $g_3\dagger_{\prec_1} = g_2 + g_3$. When the positions of g_2 and g_3 are interchanged in the order, the first three daggers under $\prec_2$ are $0\dagger_{\prec_2} = 0$, $g_3\dagger_{\prec_2} = g_3$ and $g_2\dagger_{\prec_2} = g_2 + g_3$. If $\prec_1$ is a constructive ordering for $\mathbb{Z}/n\mathbb{Z}$, the value g_3 is in $D_{\prec_1}(\mathbb{Z}/n\mathbb{Z})$. Since this value is not equal to any of the first three daggers under $\prec_1$, and $g_i\dagger_{\prec_1} = g_i\dagger_{\prec_2}$ for all $i > 3$, we have that $g_3 = g_j\dagger_{\prec_2}$ for some $j \in \{4,\ldots,n\}$. Thus under $\prec_2$ the value g_3 is given by two daggers: $g_3\dagger_{\prec_2}$ and $g_i\dagger_{\prec_2}$, so by Proposition 1.4(c), $\prec_2$ cannot be a constructive ordering. □

Proposition 2.6. *Let $\prec_1$ be a constructive ordering $0 \prec_1 g_2 \prec_1 g_3 \prec_1 \cdots \prec_1 g_n$. Then the ordering $\prec_2$ given by $0 \prec_2 n - g_2 \prec_2 n - g_3 \prec_2 \cdots \prec_2 n - g_n$ is also a constructive ordering.*

Proof. Since $\prec_1$ is a constructive ordering then each partial sum under $\prec_1$ is a distinct residue mod n. We also observe that for each $i \in \{2,\ldots,n\}$, taking partial sums under $\prec_2$ we have

$$(n-g_i)\dagger_{\prec_2} \equiv -g_i - g_{i-1} - g_{i-2} - \cdots - g_2 - 0 \pmod{n}.$$

Thus, $(n-g_i)\dagger_{\prec_2} \equiv -\left(g_i\dagger_{\prec_1}\right) \pmod{n}$. Since all $g_i\dagger_{\prec_1}$ must be distinct modulo n, so are all $-g_i\dagger_{\prec_1}$. Thus, $\prec_2$ gives us n distinct daggers and is therefore a constructive ordering. □

Proposition 2.7. *Let $\prec_1$ be a constructive ordering $0 \prec_1 g_2 \prec_1 g_3 \prec_1 \cdots \prec_1 g_n$. Then the ordering $\prec_2$ corresponding to $0 \prec_2 g_n \prec_2 \cdots \prec_2 g_3 \prec_2 g_2$ is also a constructive ordering.*

Proof. Observe that for $2 \le i \le n$

$$\begin{aligned} g_i\dagger_{\prec_2} &= 0 + g_n + g_{n-1} + \cdots + g_i \\ &= (0 + g_n + g_{n-1} + \cdots + g_i + g_{i-1} + \cdots + g_2) - (g_{i-1} + \cdots + g_2) \\ &= \frac{n}{2} - g_{i-1}\dagger_{\prec_1}. \end{aligned}$$

Since all of the elements $g_{i-1}\dagger_{\prec_1}$ are unique mod n, then so are the terms $g_i\dagger_{\prec_2} = \frac{n}{2} - g_{i-1}\dagger_{\prec_1}$. Thus by Proposition 1.4(c), $\prec_2$ is a constructive ordering. □

2.3. **Examples.** Here we will compute some examples of dagger sets and verify whether certain orderings are constructive or not, to help the reader see how the above observations interact with these sets.

Example 2.8. We will naively determine if the ordering $\prec = (0,3,1,4,7,2,5,6)$ is a constructive ordering for $\mathbb{Z}/8\mathbb{Z}$. To begin, we must calculate the daggers for each element of

$\mathbb{Z}/8\mathbb{Z}$, to construct $D_{\prec}(\mathbb{Z}/8\mathbb{Z})$. All calculations are reduced modulo 8:

$$\begin{aligned}
&0\dagger = 0 \pmod 8\\
&3\dagger = 0+3 = 3 \pmod 8\\
&1\dagger = 0+3+1 = 4 \pmod 8\\
&4\dagger = 0+3+1+4 = 8 \equiv 0 \pmod 8\\
&7\dagger = 0+3+1+4+7 = 15 \equiv 7 \pmod 8\\
&2\dagger = 0+3+1+4+7+2 = 17 \equiv 1 \pmod 8\\
&5\dagger = 0+3+1+4+7+2+5 = 22 \equiv 6 \pmod 8\\
&6\dagger = 0+3+1+4+7+2+5+6 = 28 \equiv 4 \pmod 8
\end{aligned}$$

Thus, the dagger set of $\mathbb{Z}/8\mathbb{Z}$ under $\prec$ is $D_{\prec}(\mathbb{Z}/8\mathbb{Z}) = \{0,1,3,4,6,7\}$. Since this set is missing 2 and 5, $D_{\prec}(\mathbb{Z}/8\mathbb{Z}) \neq \mathbb{Z}/8\mathbb{Z}$, and so $\prec$ is not a constructive ordering on $\mathbb{Z}/8\mathbb{Z}$.

Example 2.9. To show Propositions 1.4(a) and 1.4(c) in action, we will determine if

$$\prec = (1,0,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25)$$

is a constructive ordering on $\mathbb{Z}/26\mathbb{Z}$. As with the previous example, we will calculate the daggers of $\mathbb{Z}/26\mathbb{Z}$ under $\prec$:

$$\begin{aligned}
&1\dagger = 1\\
&0\dagger = 1+0 = 1
\end{aligned}$$

We can stop here, and apply Proposition 1.4(c), since $1\dagger = 0\dagger$. As well, Proposition 1.4(a) also directly shows that $\prec$ is not a constructive ordering on $\mathbb{Z}/26\mathbb{Z}$.

Example 2.10. We will attempt to find a counterexample to Proposition 1.4(b) by checking each ordering on $\mathbb{Z}/3\mathbb{Z}$ to see if a constructive ordering exists. We have 6 orderings to check:

$$\begin{gathered}
(0,1,2),(0,2,1)\\
(1,0,2),(1,2,0)\\
(2,0,1),(2,1,0)
\end{gathered}$$

By Proposition 1.4(a), we know that if $\prec$ is a constructive ordering, then it must start with 0. Thus, we only have two orderings to check. For the first ordering, $(0,1,2)$, $0\dagger = 0$, $1\dagger = 1$, and $2\dagger = 1+2 = 3 \equiv 0 \pmod 3$, and so the resulting dagger set is $\{0,1\}$. This means this ordering is not a constructive ordering on $\mathbb{Z}/3\mathbb{Z}$. For the second ordering, $0\dagger = 0$, $2\dagger = 2$, and $1\dagger = 2+1 = 3 \equiv 0 \pmod 3$. Thus, the resulting dagger set is $\{0,2\}$, and again the ordering is not a constructive ordering on $\mathbb{Z}/3\mathbb{Z}$. Therefore, there are no constructive orderings on $\mathbb{Z}/3\mathbb{Z}$.

3. Calculating Constructive Orderings

In this section we compute the number of constructive orderings on $\mathbb{Z}/n\mathbb{Z}$ for even values of n such that $n \leq 16$. We will see that the number of orderings grows quickly and becomes increasingly complex to compute as n grows. As well, the algorithm used to calculate these orderings will be discussed.

3.1. **Number of Constructive Orderings.** For use in this section, we will define the set

$$C(G) := \{< \mid < \text{ is a constructive ordering on } G\}.$$

As well, we use $|C(G)|$ to denote the number of constructive orderings on G. We have the following values for $|C(\mathbb{Z}/n\mathbb{Z})|$:

n	$\lvert C(\mathbb{Z}/n\mathbb{Z})\rvert$
2	1
4	2
6	4
8	24
10	288
12	3856
14	89328
16	2755968

TABLE 2. The number of constructive orderings for $\mathbb{Z}/n\mathbb{Z}$, $n \leq 16$, n even

This sequence corresponds to OEIS sequence A141599[3].

As an example, we include the complete list of constructive orderings on $\mathbb{Z}/n\mathbb{Z}$ for $n \leq 6$:

n	List of constructive orderings for $\mathbb{Z}/n\mathbb{Z}$
2	(0,1)
4	(0,1,2,3) (0,3,2,1)
6	(0,1,4,3,2,5) (0,2,5,3,1,4) (0,4,1,3,5,2) (0,5,2,3,4,1)

TABLE 3. All constructive orderings on $\mathbb{Z}/n\mathbb{Z}$, $n \leq 6$, n even

In order to compute the orderings which belong in $C(\mathbb{Z}/n\mathbb{Z})$, we wrote a basic C++ program that used some of the facts outlined in the propositions above to restrict our search in an otherwise brute force algorithm (for example, we only considered orderings beginning with 0 instead of all possible orderings). The computation times in the table below are for running our algorithm on a machine with an Intel® Core™ i7-4790 CPU @ 3.60GHz, multithreaded with 8 cores.

n	Time to calculate $\lvert C(\mathbb{Z}/n\mathbb{Z})\rvert$ (in seconds)[1]
2	$\ll 1$
4	$\ll 1$
6	$\ll 1$
8	$\ll 1$
10	0.02
12	0.93
14	148.54
16	26805.91

Table 4. Times to calculate the number of constructive orderings on $\mathbb{Z}/n\mathbb{Z}$ on a modern machine.

One sees that computation time grows very quickly with n. The running time for $n = 16$ was close to 8 hours, and we expect that this time would continue to increase at a rapid rate.

We know that the total number of orderings on a finite group G is $|G|!$, and the total number of orderings on G which begin with the identity is $(|G|-1)!$ (which by Proposition 1.4(a), we assume gives us a decent starting point for potential constructive orderings). In the table below, we compare these quantities with $|C(\mathbb{Z}/n\mathbb{Z})|$.

n	$\frac{\lvert C(\mathbb{Z}/n\mathbb{Z})\rvert}{n!}$	$\frac{\lvert C(\mathbb{Z}/n\mathbb{Z})\rvert}{(n-1)!}$
2	$\frac{1}{2} = 50\%$	$\frac{1}{1} = 100\%$
4	$\frac{2}{24} \approx 8\%$	$\frac{2}{3} \approx 33\%$
6	$\frac{4}{720} \approx 0.55\%$	$\frac{4}{120} \approx 3.33\%$
8	$\frac{24}{40320} \approx 0.060\%$	$\frac{24}{5040} \approx 0.48\%$
10	$\frac{288}{3628800} \approx 0.0079\%$	$\frac{288}{362880} \approx 0.079\%$

Table 5. Ratios of the number of constructive orderings to the total number of orderings on $\mathbb{Z}/n\mathbb{Z}$, $n \leq 10$, n even

From Table 5, it quickly becomes apparent that constructive orderings grow very sparse in the space of possible orderings for $\mathbb{Z}/n\mathbb{Z}$, even when we only look at orderings which start with the identity. Thus, we are motivated to find more strict criteria for constructive orderings which are easily computable.

[1] It is fun to note that similar results were calculated in [1], in which the authors state that "An IBM 7090 prepared (our results for $n = 2$ through $n = 10$) in 72 seconds...". This illustrates both how far computation power has come since the mid-60s, as well as the sheer difficulty this problem faces for larger values of n.

3.2. **Algorithm Discussion.** The algorithm used to calculate the number of constructive orderings for a particular value of n is presented in pseduocode, as well as in full in Appendix A.

$numOrderings \leftarrow 0$;
$currentPermutation \leftarrow (0, 1, 2, 3, \ldots, n-1)$;
while *currentPermutation not* $(0, n-1, n-2, \ldots, 2, 1)$ **do**
  $permutationSum \leftarrow 0$;
  foreach *element e in currentPermutation* **do**
    $permutationSum \leftarrow permutationSum + e \pmod{n}$;
    if *permutationSum is 0 or has been seen before* **then**
      break;
    end
    if *End of currentPermutation* **then**
      $numOrderings \leftarrow numOrderings + 1$;
    end
  end
  $currentPermutation \leftarrow nextPermutation(currentPermutation)$;
end

Algorithm 1: Pseudocode of algorithm used to find the number of constructive orderings

Acknowledgements

This research was funded in part by the Natural Sciences and Engineering Research Council of Canada. The author would like to thank the referees for their diligent work, as well as his project advisor, Dr. Amy Feaver, for her gracious continuing support throughout this project. The author would also like to thank the West Coast Number Theory Conference of 2017 for providing assistance in proving Theorem 2.1, as well as The King's University and The King's University Department of Computing Science for the opportunity and facilities required to conduct this research.

References

[1] E.N. Gilbert. Latin squares which contain no repeated digrams. *SIAM Review*, 7(2):189–198, 4 1965. URL: http://www.jstor.org/stable/2027267.

[2] B Gordon. Sequences in groups with distinct partial products. *Pacific J. Math*, 11(4):1309–1313, 1961. URL: https://projecteuclid.org/euclid.pjm/1103036916.

[3] OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences (2019), https://oeis.org/A141599.

Student biographies

Zackary Baker: (*Corresponding author:* zack@zackb.io) Zackary Baker graduated from The King's University in 2018 with a B.Sc. in Computing Science. He is currently continuing to pursue his research on constructive orderings while tutoring and working as a lab assistant at The King's University. He plans to begin graduate studies in Computer Science in the winter.

Appendix A C++ Algorithm

This is the algorithm in full used to calculate the results in Section 3, written in C++

```
/*
Copyright 2019 Zackary Baker

Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to
    deal in the Software without restriction, including without limitation the
     rights to use, copy, modify, merge, publish, distribute, sublicense, and/
    or sell copies of the Software, and to permit persons to whom the Software
     is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all
     copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
    THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
     LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
    FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
    DEALINGS IN THE SOFTWARE.
*/

#include <stdlib.h> //used for atoi
#include <stdio.h> //used for printf
#include <unistd.h> //used for sysconf
#include <chrono> //used for timekeeping variables
#include <vector> //used for the vector datatype
#include <pthread.h> // used for pthread_create, pthread_join
#include <algorithm> //used for std::next_permutation


//typedef used to make definition of large variables more readable
typedef  unsigned long largeNum;

//function prototypes
largeNum factorial(int);
int* lookupOrdering(int, largeNum);
void* threadProcessorFunc(void*);
int verifyOrdering(int*, int);


//definition of the struct used to pass information to each thread
struct Thread_Param{
  int id; // the id of the thread, beginning at 0
  int n; // the size of the group Z/nZ
  largeNum partitionSize;// the number of orderings for the thread to process
  int* firstOrdering;//a pointer to an array corresponding to the first
    ordering the thread will process
  //default constructor; not used
  Thread_Param(){}
  //main constructor used; takes individual values and sets the corresponding
    members of the struct
```

```
  Thread_Param(int id, int n, largeNum partitionSize){
    //set the member variables passed in
    this->id = id;
    this->n = n;
    this->partitionSize = partitionSize;
    //calculate the ordering this thread will begin calculation with
    this->firstOrdering = lookupOrdering(n, id*partitionSize);
  }
  //deconstructor; frees firstOrdering
  ~Thread_Param(){
    delete [] firstOrdering;
  }

};


/**
* The main method of the program. This program calculates the number of constructive orderings for the integers mod n
*/
int main(int argc, char const *argv[])
{
  //exit and print usage information if the program is run with 0 args
  if(argc==1){
    fprintf(stderr, "USAGE: %s n threadMult\n", argv[0]);
    return 1;
  }
  //get program parameters from the command line as the program is executed
  int n = atoi(argv[1]);
  int threadMult = 1;
  //set threadMult if provided by the user
  if(argc==3){
    threadMult = atoi(argv[2]);
  }

  //calculate the total number of threads to run based on the multiplier provided by the user and the total number of online processors at the time of program execution
  int maxThreads = threadMult*sysconf(_SC_NPROCESSORS_ONLN);

  printf("n is %d, threadMult is %d, total threads to create is %d\n", n, threadMult, maxThreads);
  //the total number of orderings to calculate; Since we only care about orderings that start with zero, we only have (n-1)! orderings to check, and since each constructive ordering of the form (0, k, ...), k<n/2 has exactly one corresponding constructive ordering (0,n-k,...), we only need to check the first half of the orderings, hence the division by 2
  largeNum orderingCount = factorial(n-1)/2;

  //if there arent an evenly divisible number of orderings per thread, we can't continue
  if(orderingCount%maxThreads!=0){
```

```
    fprintf(stderr,"[ERROR] The number of orderings (%ld) is not evenly divisible across the number of threads (%d)\n", orderingCount, maxThreads);
    return 2;
  }
  //each thread will process exactly partitionSize orderings
  largeNum partitionSize = orderingCount/maxThreads;

  //create an array of thread objects
  pthread_t threads[maxThreads];


  //create the variables used for timekeeping
  std::chrono::time_point<std::chrono::system_clock> start, end;
  start = std::chrono::system_clock::now();

  //launch each thread
  for(int i=0;i<maxThreads;i++){
    Thread_Param* tp = new Thread_Param(i, n, partitionSize);
    int threadStatus = pthread_create(&threads[i], NULL, threadProcessorFunc,(void*)tp);
    if(threadStatus!=0){// if there is an error creating the thread, exit
      fprintf(stderr, "[ERROR] Error creating thread %d\n", i);
      return 3;
    }
  }

  largeNum constructiveOrderings = 0;

  for(int i=0;i<maxThreads;i++){
    void* results = NULL;
    pthread_join(threads[i], &results);
    constructiveOrderings+=(largeNum)(results);
  }

  //multiply the final result by 2, to compensate for the fact that only the first half of orderings are checked.
  constructiveOrderings*=2;

  end = std::chrono::system_clock::now();
  std::chrono::duration<double> timeTaken = end-start;
  printf("FINISHED - Total constructive orderings: %ld - Time taken: %f\n", constructiveOrderings, timeTaken.count());


  return 0;
}

/**
* A simple recursive implementation of the factorial function
* INPUT:
    - n: the value to calculate the factorial of
  OUTPUT:
    returns n!
```

```
*/
largeNum factorial(int n){
  if(n>1){
    return factorial(n-1)*n;
  }
  else{
    return 1;
  }
}

/*
* This method calculates the lexicographical ordering based on a given index
   for the integers mod n, minus the first element.
  Example:
  All permutations of the integers mod 3 are listed as follows, in
   lexicographical order:
  (0,1,2)
  (0,2,1)
  (1,0,2)
  (1,2,0)
  (2,0,1)
  (2,1,0)

  Indexing these in this order gives the following relation:
  0 -> (0,1,2)
  1 -> (0,2,1)
  2 -> (1,0,2)
  3 -> (1,2,0)
  4 -> (2,0,1)
  5 -> (2,1,0)

  Thus, lookupOrdering(3,4), for example, would return [0,1], which is [2,0,1]
    without the first element.

  INPUT:
    - n: the size of the group
    - orderingIndex: the index of the ordering we are interested in
  OUTPUT:
    - the ordering with index orderingIndex, in lexicographical order, minus
   the first element
*/
int* lookupOrdering(int n, largeNum orderingIndex){
  //tuple is an array which holds the factorial representation of
   orderingIndex
  int* tuple = new int[n-1];
  for(int i=0;i<n-1;i++){
    //calculate the ith coefficient of orderingIndex
    int coefficient = orderingIndex/factorial(n-1-i);
    tuple[i] = coefficient;
    //reduce orderingIndex to calculate the next coefficient
    orderingIndex-=factorial(n-1-i)*coefficient;
  }

  //create a standard vector of size n with values 0 through n-1
```

```
  std::vector<int> orderedList;
  for(int i=0;i<n;i++){
    orderedList.push_back(i);
  }
  int* ordering = new int[n];
  //for each element in the sequence
  for(int i=0;i<n-1;i++){
    //set each position to the value of the ordered list, indexed by the
   values in tuple created above, then remove that value to prevent
   duplicates
    ordering[i] = orderedList.at(tuple[i]);
    orderedList.erase(orderedList.begin()+tuple[i]);
  }
  //manually add the final value to the ordering.
  ordering[n-1] = orderedList.front();

  //simply create a new array from the old array of one size smaller to remove
    the first element
  int* nl = new int[n-1];
  for(int i=0;i<n-1;i++){
    nl[i] = ordering[i+1];
  }

  delete [] ordering;
  delete [] tuple;

  return nl;


}


/**
* This method is used by each thread to begin calculation.
* INPUT:
*   - args: a pointer which is cast into a Thread_Param pointer, used to
   access parameters intended for the method.
  OUTPUT:
    - the number of constructive orderings in the range for the thread, cast
   into a void pointer
*/
void* threadProcessorFunc(void* args){
  //cast args into a Thread_Param struct
  Thread_Param* params = (Thread_Param*)(args);

  //extract the members of the params struct
  int n = params->n;
  largeNum partitionSize = params->partitionSize;
  int* currentOrdering = params->firstOrdering;
  //constructiveOrderings holds the count of how many constructive orderings
   are in the range of the thread
  largeNum constructiveOrderings = 0;

  do{//iterate through each permutation in range and verify each of them.
```

```cpp
    constructiveOrderings+=verifyOrdering(currentOrdering, n);
    partitionSize--;
    if(partitionSize==0){
      break;
    }
  }while(std::next_permutation(currentOrdering, currentOrdering+(n-1)));

  //clean up allocated memory and return constructiveOrderings
  delete params;
  return (void*)(constructiveOrderings);


}

/**
  This method determines if a given ordering is a constructive ordering.
  INPUT:
    - ordering: an integer array of size n-1 representing a potential constructive ordering.
    - n: the size of the group
  OUTPUT:
    returns 1 if ordering is a constructive ordering, and 0 otherwise.
*/
int verifyOrdering(int* ordering, int n){

  int total = 0;// the running total sum
  bool* elementsSeen = new bool[n](); // an array to keep track of whether each index has been seen previously
  for(int i=0;i<n-1;i++){
    total += ordering[i];
    total %= n;
    if(total==0 || elementsSeen[total]==true || (total==n/2 && i!=n-2)){//if the running total is 0 or this total has been seen before or the total is n/2 and is not the final total, return 0 (false)
      delete [] elementsSeen;
      return 0;
    }
    else{//otherwise indicate that we have seen this total for future passes
      elementsSeen[total] = true;
    }
  }

  //if no total is seen twice, and the other conditions are met, this is a constructive ordering
  delete [] elementsSeen;
  return 1;
}
```

constructiveOrderingsMultithread.cpp