Efficient Algorithms for Ranking, Unranking, and Generating Stacks of Pancakes and Burnt Pancakes

Joe Sawada[a], Aaron Williams[b]

[a]*School of Computer Science, University of Guelph, Canada. Research supported by NSERC.*
[b]*Department of Mathematics and Statistics, McGill University, Canada.*

Abstract

Stacks of pancakes and burnt pancakes can be modeled by permutations and signed permutations, with the 'flip' operation corresponding to prefix-reversal and complementing prefix-reversal, respectively. Recently it was shown that stacks can be rearranged in all possible ways with successive stacks differing by a single flip by greedily flipping the minimum or maximum number of pancakes (Sawada and Williams, *Greedy Flipping of Pancakes and Burnt Pancakes*, Discrete Applied Mathematics, 2016). This article translates the four Gray code orders into efficient algorithms for ranking, unranking, and generating the underlying permutations and signed permutations. This builds on the results of Zaks who provided these results for pancakes using the minimum-flip strategy (Zaks, *A New Algorithm for Generation of Permutations*, BIT Numerical Mathematics, 1984).

Keywords: Gray code, permutations, signed permutations, prefix-reversal, ranking, unranking

1. Introduction

In this paper we investigate two orders for the $n!$ permutations of $\{1, 2, \dots, n\}$ and two orders for the $2^n n!$ signed permutations of $\{1, 2, \dots, n\}$. A *signed permutation* is a permutation in which each symbol x is either positive or negative, which are denoted x and $\bar{x}$, respectively. We write both permutations and signed permutations using one-line notation.

The four orders originate from a recent article [1] and are illustrated below on the left for $n = 4$:

$$
\begin{array}{ll}
1234, 2134, 3124, 1324, 2314, 3214, 4123, \dots & 2, 3, 2, 3, 2, 3, 4, \dots \\
1234, 4321, 2341, 1432, 3412, 2143, 4123, 3214, 2314, \dots & 4, 3, 4, 3, 4, 3, 4, 2, \dots \\
1234, \bar{1}234, \bar{2}134, 2134, \bar{1}\bar{2}34, 1\bar{2}34, 2\bar{1}34, \bar{2}\bar{1}34, \bar{3}124, \dots & 1, 2, 1, 2, 1, 2, 1, 3, \dots \\
1234, \bar{4}\bar{3}\bar{2}\bar{1}, 234\bar{1}, 1\bar{4}\bar{3}\bar{2}, 34\bar{1}\bar{2}, 21\bar{4}\bar{3}, 4\bar{1}\bar{2}\bar{3}, 321\bar{4}, \bar{1}\bar{2}\bar{3}\bar{4}, 4321, \dots & 4, 3, 4, 3, 4, 3, 4, 3, 4, \dots
\end{array}
$$

The significance of the four orders is they are all *flip Gray codes*. This means that each successive (signed) permutation is obtained from the previous (signed) permutation by an operation known as a 'flip'. When applied to a permutation a *flip* of length k reverses the order of first k symbols, and in addition when it is applied to a signed permutation the sign of these symbols is also reversed. The specific flips that are used to create the four orders are illustrated above on the right.

The flip terminology comes from the representation of permutations and signed permutations as stacks of pancakes and burnt pancakes, respectively. In this analogy, flip of length k corresponds to taking a spatula, and flipping over the top k pancakes, thereby reversing their order, and their up/down

Email addresses: `jsawada@uoguelph.ca` (Joe Sawada), `haron@uvic.ca` (Aaron Williams) orientation in the case of burnt pancakes which are burnt on one side. This analogy is illustrated below for a permutation and its stack of pancakes (left), and for a signed permutation and its stack of burnt pancakes (right).

$632514 \overset{5}{\rightarrow} 152364$
$\bar{6}3\bar{2}514 \overset{5}{\rightarrow} \bar{1}\bar{5}2\bar{3}64$

The beauty of the four orders is that they can all be generated by simple greedy algorithms. In particular, the first order is obtained by greedily flipping the smallest number of symbols that produces a new permutation. To explain one step of this process in more detail, observe that the order begins 1234, 2134, 3124, 1324, 2314, 3214. At this stage the algorithm examines the last permutation 3214 and tries to determine the smallest flip that will create a permutation that does not already appear in the list. The algorithm does not flip the first two symbols, since 2314 already appears in the list. Similarly, it does not flip the first three symbols, since 1234 already appears in the list. Thus, the algorithm considers flipping the first four symbols, and this time it succeeds since 4123 has not already appeared in the list. The new permutation 4123 is added to the end of the list and the algorithm continues by trying to flip the smallest number of symbols in it. Amazingly, this process does not get stuck until all $n!$ permutations are created. The resulting order of permutations was first discovered by Zaks [3].

We refer to the algorithm that produces Zaks's order as the *minimum flip order for permutations*. The recent article by Sawada and Williams [1] showed that the greedy strategy also works when using the largest number of possible flips, and moreover, both strategies also work for signed permutations. In fact, the three other orders illustrated above are the *maximum flip order for permutations*, the *minimum flip order for signed permutations*, and the *maximum flip order for signed permutations*, respectively.

Although the greedy algorithms are elegant, they are not efficient since all previous (signed) permutations must be stored. The purpose of this article is to translate the four orders into efficient algorithms. More specifically we provide three efficient algorithms for each of the orders:

- *Generation*. These algorithms are used for creating one (signed) permutation after another as quickly as possible and with little additional memory.
- *Ranking*. These algorithms determine the *rank* (ie position) of each (signed) permutation in the list.
- *Unranking*. These algorithms determine the (signed) permutation that has a particular rank.

Prior to this article Zaks provided all three algorithms for the minimum flip order for permutations [3]. Collectively, the provided algorithms help bolster the value of the corresponding orders in the context of interconnection networks, where both the *pancake network* and *burnt pancake network* are popular in applications. See Siegel [2] for further discussion of interconnection networks.

The remainder of this document is organized as follows. Section 2 introduces notation. Then Section 3 discusses the ranking and unranking algorithms for the minimum flip order of permutations originally given by Zaks. Sections 5, 4, and 6 repeat these results for the three other orders. Section 7 provides efficient generation algorithms for all four orders. Finally, Section 8 summarizes our results. Full implementations are provided in C in the Appendix.

2. Notation

In this paper we are concerned with providing Gray code listings for the permutations and signed permutations for the set $\mathbf{S} = \{1, 2, 3, \ldots, n\}$. However, for the proofs we must consider arbitrary sets of n elements. For example the six permutations of $\mathbf{S} = \{1, 2, 4\}$ are $\{124, 142, 214, 241, 412, 421\}$ and the eight signed permutations of $\mathbf{S} = \{1, 4\}$ are $\{14, 41, \bar{1}4, 4\bar{1}, 1\bar{4}, \bar{4}1, \bar{1}\bar{4}, \bar{4}\bar{1}\}$ where $\bar{p}$ denotes $-p$.

Let the set of (unsigned) permutations of an n-set be denoted by $\mathbb{P}(n)$. Given $\mathbf{p} = p_1p_2p_3 \cdots p_n \in \mathbb{P}(n)$, we will use the following notation for $1 \le j \le n$:

- $\mathsf{flip}_j(\mathbf{p}) = p_jp_{j-1} \cdots p_1p_{j+1} \cdots p_n$ denotes a flip (prefix-reversal) of length j, and
- $\mathbf{p}(j) = p_{j+1} \cdots p_np_1 \cdots p_{j-1}$ denotes a full rotation to the left by j positions followed by the removal of the element p_j.

Let the set of signed permutations of an n-set be denoted by $\overline{\mathbb{P}}(n)$. Given $\mathbf{p} = p_1p_2p_3 \cdots p_n \in \overline{\mathbb{P}}(n)$, we will use the following notation for $1 \le j \le n$:

- $\overline{\mathsf{flip}}_j(\mathbf{p}) = \bar{p}_j\bar{p}_{j-1} \cdots \bar{p}_1p_{j+1} \cdots p_n$ denotes a flip (complemented prefix reversal) of length j, and
- $\mathsf{flipSign}(\mathbf{p}) = \bar{p}_1\bar{p}_2\bar{p}_3 \cdots \bar{p}_n$ flips the sign of every element.
- $\mathbf{p}'(j) = \bar{p}_{j+1} \cdots \bar{p}_np_1p_2 \cdots p_{j-1}$, and
- $\bar{\mathbf{p}}(j) = p_{j+1} \cdots p_n\bar{p}_1\bar{p}_2 \cdots \bar{p}_{j-1} = \mathsf{flipSign}(\mathbf{p}'(i))$.

For both signed and unsigned permutations we will use the following notation for a permutation $\mathbf{p}$:

- $\mathbf{p} \cdot n$ denotes the concatenation of the symbol n to the permutation $\mathbf{p}$.
- $\mathsf{rotLeft}(\mathbf{p}, j) = p_{j+1} \cdots p_np_1 \cdots p_j$ denotes a full rotation to the left by j positions,
- $\mathsf{rotRight}(\mathbf{p}, j) = p_{n-j+1} \cdots p_np_1 \cdots p_{n-j}$ denotes a full rotation to the right by j positions,

Consider a sequence of unsigned permutations $\rho = \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m$ and an integer sequence $\phi = f_1, f_2, \ldots f_{m-1}$ for some $m > 1$. We say that ϕ is the *flip-sequence* for ρ if $\mathbf{p}_{i+1} = \mathsf{flip}_{f_i}(\mathbf{p}_i)$ for $1 \le i \le m-1$. Similarly, if $\rho = \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m$ is a sequence of m signed permutations then $\phi = f_1, f_2, \ldots f_{m-1}$ is said to be the *flip-sequence* for ρ if $\mathbf{p}_{i+1} = \overline{\mathsf{flip}}_{f_i}(\mathbf{p}_i)$ for $1 \le i \le m-1$.

When describing sequences, we let x^k denote k repeated concatenations of the sequence x. For example $(1, 3)^4 = 1, 3, 1, 3, 1, 3, 1, 3$.

3. Minimum Flips for Permutations

In [1], the following cyclic prefix-reversal Gray code for permutations is presented. It is equivalent to one that was initially discovered by Zaks [3], except that we use prefix-reversals as opposed to suffix-reversals.

Let $\mathbf{p} = p_1p_2p_3 \cdots p_n \in \mathbb{P}(n)$. Let $\mathbf{p}(i) = p_{i+1} \cdots p_np_1 \cdots p_{i-1}$ where $1 \le i \le n$. Then $\mathbf{Min}(\mathbf{p})$, defined below, produces a prefix-reversal Gray code for permutations:

$$\mathbf{Min}(\mathbf{p}) = \begin{cases} \mathbf{p} & \text{if } n = 1 \\ \mathbf{Min}(\mathbf{p}(n)) \cdot p_n,\ \mathbf{Min}(\mathbf{p}(n-1)) \cdot p_{n-1}, \ldots,\ \mathbf{Min}(\mathbf{p}(1)) \cdot p_1 & \text{if } n \ge 2. \end{cases} \quad (1)$$

Example 3.1. $\mathbf{Min}(1234) = \mathbf{Min}(123){\cdot}4,\ \mathbf{Min}(412){\cdot}3,\ \mathbf{Min}(341){\cdot}2,\ \mathbf{Min}(234){\cdot}1.$ *The full listing is given below (read down, then left to right). The length of the prefix-reversal to go from one permutation to the next is given in parentheses after each permutation.*

1 2 3 4 (2)	4 1 2 3 (2)	3 4 1 2 (2)	2 3 4 1 (2)
2 1 3 4 (3)	1 4 2 3 (3)	4 3 1 2 (3)	3 2 4 1 (3)
3 1 2 4 (2)	2 4 1 3 (2)	1 3 4 2 (2)	4 2 3 1 (2)
1 3 2 4 (3)	4 2 1 3 (3)	3 1 4 2 (3)	2 4 3 1 (3)
2 3 1 4 (2)	1 2 4 3 (2)	4 1 3 2 (2)	3 4 2 1 (2)
3 2 1 4 (4)	2 1 4 3 (4)	1 4 3 2 (4)	4 3 2 1 (4)

The permutations at the top of each column in this example are equivalent under rotation and each column has the same flip-sequence. If we ignore the final flip to return to the initial permutation, then the sequence σ_n given by Zaks [3] is the flip-sequence for $\mathbf{Min}(\mathbf{p})$:

$$\sigma_n = \begin{cases} 2 & \text{if } n = 2 \\ (\sigma_{n-1}, n)^{n-1}, \sigma_{n-1} & \text{if } n > 2. \end{cases}$$

3.1. Ranking

In this subsection we provide a ranking algorithm for the listing $\mathbf{Min}(123\cdots n)$. The algorithm corresponds to the one described by Zaks in [3] which uses suffix-reversals instead of prefix-reversals. We provide a similar presentation here. Similar ideas will be applied when we discuss the minimum flip approach for signed permutations.

Let $\mathsf{Rank}(p_1p_2p_3\cdots p_n)$ denote the position of the permutation $\mathbf{p} = p_1p_2p_3\cdots p_n$ in the listing $\mathbf{Min}(123\cdots n)$. Observe from the recurrence in (1) that all $(n-p_n)(n-1)!$ permutations ending with one of $n, n-1, n-2, \ldots, p_n+1$ will precede $\mathbf{p}$ in the ordering. So the crux of the problem is to determine the position of $\mathbf{p}$ among permutations ending with p_n. From Lemma **??** we can deduce that the first permutation ending with p_n is $(p_n{+}1)(p_n{+}2)\cdots n12\cdots(p_n{-}1)$. Notice that by subtracting p_n from each element in this permutation (modulo n) we obtain $123\cdots n-1$. Thus, we can reduce our problem to recursively solving for $\mathsf{Rank}(q_1q_2q_3\cdots q_{n-1})$ where $q_i = (p_i - p_n) \bmod n$. This leads to the following algorithm:

$$\mathsf{Rank}(p_1p_2p_3\cdots p_n) = \begin{cases} 1 & \text{if } n = 1 \\ (n-p_n)(n-1)! \ + \ \mathsf{Rank}(q_1q_2q_3\cdots q_{n-1}) & \text{if } n > 1, \end{cases}$$

where $q_i = (p_i - p_n) \bmod n$.

As an example, we consider the recursive decomposition for computing the rank of 261453 in the ordering $\mathbf{Min}(123456)$:

$$\begin{aligned} \mathsf{Rank}(261453) &= 3\cdot 5! + \mathsf{Rank}(53412) \\ &= 3\cdot 5! + 3\cdot 4! + \mathsf{Rank}(3124) \\ &= 3\cdot 5! + 3\cdot 4! + 0\cdot 3! + \mathsf{Rank}(312) \\ &= 3\cdot 5! + 3\cdot 4! + 0\cdot 3! + 1\cdot 2! + \mathsf{Rank}(12) \\ &= 3\cdot 5! + 3\cdot 4! + 0\cdot 3! + 1\cdot 2! + 0\cdot 1! + \mathsf{Rank}(1) \\ &= \ 360 \ + \ 72 \ + \ 0 \ + \ 2 \ + \ 0 \ + \ 1 \ = \ 435. \end{aligned}$$

3.2. Unranking

Finding the permutation $\mathbf{p} = p_1p_2p_3\cdots p_n$ at rank $rank$ in the listing $\mathbf{Min}(123\cdots n)$ also follows from the discussion in [3]. The first step is to determine the value of the last element p_n in the permutation. Based on the recurrence in (1) we have $p_n = n - x$ where $x = \lfloor \frac{rank-1}{(n-1)!} \rfloor$. Then recursion is applied to find the permutation $q_1q_2q_3 \dots q_{n-1}$ at rank $rank - x(n-1)!$ in $\mathbf{Min}(123 \cdots n-1)$. By observing that the first permutation that ends with p_n in $\mathbf{Min}(123 \cdots n)$ is given by $(p_n{+}1)(p_n{+}2) \cdots n12 \cdots (p_n{-}1)$ we can apply a direct mapping of these symbols to $123 \cdots n-1$ to obtain $p_1p_2 \cdots p_{n-1}$. Pseudocode that follows this approach is given in Algorithm 1. Since the work, not counting the recursive call, is $O(n)$, the permutation $\mathbf{p}$ at position $rank$ in the listing $\mathbf{Min}(123 \cdots n)$ can be computed using $O(n^2)$ basic operations.

Algorithm 1 Computing the permutation at position $rank$ in the listing $\mathbf{Min}(123 \cdots n)$, $n \geq 1$

```
1: function UNRANK($rank, n$) returns permutation
2:     if $n = 1$ then return 1
3:     $x \leftarrow \lfloor \frac{rank-1}{(n-1)!} \rfloor$
4:     $q_1q_2 \cdots q_{n-1} \leftarrow$ UNRANK($rank - x(n-1)!,\ n-1$)
5:     $p_n \leftarrow n - x$
6:     for $j \leftarrow 1$ to $n-1$ do $p_j \leftarrow 1 + (q_j + p_n - 1) \bmod n$
7:     return $p_1p_2p_3 \cdots p_n$
```

As an example, we determine the permutation $p_1p_2p_3p_4p_5p_6$ at rank 435 in the listing $\mathbf{Min}(123456)$. First, compute $x = \lfloor \frac{435-1}{120} \rfloor = 3$, which means that $p_6 = 6 - 3 = 3$. Recursively, the permutation at rank $435 - 3(120) = 75$ in $\mathbf{Min}(12345)$ is found to be $q_1q_2q_3q_4q_5 = 53412$. However, from the recurrence in (1), the first permutation ending with $p_6 = 3$ will start with 45612 (and not the 12345 solved for recursively). Thus, we map $1 \rightarrow 4$, $2 \rightarrow 5$, $3 \rightarrow 6$, $4 \rightarrow 1$ and $5 \rightarrow 2$ to obtain $p_1p_2p_3p_4p_5p_6$ = 261453.

4. Minimum Flips for Signed Permutations

The results in this section for signed permutations mirror the results for unsigned permutations. Again, we begin by looking at an example, this time considering the greedy listing $\overline{\mathsf{MinGreedy}}(123)$. The length of the flip to go from one signed permutation to the next is given in parentheses after each signed permutation.

Example 4.1. $\overline{\mathsf{MinGreedy}}(123)$ *(read down, then left to right):*

$1\,2\,3$ (1)	$\bar{3}\,1\,2$ (1)	$\bar{2}\,\bar{3}\,1$ (1)	$\bar{1}\,\bar{2}\,\bar{3}$ (1)	$3\,\bar{1}\,\bar{2}$ (1)	$2\,3\,\bar{1}$ (1)
$\bar{1}\,2\,3$ (2)	$3\,1\,2$ (2)	$2\,\bar{3}\,1$ (2)	$1\,\bar{2}\,\bar{3}$ (2)	$\bar{3}\,\bar{1}\,\bar{2}$ (2)	$\bar{2}\,3\,\bar{1}$ (2)
$\bar{2}\,1\,3$ (1)	$\bar{1}\,\bar{3}\,2$ (1)	$3\,\bar{2}\,1$ (1)	$2\,\bar{1}\,\bar{3}$ (1)	$1\,3\,\bar{2}$ (1)	$\bar{3}\,2\,\bar{1}$ (1)
$2\,1\,3$ (2)	$1\,\bar{3}\,2$ (2)	$\bar{3}\,\bar{2}\,1$ (2)	$\bar{2}\,\bar{1}\,\bar{3}$ (2)	$\bar{1}\,3\,\bar{2}$ (2)	$3\,2\,\bar{1}$ (2)
$\bar{1}\,\bar{2}\,3$ (1)	$3\,\bar{1}\,2$ (1)	$2\,3\,1$ (1)	$1\,2\,\bar{3}$ (1)	$\bar{3}\,1\,\bar{2}$ (1)	$\bar{2}\,\bar{3}\,\bar{1}$ (1)
$1\,\bar{2}\,3$ (2)	$\bar{3}\,\bar{1}\,2$ (2)	$\bar{2}\,3\,1$ (2)	$\bar{1}\,2\,\bar{3}$ (2)	$3\,1\,\bar{2}$ (2)	$2\,\bar{3}\,\bar{1}$ (2)
$2\,\bar{1}\,3$ (1)	$1\,3\,2$ (1)	$\bar{3}\,2\,1$ (1)	$\bar{2}\,1\,\bar{3}$ (1)	$\bar{1}\,\bar{3}\,\bar{2}$ (1)	$3\,\bar{2}\,\bar{1}$ (1)
$\bar{2}\,\bar{1}\,3$ (3)	$\bar{1}\,3\,2$ (3)	$3\,2\,1$ (3)	$2\,1\,\bar{3}$ (3)	$1\,\bar{3}\,\bar{2}$ (3)	$\bar{3}\,\bar{2}\,\bar{1}$ (3)

In each column of this example, note that the last element of each signed permutation is the same. Additionally, each column has the same flip-sequence. If we ignore the final flip to return to the initial signed permutation, then we will prove that the following flip-sequence $\overline{\sigma}_n$ is the same sequence used by $\overline{\mathsf{MinGreedy}}(\mathbf{p})$:

$$\overline{\sigma}_n = \begin{cases} 1 & \text{if } n = 1 \\ (\overline{\sigma}_{n-1}, n)^{2n-1}, \overline{\sigma}_{n-1} & \text{if } n > 1. \end{cases}$$

However, to formally prove that this flip-sequence is the one used by $\overline{\mathsf{MinGreedy}}(\mathbf{p})$, we need to further understand the ordering of signed permutations produced.

Fortunately, by studying the greedy listing from the example and using the recurrence $\overline{\sigma}_n$, we can deduce a simple recurrence to list all signed permutations. Let $\mathbf{p} = p_1p_2p_3\cdots p_n$ denote a signed permutation of an arbitrary n-set $\mathbf{S}$. Recall the notation from Section 2 for $1 \le i \le n$:

- $\mathbf{p}'(i) = \bar{p}_{i+1}\cdots\bar{p}_n p_1 p_2\cdots p_{i-1}$, and
- $\bar{\mathbf{p}}(i) = \mathsf{flipSign}(\mathbf{p}'(i))p_{i+1}\cdots p_n\bar{p}_1\bar{p}_2\cdots\bar{p}_{i-1}$.

We will show that following recurrence for $\overline{\mathbf{Min}}(\mathbf{p})$ produces the same listing as $\overline{\mathsf{MinGreedy}}(n)$:

$$\overline{\mathbf{Min}}(\mathbf{p}) = \begin{cases} p_1, \bar{p}_1 & \text{if } n = 1 \\ \\ \overline{\mathbf{Min}}(\mathbf{p}'(n))\cdot p_n,\ \overline{\mathbf{Min}}(\mathbf{p}'(n-1))\cdot p_{n-1}, \ldots,\ \overline{\mathbf{Min}}(\mathbf{p}'(1))\cdot p_1, & \text{if } n \ge 2. \\ \overline{\mathbf{Min}}(\bar{\mathbf{p}}(n))\cdot \bar{p}_n,\ \overline{\mathbf{Min}}(\bar{\mathbf{p}}(n-1))\cdot \bar{p}_{n-1}, \ldots,\ \overline{\mathbf{Min}}(\bar{\mathbf{p}}(1))\cdot \bar{p}_1 & \end{cases} \tag{2}$$

As an example:

$$\overline{\mathbf{Min}}(123) = \overline{\mathbf{Min}}(12)\cdot 3,\ \overline{\mathbf{Min}}(\bar{3}1)\cdot 2,\ \overline{\mathbf{Min}}(\bar{2}\bar{3})\cdot 1,\ \overline{\mathbf{Min}}(\bar{1}\bar{2})\cdot \bar{3},\ \overline{\mathbf{Min}}(3\bar{1})\cdot \bar{2},\ \overline{\mathbf{Min}}(23)\cdot \bar{1}.$$

4.1. Ranking

In this subsection we provide a ranking algorithm for the listing $\overline{\mathbf{Min}}(123\cdots n)$. The rank of a signed permutation $\mathbf{p} = p_1p_2\cdots p_n$ in the listing $\overline{\mathbf{Min}}(123\cdots n)$ follows directly from the recurrence given in (2), and is similar in spirit to the approach used in Section 3.1.

$$\overline{\mathsf{Rank}}(p_1p_2\cdots p_n) = \begin{cases} 1 & \text{if } n = 1 \text{ and } p_1 = 1 \\ 2 & \text{if } n = 1 \text{ and } p_1 = \bar{1} \\ (n - p_n) \quad \cdot 2^{n-1}(n-1)! \ +\ \overline{\mathsf{Rank}}(q_1q_2\cdots q_{n-1}) & \text{if } n > 1 \text{ and } p_n > 0 \\ (2n - |p_n|)\cdot 2^{n-1}(n-1)! \ +\ \overline{\mathsf{Rank}}(q_1q_2\cdots q_{n-1}) & \text{if } n > 1 \text{ and } p_n < 0, \end{cases}$$

where $|q_i| = (|p_i| - |p_n|) \bmod n$ and the sign of q_i is *negative* if and only if (1) the sign of p_i is the same as the sign of p_n and $|p_n| < |p_i|$ or (2) the sign of p_i is different from the sign of p_n and $|p_n| > |p_i|$. Since $q_1q_2\cdots q_{n-1}$ can easily be computed using $O(n)$ operations, a straightforward implementation of this recurrence allows the rank to be computed using $O(n^2)$ operations.

The mapping of p_i to q_i is perhaps best understood with an example. Consider $p_1p_2\cdots p_6 = 1\bar{4}3\bar{6}52$ and the following map table constructed given that $p_6 = 2$:

p_i	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	$\bar{6}$	1	2	3	4	5	6
q_i	$\bar{5}$	-	1	2	3	4	5	-	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$

From this table: $p_1 = 1$ implies $q_1 = 5$, $p_2 = \bar{4}$ implies $q_2 = 2$, $p_3 = 3$ implies $q_3 = \bar{1}$, $p_4 = \bar{6}$ implies $q_4 = 4$, and $p_5 = 5$ implies $q_5 = \bar{3}$.

As a full example of the recursive ranking process, we consider the recursive decomposition for computing the rank of $1\bar{4}3\bar{6}52$ in the ordering $\overline{\mathbf{Min}}(123456)$:

$$\begin{aligned}
\overline{\mathsf{Rank}}(1\bar{4}3\bar{6}52) &= 4\cdot 2^5\cdot 5! + \overline{\mathsf{Rank}}(52\bar{1}4\bar{3})\\
&= 4\cdot 2^5\cdot 5! + 7\cdot 2^4\cdot 4! + \overline{\mathsf{Rank}}(2\bar{4}31)\\
&= 4\cdot 2^5\cdot 5! + 7\cdot 2^4\cdot 4! + 3\cdot 2^3\cdot 3! + \overline{\mathsf{Rank}}(\bar{1}3\bar{2})\\
&= 4\cdot 2^5\cdot 5! + 7\cdot 2^4\cdot 4! + 3\cdot 2^3\cdot 3! + 4\cdot 2^2\cdot 2! + \overline{\mathsf{Rank}}(21)\\
&= 4\cdot 2^5\cdot 5! + 7\cdot 2^4\cdot 4! + 3\cdot 2^3\cdot 3! + 4\cdot 2^2\cdot 2! + 1\cdot 2^1\cdot 1! + \overline{\mathsf{Rank}}(\bar{1})\\
&= \quad 15360 \;+\; 2688 \;+\; 144 \;+\; 32 \;+\; 2 \;+\; 2 \;=\; 18228.
\end{aligned}$$

4.2. Unranking

Finding the signed permutation $\mathbf{p} = p_1p_2p_3\cdots p_n$ at rank $rank$ in the listing $\overline{\mathbf{Min}}(123\cdots n)$ follows a similar approach as the unsigned case. The first step is to determine the value of the last element p_n in the signed permutation. Let $x = \lfloor \frac{rank-1}{2^{n-1}(n-1)!} \rfloor$. Then based on the recurrence in (2), if $x < n$ then $p_n = n - x$ and otherwise $p_n = -(2n - x)$. Then recursion is applied to find the signed permutation $q_1q_2q_3\ldots q_{n-1}$ at rank $rank - x2^{n-1}(n-1)!$ in $\overline{\mathbf{Min}}(123\cdots n-1)$. Observe that the first signed permutation that ends with p_n in $\overline{\mathbf{Min}}(123\cdots n)$ is $\overline{(p_n{+}1)}\;\overline{(p_n{+}2)}\cdots\bar{n}12\cdots(p_n{-}1)$ if $p_n > 0$ and is $(p_n{+}1)(p_n{+}2)\cdots n\bar{1}\bar{2}\cdots\overline{(p_n{-}1)}$ if $p_n < 0$. Thus, by applying an appropriate mapping of these symbols to $123\cdots n$ (and the corresponding $\bar{1}\bar{2}\bar{3}\cdots\bar{n}$), we obtain $p_1p_2p_3\cdots p_{n-1}$ from the string returned by the recursive call. Pseudocode that follows this approach is given in Algorithm 2. The function $\textsc{sign}(x)$ returns 1 if x is positive and 0 otherwise. Since the work, not counting the recursive call, is $O(n)$, the signed permutation $\mathbf{p}$ at position $rank$ in the listing $\overline{\mathbf{Min}}(123\cdots n)$ can be computed using $O(n^2)$ operations.

Algorithm 2 Computing the signed permutation at position $rank$ in the listing $\overline{\mathbf{Min}}(12\cdots n)$, $n \geq 1$

```
1:  function $\overline{\textsc{UnRank}}(rank, n)$ returns signed permutation
2:      if $n = 1$ and $rank = 1$ then return 1
3:      if $n = 1$ and $rank = 2$ then return $-1$
4:      $x \leftarrow \lfloor \frac{rank-1}{2^{n-1}(n-1)!} \rfloor$
5:      $q_1q_2\cdots q_{n-1} \leftarrow \overline{\textsc{UnRank}}(rank - x2^{n-1}(n-1)!,\ n-1)$
6:      if $x < n$ then $p_n \leftarrow n - x$
7:      else $p_n \leftarrow -(2n - x)$
8:      for $j \leftarrow 1$ to $n-1$ do
9:          $p_j \leftarrow 1 + (|q_j| + |p_n| - 1) \bmod n$
10:         if ($\textsc{sign}(p_n) = \textsc{sign}(q_j)$ and $p_j > p_n$) or ($\textsc{sign}(p_n) \neq \textsc{sign}(q_j)$ and $p_j < p_n$) then $p_j \leftarrow -p_j$
11:     return $p_1p_2\cdots p_n$
```

As an example, we determine the signed permutation $p_1p_2\cdots p_6$ at rank 18228 in the listing $\overline{\mathbf{Min}}(123456)$. First, we compute $x = \lfloor \frac{18228-1}{32\cdot 120} \rfloor = 4$, which means that $p_6 = 6 - 4 = 2$. Recursively, the signed permutation at rank $18228 - 4(3840) = 2868$ in $\overline{\mathbf{Min}}(12345)$ is found to be $52\bar{1}4\bar{3}$. However, from the recurrence in (2), the first signed permutation ending with $p_6 = 2$ will start with $\bar{3}\bar{4}\bar{5}\bar{6}1$ (and not the 12345 solved for recursively). Thus, we map:

$$1 \to \bar{3}, \quad 2 \to \bar{4}, \quad 3 \to \bar{5}, \quad 4 \to \bar{6}, \quad 5 \to 1$$

and similarly we would map

$$\bar{1} \to 3, \quad \bar{2} \to 4, \quad \bar{3} \to 5, \quad \bar{4} \to 6, \quad \bar{5} \to \bar{1}.$$

Applying these symbol mappings to $52\bar{1}4\bar{3}$ we obtain $p_1p_2p_3p_4p_5p_6 = 1\bar{4}3\bar{6}52$.

5. Maximum Flips for Permutations

In this section we study the maximum flip greedy algorithm and prove that it exhaustively lists all unsigned permutations by deriving an equivalent recursive formulation. We begin by looking at the greedy listings $\mathsf{MaxGreedy}(1234)$ and $\mathsf{MaxGreedy}(12345)$. The length of the flip to go from one permutation to the next is given in parentheses after each permutation.

Example 5.1. $\mathsf{MaxGreedy}(1234)$ *(read down, then left to right):*

1234 (4)	2314 (4)	3124 (4)
4321 (3)	4132 (3)	4213 (3)
2341 (4)	3142 (4)	1243 (4)
1432 (3)	2413 (3)	3421 (3)
3412 (4)	1423 (4)	2431 (4)
2143 (3)	3241 (3)	1342 (3)
4123 (4)	4231 (4)	4312 (4)
3214 (2)	1324 (2)	2134 (2)

Example 5.2. $\mathsf{MaxGreedy}(12345)$ *(read down, then left to right):*

12345 (5)	23415 (5)	34125 (5)	41235 (5)	23145 (5)	31425 (5)	14235 (5)	42315 (5)	31245 (5)	12435 (5)	24315 (5)	43125 (5)
54321 (4)	51432 (4)	52143 (4)	53214 (4)	54132 (4)	52413 (4)	53241 (4)	51324 (4)	54213 (4)	53421 (4)	51342 (4)	52134 (4)
23451 (5)	34152 (5)	41253 (5)	12354 (5)	31452 (5)	14253 (5)	42351 (5)	23154 (5)	12453 (5)	24351 (5)	43152 (5)	31254 (5)
15432 (4)	25143 (4)	35214 (4)	45321 (4)	25413 (4)	35241 (4)	15324 (4)	45132 (4)	35421 (4)	15342 (4)	25134 (4)	45213 (4)
34512 (5)	41523 (5)	12534 (5)	23541 (5)	14523 (5)	42531 (5)	23514 (5)	31542 (5)	24531 (5)	43512 (5)	31524 (5)	12543 (5)
21543 (4)	32514 (4)	43521 (4)	14532 (4)	32541 (4)	13524 (4)	41532 (4)	24513 (4)	13542 (4)	21534 (4)	42513 (4)	34521 (4)
45123 (5)	15234 (5)	25341 (5)	35412 (5)	45231 (5)	25314 (5)	35142 (5)	15423 (5)	45312 (5)	35124 (5)	15243 (5)	25431 (5)
32154 (4)	43251 (4)	14352 (4)	21453 (4)	13254 (4)	41352 (4)	24153 (4)	32451 (4)	21354 (4)	42153 (4)	34251 (4)	13452 (4)
51234 (5)	52341 (5)	53412 (5)	54123 (5)	52314 (5)	53142 (5)	51423 (5)	54231 (5)	53124 (5)	51243 (5)	52431 (5)	54312 (5)
43215 (3)	14325 (3)	21435 (3)	32145 (2)	41325 (3)	24135 (3)	32415 (3)	13245 (2)	42135 (3)	34215 (3)	13425 (3)	21345 (2)

In each example, observe that every second flip has length n. Thus, when deriving a recurrence for the sequence of flips required to generate the greedy maximum-flip listing, we start by deriving a recurrence for the length of every second flip. To derive this recurrence, the important flip to consider is the one happening at the bottom of each column in the examples. Observe that the flip lengths at the bottom of each column in the example for $\mathsf{MaxGreedy}(12345)$ correspond to every second flip length in the example for $\mathsf{MaxGreedy}(1234)$. Letting $\tau'_n = t_{n,1}, t_{n,2}, \ldots, t_{n,j}$ consider the following recurrence:

$$\tau'_{n+1} = \begin{cases} 2, 2 & \text{if } n+1 = 3 \\ n^n, t_{n,1}, n^n, t_{n,2}, \ldots, n^n, t_{n,j}, n^n & \text{if } n+1 > 3. \end{cases}$$

Lemma 5.3. *For $n \geq 3$, the number of elements in the sequence τ'_n is $\frac{n!}{2} - 1$.*

Proof. By induction. In the base case when $n = 3$, τ'_n has 2 elements and $\frac{3!}{2} - 1 = 2$. Inductively, it is easy to see that the number of elements in τ'_{n+1} is $(n+1) \cdot (\frac{(n)!}{2} - 1) + n = \frac{(n+1)!}{2} - 1$. $\square$

Using τ'_n, we will show that the sequence of flips used to create $\mathsf{MaxGreedy}(\mathbf{p})$ is given by σ'_n which is defined as follows:

$$\sigma'_n = \begin{cases} 2 & \text{if } n = 2 \\ n, t_{n,1}, n, t_{n,2}, \ldots, n, t_{n,m}, n & \text{if } n > 2. \end{cases}$$

Before we can prove this claim, we need to better understand the permutation ordering. Again, by considering the examples for $\mathsf{MaxGreedy}(1234)$ and $\mathsf{MaxGreedy}(12345)$, observe that the permutations in each column are closed under rotation and reversal: they form a *bracelet class*. The rotation effect is easily observed since:

- ▷ applying flip_n followed by flip_{n-1} rotates a permutation one position to the left and
- ▷ applying flip_{n-1} followed by flip_n rotates a permutation one position to the right.

These bracelet sequences form the crux of a recursive formulation for $\mathsf{MaxGreedy}(\mathbf{p})$. Define the *bracelet sequence* of permutation $\mathbf{p_1} = p_1p_2p_3 \cdots p_n$, where $n \geq 3$ as:

$$\mathsf{brace}(\mathbf{p_1}) = \mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \ldots, \mathbf{p_{2n}} \;\text{ such that }\; \mathbf{p_i} = \begin{cases} \mathsf{flip}_n(\mathbf{p_{i-1}}) & \text{if } i \text{ is even} \\ \mathsf{flip}_{n-1}(\mathbf{p_{i-1}}) & \text{if } i > 1 \text{ is odd.} \end{cases}$$

The permutation $\mathbf{p_1}$ is called the *representative* of the bracelet sequence $\mathsf{brace}(\mathbf{p_1})$. Since the order alternates flips of lengths n and $n-1$, the odd permutations $\mathbf{p_1}, \mathbf{p_3}, \mathbf{p_5}, \ldots, \mathbf{p_{2n-1}}$ are all rotations of $\mathbf{p_1}$ and the even permutations $\mathbf{p_2}, \mathbf{p_4}, \mathbf{p_6}, \ldots, \mathbf{p_{2n}}$ are all rotations of the reversal of $\mathbf{p_1}$ which is $\mathsf{flip}_n(\mathbf{p_1}) = \mathbf{p_2}$. Thus, the permutations in each bracelet sequence form a bracelet class.

5.1. Ranking

Observe that in every bracelet sequence generated by $\mathbf{Max}(123 \cdots n)$, the position of n in each permutation follows the sequence:

$$n, 1, n-1, 2, n-2, 3, \ldots 3, n-2, 2, n-1, 1, n.$$

Thus, given permutation $\mathbf{p} = p_1p_2 \cdots p_n$, where $p_j = n$, by rotating $\mathbf{p}$ to the right by $n-j$ positions to get $q_1q_2 \cdots q_n$, we obtain either the first or last permutation in its bracelet sequence. By recursively finding the rank x of $q_1q_2 \cdots q_{n-1}$, we can determine which it is: if x is odd, then $q_1q_2 \cdots q_n$ is the first permutation; otherwise it is the last permutation. In either case, $\mathbf{p}$ will be at a distance of $d = 2n - 2j$ from $q_1q_2 \cdots q_n$. Based on these observations, the recurrence for $\mathbf{Max}(\mathbf{p})$ and Remark **??**, we obtain the simple ranking procedure given in Algorithm 3. It is easy to see that this procedure requires $O(n^2)$ operations.

As an example, consider the rank of the permutation $\mathbf{p} = p_1p_2p_3p_4p_5 = 32451$ in the listing $\mathbf{Max}(12345)$. To rotate $n = 5$ into the rightmost position we rotate $\mathbf{p}$ to the right $n - j = 5 - 4 = 1$ positions to get $q_1q_2q_3q_4q_5 = 13245$. Recursively, $q_1q_2q_3q_4$ has rank 16 in the listing $\mathbf{Max}(1234)$. Since 16 is even, $\mathbf{p}$ will be $d = 10 - 8 = 2$ positions from the end of its bracelet sequence and hence has rank $nx - d = 5(16) - 2 = 78$.

5.2. Unranking

By applying the same recursive ideas as the corresponding ranking algorithm, Algorithm 4 returns the permutation $\mathbf{p}$ at position $rank$ in the listing $\mathbf{Max}(123 \cdots n)$. If the $rank$ is even, then the al-

Algorithm 3 Computing the rank of a permutation in $\mathbf{Max}(123\cdots n)$, $n \geq 2$

```
1: function MAXRANK($\mathbf{p} = p_1p_2\cdots p_n$) returns integer
2:     if $\mathbf{p} = 12$ then  return $1$
3:     if $\mathbf{p} = 21$ then  return $2$
4:     $d \leftarrow 2n - 2j$  (where $p_j = n$)
5:     $q_1q_2\cdots q_n \leftarrow \mathsf{rotRight}(\mathbf{p}, n-j)$
6:     $x \leftarrow$ MAXRANK($q_1q_2\cdots q_{n-1}$)
7:     if $x$ is EVEN then  return $nx - d$
8:     else  return $n(x-1) + d + 1$
```

gorithm finds the *last* permutation $\mathbf{p}' = p_1p_2p_3\cdots p_n$ in the bracelet sequence containing $\mathbf{p}$. Based on the recursive definition of the listing $\mathbf{Max}(\mathbf{p})$, $\mathbf{p}'$ will be the permutation at rank $2\lfloor\frac{rank-1}{2n}\rfloor + 2$ in $\mathbf{Max}(p_1p_2p_3\cdots p_{n-1})$, with the final element n concatenated to the end. From the discussion in the corresponding ranking algorithm, $\mathbf{p}$ will be at a distance $d = 2n - (rank \bmod 2n)$ from $\mathbf{p}'$; thus $\mathbf{p}$ can be obtained by rotating $\mathbf{p}'$ to left by $d/2$ positions. A similar argument applies when $rank$ is odd. It is easy to verify that this procedure requires $O(n^2)$ operations.

Algorithm 4 Computing the permutation at position $rank$ in the listing $\mathbf{Max}(123\cdots n)$, $n \geq 2$

```
1: function MAXUNRANK($rank, n$) returns permutation
2:     if $n = 2$ and $rank = 1$ then  return 12
3:     if $n = 2$ and $rank = 2$ then  return 21
4:     if $rank$ is EVEN then
5:         $p_1p_2\cdots p_{n-1} \leftarrow$ MAXUNRANK( $2\lfloor\frac{rank-1}{2n}\rfloor + 2, n-1$ )
6:         $d \leftarrow 2n - (rank \bmod 2n)$
7:     else
8:         $p_1p_2\cdots p_{n-1} \leftarrow$ MAXUNRANK( $2\lfloor\frac{rank-1}{2n}\rfloor + 1, n-1$ )
9:         $d \leftarrow (rank - 1) \bmod 2n$
10:    return $\mathsf{rotLeft}(\ p_1p_2\cdots p_{n-1}n\ , d/2)$
```

As an example, we find the permutation at rank 78 in the listing $\mathbf{Max}(12345)$. The first step is to find the permutation at the end of its bracelet sequence in the listing, since 78 is even. This permutation will be at position $2\lfloor\frac{78-1}{10}\rfloor + 2 = 16$ in the listing $\mathbf{Max}(1234)$. Recursively, we find this permutation to be 1324. Adding $n = 5$ to this permutation we obtain the permutation 13245 which will be $d = 10 - 8 = 2$ positions after our target permutation in the listing (and corresponding bracelet sequence). Finally, by rotating this permutation $d/2 = 1$ position to the left we obtain the permutation 32451 at rank 78.

6. Maximum Flips for Signed Permutations

In this section we study the maximum flip greedy algorithm and prove that it exhaustively lists all signed permutations by deriving an equivalent recursive formulation. We begin by looking at the greedy listing $\overline{\mathsf{MaxGreedy}}(123)$. The length of the flip to go from one signed permutation to the next is given in parentheses after each signed permutation.

Example 6.1. $\overline{\mathsf{MaxGreedy}}(123)$ *(read down, then left to right):*

123	(3)	$2\bar{1}3$	(3)	$\bar{1}\bar{2}3$	(3)	$\bar{2}13$	(3)
$\bar{3}\bar{2}\bar{1}$	(2)	$\bar{3}1\bar{2}$	(2)	$\bar{3}21$	(2)	$\bar{3}\bar{1}2$	(2)
$23\bar{1}$	(3)	$\bar{1}3\bar{2}$	(3)	$\bar{2}31$	(3)	132	(3)
$1\bar{3}\bar{2}$	(2)	$2\bar{3}1$	(2)	$\bar{1}\bar{3}2$	(2)	$\bar{2}\bar{3}\bar{1}$	(2)
$3\bar{1}\bar{2}$	(3)	$3\bar{2}1$	(3)	312	(3)	$32\bar{1}$	(3)
$21\bar{3}$	(2)	$\bar{1}2\bar{3}$	(2)	$\bar{2}\bar{1}\bar{3}$	(2)	$1\bar{2}\bar{3}$	(2)
$\bar{1}\bar{2}\bar{3}$	(3)	$\bar{2}1\bar{3}$	(3)	$12\bar{3}$	(3)	$2\bar{1}\bar{3}$	(3)
321	(2)	$3\bar{1}2$	(2)	$3\bar{2}\bar{1}$	(2)	$31\bar{2}$	(2)
$\bar{2}\bar{3}1$	(3)	$1\bar{3}2$	(3)	$2\bar{3}\bar{1}$	(3)	$\bar{1}\bar{3}\bar{2}$	(3)
$\bar{1}32$	(2)	$\bar{2}3\bar{1}$	(2)	$13\bar{2}$	(2)	231	(2)
$\bar{3}12$	(3)	$\bar{3}2\bar{1}$	(3)	$\bar{3}\bar{1}\bar{2}$	(3)	$\bar{3}\bar{2}1$	(3)
$\bar{2}\bar{1}3$	(1)	$1\bar{2}3$	(1)	213	(1)	$\bar{1}23$	(1)

As with the maximum flip greedy algorithm for unsigned permutations, observe that every second flip (starting from the first signed permutation) has length n. Thus, we start by deriving a recurrence for the length of every second flip, starting from the second signed permutation in the ordering. To derive this recurrence, the important flip to consider is the one happening at the bottom of each column in the examples. The flip lengths at the bottom of each column in the example for $\overline{\mathsf{MaxGreedy}}(123)$ correspond to every second flip in the sequence for $\overline{\mathsf{MaxGreedy}}(12)$ which is given by 2,**1**,2,**1**,2,**1**,2,**1** when considered circularly.

Let $\overline{\tau'}_n = t_{n,1}, t_{n,2}, \ldots, t_{n,m}$ and consider the following recurrence:

$$\overline{\tau'}_{n+1} = \begin{cases} 1,1,1 & \text{if } n+1=2 \\ n^{2n+1}, t_{n,1}, n^{2n+1}, t_{n,2}, \ldots, n^{2n+1}, t_{n,m}, n^{2n+1} & \text{if } n+1>2. \end{cases}$$

Lemma 6.2. *For $n \geq 2$, the number of elements in the sequence $\overline{\tau'}_n$ is $2^{n-1}n! - 1$.*

Proof. By induction. In the base case when $n = 2$, $\overline{\tau'}_n$ has 3 elements and $2^1 2! - 1 = 3$. Inductively, it is easy to see that the number of elements in $\overline{\tau'}_{n+1}$ is $(2n+2)\cdot(2^{n-1}n! - 1) + 2n + 1 = 2^n(n+1)! - 1$. □

Using $\overline{\tau'}_n$, we will show that the sequence of flips used to create $\overline{\mathsf{MaxGreedy}}(\mathbf{p})$ is given by $\overline{\sigma'}_n$ which is defined as follows:

$$\overline{\sigma'}_n = \begin{cases} 1 & \text{if } n=1 \\ n, t_{n,1}, n, t_{n,2}, \ldots, n, t_{n,j}, n & \text{if } n>1. \end{cases}$$

Before we can prove this claim, we need to better understand the signed permutation ordering. Observe that each column in the example for $\overline{\mathsf{MaxGreedy}}(123)$ has *signed* groupings similar to the unsigned case, but of size $4n$ compared to $2n$ in the unsigned case. Define the *signed bracelet sequence* of a signed permutation $\mathbf{p_1} = p_1p_2p_3\cdots p_n$, where $n \geq 2$ as:

$$\overline{\mathsf{brace}}(\mathbf{p_1}) = \mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \ldots, \mathbf{p_{4n}} \text{ such that } \mathbf{p_i} = \begin{cases} \overline{\mathsf{flip}}_n(\mathbf{p_{i-1}}) & \text{if } i \text{ is even} \\ \overline{\mathsf{flip}}_{n-1}(\mathbf{p_{i-1}}) & \text{if } i>1 \text{ is odd.} \end{cases}$$

As with the unsigned case, the signed permutation $\mathbf{p_1}$ is called the *representative* of the signed bracelet sequence $\mathsf{brace}(\mathbf{p_1})$. Observe that applying a flip of size n followed by a flip of size $n-1$ to any signed permutation rotates the values one position to the left, changing the sign of the element that moved to the end. Repeating such a rotation n times, we obtain the original signed permutation with all the signs flipped. Repeating such a rotation $2n$ times returns us to the original starting signed permutation. From these observations we make the following remarks.

Remark 6.3. *The last signed permutation in a signed bracelet sequence* $\overline{\mathsf{brace}}(\mathbf{p_1})$ *is* $\overline{\mathsf{flip}}_{n-1}(\mathbf{p_1})$.

Lemma 6.4. *If* $\overline{\mathsf{brace}}(\mathbf{p_1}) = \mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \ldots, \mathbf{p_{4n}}$ *where* $n \geq 2$ *then* $\mathsf{flipSign}(\mathbf{p_i}) = \mathbf{p_{2n+i}}$ *for* $1 \leq i \leq 2n$.

Proof. Consider a signed permutation $\mathbf{p} = p_1 p_2 \cdots p_n$. The result of applying a flip of size n followed by a flip of size $n-1$ is $p_2 p_3 \cdots p_n \bar{p}_1$ which is a rotation of $\mathbf{p}$ to the left and flipping the sign of the new last element. By repeatedly applying these two successive operations n times, the resulting permutation is $\bar{p}_1 \bar{p}_2 \cdots \bar{p}_n$. Thus, by the definition of a signed bracelet sequence, when i is odd, $\mathsf{flipSign}(\mathbf{p_i}) = \mathbf{p_{2n+i}}$ for $1 \leq i \leq 2n$.

The result of applying a flip of size $n-1$ followed by a flip of size n is $\bar{p}_n p_1 p_2 \cdots p_{n-1}$ which is a rotation of $\mathbf{p}$ to the right and flipping the sign of the new first element. By repeatedly applying these two successive operations n times, the resulting permutation is $\bar{p}_1 \bar{p}_2 \cdots \bar{p}_n$. Thus, by the definition of a signed bracelet sequence, when i is even, $\mathsf{flipSign}(\mathbf{p_i}) = \mathbf{p_{2n+i}}$ for $1 \leq i \leq 2n$. □

Remark 6.5. *There are exactly two signed permutations in* $\overline{\mathsf{brace}}(p_1 p_2 p_3 \cdots p_n)$ *that end with* p_n*, namely* $p_1 p_2 p_3 \cdots p_n$ *and* $\bar{p}_{n-1} \cdots \bar{p}_3 \bar{p}_2 \bar{p}_1 p_n$*, and they differ by* flip_{n-1}.

Using the definition for signed bracelet sequences, we arrive at a recurrence for the sequence $\overline{\mathbf{Max}}(\mathbf{p})$ similar to the one for the unsigned case. If $\mathbf{p} = p_1 p_2 p_3 \cdots p_n$ is a signed permutation, then:

$$\overline{\mathbf{Max}}(\mathbf{p}) = \begin{cases} p_1, \bar{p}_1 & \text{if } n = 1 \\ \overline{\mathsf{brace}}(\mathbf{q_1} \cdot p_n), \overline{\mathsf{brace}}(\mathbf{q_3} \cdot p_n), \overline{\mathsf{brace}}(\mathbf{q_5} \cdot p_n), \ldots, \overline{\mathsf{brace}}(\mathbf{q_{m-1}} \cdot p_n) & \text{if } n \geq 2, \end{cases} \tag{3}$$

where $\overline{\mathbf{Max}}(p_1 p_2 p_3 \cdots p_{n-1}) = \mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_m}$. The following lemma shows that $\mathbf{m} = 2^{n-1}(n-1)!$.

Lemma 6.6. *The number of elements in the sequence* $\overline{\mathbf{Max}}(p_1 p_2 p_3 \cdots p_n)$ *is* $2^n n!$.

Proof. By induction. The base case is clearly satisfied for $n = 1$. Inductive Hypothesis: For $n \geq 1$, assume the claim is true. Consider $\overline{\mathbf{Max}}(p_1 p_2 p_3 \cdots p_{n+1})$. By the inductive hypothesis $\overline{\mathbf{Max}}(p_1 p_2 p_3 \cdots p_n)$ has $2^n n!$ elements which is an even number since $n \geq 1$. Thus, from the recursive definition, $\overline{\mathbf{Max}}(p_1 p_2 p_3 \cdots p_{n+1})$ is the concatenation of $2^{n-1} n!$ signed bracelet sequences of length $4(n+1)$. Thus, the total number of signed permutations in the sequence $\overline{\mathbf{Max}}(p_1 p_2 p_3 \cdots p_{n+1})$ is $2^{n+1}(n+1)!$. □

Our goal is to show that $\overline{\mathsf{MaxGreedy}}(\mathbf{p})$ and $\overline{\mathbf{Max}}(\mathbf{p})$ are equivalent flip Gray code listings for signed permutations.

6.1. Ranking

The ranking strategy for a signed permutation $\mathbf{p} = p_1 \cdots p_n$ is similar to the unsigned case: determine the distance from $\mathbf{p}$ to either the first or last signed permutation in its signed bracelet sequence. However, for the signed case, there are two main differences:

1. The $2n-2j$ operations required to shift the symbol p_j whose absolute value is n to the last position correspond to a right rotation of length $n-j$ **and** a flipping of the signs for the first $n-j$ symbols in the resulting rotated signed permutation.

2. If the symbol p_j is $-n$, then we need to perform an additional n right rotations (which corresponds to flipping the signs of each symbol) to obtain either the first or last signed permutation in the signed bracelet sequence.

With these two modifications and applying the recurrence for $\overline{\mathbf{Max}}(\mathbf{p})$, we obtain the ranking procedure given in Algorithm 5. This algorithm requires $O(n^2)$ basic operations.

Algorithm 5 Computing the rank of a signed permutation in $\overline{\mathbf{Max}}(123\cdots n)$, $n \geq 1$

```
1: function $\overline{\text{MAXRANK}}(\mathbf{p} = p_1p_2\cdots p_n)$ returns integer
2:     if $\mathbf{p} = 1$ then  return $1$
3:     if $\mathbf{p} = -1$ then  return $2$
4:     $d \leftarrow 2n - 2j$     ▷ where $p_j = n$
5:     $q_1q_2\cdots q_n \leftarrow \text{rotRight}(\mathbf{p}, n-j)$
6:     $\text{flipSign}(q_1q_2\cdots q_{n-j})$
7:     if $p_j < 0$ then
8:         $\text{flipSign}(q_1q_2\cdots q_n)$
9:         $d \leftarrow d + 2n$
10:    $x \leftarrow \overline{\text{MAXRANK}}(q_1q_2\cdots q_{n-1})$
11:    if $x$ is EVEN then  return $2nx - d$
12:    else  return $2n(x-1) + d + 1$
```

As an example, consider the rank of the signed permutation $\mathbf{p} = p_1p_2p_3p_4 = \bar{1}3\bar{4}2$ in the ordering $\overline{\mathbf{Max}}(1234)$. To shift the element p_j, such that $|p_j| = n = 4$, into the right most position we perform a right rotation $n - j = 4 - 3 = 1$ positions and flip the sign of the first $n - j = 1$ positions. The resulting signed permutation $q_1q_2q_3q_4 = \bar{2}\bar{1}3\bar{4}$ will be $d = 2n - 2j = 2$ from the original signed permutation. The signed permutation $\bar{2}\bar{1}3\bar{4}$, in turn, will be $2n$ additional positions away from its corresponding signed permutation with all its signs flipped, namely $21\bar{3}4$. This latter signed permutation will be either the first or last signed permutation in a signed bracelet sequence used by the recurrence for $\overline{\mathbf{Max}}(1234)$. Recursively, $21\bar{3}$ has rank $x = 6$ in the listing $\overline{\mathbf{Max}}(123)$. Since 6 is even, $\mathbf{p}$ will be $d = 2n - 2j + 2n = 10$ positions from the end of its bracelet sequence and hence has rank $2nx - d = 8(6) - 10 = 38$.

6.2. Unranking

By applying the same recursive ideas as the corresponding ranking algorithm, Algorithm 6 returns the signed permutation $\mathbf{p}$ at position $rank$ in the listing $\overline{\mathbf{Max}}(123\cdots n)$. By computing the distance $d = (rank - 1) \bmod 4n$ that $\mathbf{p}$ must be from its signed bracelet sequence representative, consider two cases:

1. If $d \geq 2n$ recursively find the *last* signed permutation in its signed bracelet sequence which is the signed permutation at rank $2\lfloor \frac{rank-1}{4n} \rfloor + 2$ in the listing $\overline{\mathbf{Max}}(n-1)$ with the symbol n appended to the last position. The target signed permutation $\mathbf{p}$ will then be at distance of $4n - d - 1$ before this signed permutation in the corresponding signed bracelet sequence.

2. If $d < 2n$ recursively find the representative signed permutation in the signed bracelet sequence which is the signed permutation at rank $2\lfloor \frac{rank-1}{4n} \rfloor + 1$ in the listing $\overline{\mathbf{Max}}(n-1)$ with the symbol n appended to the last position.

Based on the definition of a signed bracelet sequence, it is easy to simulate the operations from the first (or last) signed permutation in the class to obtain the target signed permutation $\mathbf{p}$ as is illustrated in Algorithm 6. This algorithm uses $O(n^2)$ basic operations.

Algorithm 6 Computing the signed permutation at position $rank$ in the listing $\overline{\mathbf{Max}}(123\cdots n)$, $n \geq 1$

```
function $\overline{\text{MAXUNRANK}}(rank, n)$ returns signed permutation
    if $n = 1$ and $rank = 1$ then  return 1
    if $n = 1$ and $rank = 2$ then  return -1
    $d \leftarrow (rank - 1) \bmod 4n$
    if $d \geq 2n$ then
        $d \leftarrow 4n - d - 1$
        $p_1p_2\cdots p_{n-1} \leftarrow \overline{\text{MAXUNRANK}}(\ 2\lfloor \frac{rank-1}{4n} \rfloor + 2, n-1\ )$
    else
        $p_1p_2\cdots p_{n-1} \leftarrow \overline{\text{MAXUNRANK}}(\ 2\lfloor \frac{rank-1}{4n} \rfloor + 1, n-1\ )$
    if $d$ is EVEN then
        $p_1p_2\cdots p_{d/2} \leftarrow$ flipSign$(p_1p_2\cdots p_{d/2})$
        return  rotLeft( $p_1p_2\cdots p_{n-1}n$ , $d/2$)
    else
        $p_1p_2\cdots p_n \leftarrow$ rotRight( $\overline{\text{flip}}_n(p_1p_2\cdots p_{n-1}n)$ , $\lfloor d/2 \rfloor$)
        $p_1p_2\cdots p_{\lfloor d/2 \rfloor} \leftarrow$ flipSign$(p_1p_2\cdots p_{\lfloor d/2 \rfloor})$
        return  $p_1p_2\cdots p_n$
```

As an example, we find the signed permutation $\mathbf{p} = p_1p_2p_3p_4$ at rank 38 in $\overline{\mathbf{Max}}(1234)$. Since each signed bracelet sequence for $n = 4$ has $4n = 16$ elements, the signed permutation we are looking for will be $d = (38 - 1) \bmod 16 = 5$ positions from the first permutation in its corresponding signed bracelet sequence which is at rank 33. Since this is the 3rd bracelet sequence in the recurrence for $\overline{\mathbf{Max}}(1234)$, we recursively find the 5th signed permutation in $\overline{\mathbf{Max}}(123)$ which is $3\bar{1}\bar{2}$. Thus, $\mathbf{p}$ will be $d = 5$ permutations after $3\bar{1}\bar{2}4$ in its signed bracelet sequenceing. This permutation is obtained by performing a signed flip of size n to get $\bar{4}21\bar{3}$ followed by a right rotation of $\lfloor d/2 \rfloor = 2$ positions to get $1\bar{3}\bar{4}2$. Finally, we flip the sign of the first $\lfloor d/2 \rfloor = 2$ elements to obtain $\mathbf{p} = \bar{1}3\bar{4}2$.

7. Efficient Generation

In this section we present efficient implementations of the minimum-flip and maximum-flip greedy Gray codes for both signed and unsigned permutations. Algorithm 7 outlines a simple iterative procedure that will generate the specified Gray code provided there is an iterative function NEXT-σ that will produce the next flip in the corresponding flip-sequence. When the sequence is exhausted, the function NEXT-σ is assumed to return 0.

Algorithm 7 Iterative approach to list $\mathbf{Min}(\mathbf{p})$, $\overline{\mathbf{Min}}(\mathbf{p})$, $\mathbf{Max}(\mathbf{p})$, or $\overline{\mathbf{Max}}(\mathbf{p})$

```
procedure ITERATIVEGEN($\mathbf{p}$)
    repeat
        VISIT($\mathbf{p}$)
        $j \leftarrow$ NEXT-$\sigma$
        $\mathbf{p} \leftarrow$ flip$_j(\mathbf{p})$        ▷ use $\overline{\text{flip}}_j(\mathbf{p})$ for the signed case
    until $j = 0$
```

The loop-free functions for NEXT-σ, outlined in Algorithm 8, can be used to produce the flip-sequences σ_n for each Gray code discussed in the previous four sections. Each function uses an array of counters $c_0, c_1, \ldots, c_{n+1}$ initialized to 0, and an array of flip lengths $f_0, f_1, \ldots, f_{n+1}$ with each f_i initialized to i. A formal proof of correctness for the minimum-flip case for permutations is provided in [3].

Similar techniques can be used to prove the correctness for the other three sequences, and are omitted from this paper. A complete C implementation for these algorithms (including the ranking and unranking algorithms) is provided in the appendix.

Algorithm 8 Iterative approaches for producing the sequences σ_n, $\overline{\sigma}_n$, σ'_n, and $\overline{\sigma'}_n$,

```
function NEXT-$\sigma$ returns integer                ▷ for $\sigma_n$
    $x \leftarrow f_2$
    $f_2 \leftarrow 2$
    $c_x \leftarrow c_x + 1$
    if $x = 2$ or $c_x = x{-}1$ then
        $c_x \leftarrow 0$
        $f_x \leftarrow f_{x+1}$
        $f_{x+1} \leftarrow x + 1$
    if $x = n$ then return 0
    return $x$
```

```
function NEXT-$\sigma$ returns integer                ▷ for $\overline{\sigma}_n$
    $x \leftarrow f_1$
    $f_1 \leftarrow 1$
    $c_x \leftarrow c_x + 1$
    if $x = 1$ or $c_x = 2x{-}1$ then
        $c_x \leftarrow 0$
        $f_x \leftarrow f_{x+1}$
        $f_{x+1} \leftarrow x + 1$
    if $x = n$ then return 0
    return $x$
```

```
function NEXT-$\sigma$ returns integer                ▷ for $\sigma'_n$
    $x \leftarrow f_n$
    $f_n \leftarrow n$
    $c_x \leftarrow c_x + 1$
    if $x = n$ or $c_x = x$ then
        $c_x \leftarrow 0$
        $f_x \leftarrow f_{x-1}$
        $f_{x-1} \leftarrow x - 1$
    if $x = 1$ then return 0
    return $x$
```

```
function NEXT-$\sigma$ returns integer                ▷ for $\overline{\sigma'}_n$
    $x \leftarrow f_n$
    $f_n \leftarrow n$
    $c_x \leftarrow c_x + 1$
    if $x = n$ or $c_x = 2x{+}1$ then
        $c_x \leftarrow 0$
        $f_x \leftarrow f_{x-1}$
        $f_{x-1} \leftarrow x - 1$
    if $x = 0$ then return 0
    return $x$
```

Using a standard array representation for the permutation, this framework will result in a CAT algorithm for the *minimum-flip* algorithms, since the average flip length is constant.

Theorem 7.1. *The listing* $\mathbf{Min}(\mathbf{p})$ *can be generated in constant amortized time.*

Theorem 7.2. *The listing* $\overline{\mathbf{Min}}(\mathbf{p})$ *can be generated in constant amortized time.*

The average flip length is $O(n)$ in the *maximum-flip* algorithms, so the corresponding generation algorithms will run in $O(n)$ amortized time using the standard array representation. However, we can achieve CAT generation algorithms by using two doubly linked lists to represent the (signed) permutations. To explain how this is done for permutations, recall that applying the operations flip_n and flip_{n-1} successively to a permutation is equivalent to a left rotation. Similarly, applying the operations flip_{n-1} and flip_n corresponds to a right rotation. Thus, a bracelet sequence $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_{2n}}$ satisfies the following:

$$\mathbf{p_{2i+1}} = \mathsf{rotLeft}(\mathbf{p_{2i-1}}, 1) \quad \text{and} \quad \mathbf{p_{2i+2}} = \mathsf{rotRight}(\mathbf{p_{2i}}, 1) \quad \text{for all} \quad 1 \leq i < n.$$

By maintaining pointers to the first and last elements, each of the individual rotations can be implemented in constant time. Thus, by maintaining two linked lists initialized to $\mathbf{p_1}$ and $\mathbf{p_2}$ and a pointer to indicate the current list, we can generate each bracelet sequence in constant amortized time. To complete the $\mathbf{Max}(\mathbf{p})$ generation algorithm, we must consider how to transition between bracelet classes. These transitions can trivially be done in linear time; however, since it is only required for every $2n$ permutations, the overall algorithm will be CAT.

Theorem 7.3. *The listing* $\mathrm{Max}(\mathbf{p})$ *can be generated in constant amortized time using two linked lists.*

The same approach works for signed permutations. The only difference is that left and right rotations require the sign of the last and first symbols to be complemented in the resulting permutations, respectively.

Theorem 7.4. *The listing* $\overline{\mathrm{Max}}(\mathbf{p})$ *can be generated in constant amortized time using two linked lists.*

8. Concluding Remarks

To do. (Mention FUN journal paper with successor rules.)

References

[1] Sawada, J., Williams, A.: Greedy flipping of pancakes and burnt pancakes. Discrete Applied Mathematics accepted (2016)

[2] Siegel, J.: Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies. McGraw-Hill (1990)

[3] Zaks, S.: A new algorithm for generation of permutations. BIT Numerical Mathematics 24(2), 196–204 (1984)

9. Appendix

```
//-------------------------------------------------------
// GENERATING (SIGNED) PERMUTATIONS BY MIN or MAX FLIPS
// Research by: Aaron Williams, Joe Sawada
//-------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>

#define MAX_N 21

int MAX=0, MIN=0, SIGNED=0, GEN=0, RANK=0, UNRANK=0;
int n, type, a[MAX_N], sign[MAX_N], f[MAX_N], c[MAX_N];
long long int rank, total, pow2[MAX_N], factorial[MAX_N];

//--------------------------------------------------------------
void Input() {
int i;

      printf(" ----------------------\n");
      printf(" Permutation Generation \n");
      printf(" ----------------------\n");
      printf(" 1. Max Flip \n");
      printf(" 2. Min Flip\n");
      printf(" 3. Max Flip (Signed) \n");
      printf(" 4. Min Flip (Signed) \n");
      printf("\n");
      printf(" ----------------------\n");
      printf(" Permutation Ranking \n");
      printf(" ----------------------\n");
      printf(" 5. Max Flip \n");
      printf(" 6. Min Flip\n");
      printf(" 7. Max Flip (Signed) \n");
      printf(" 8. Min Flip (Signed) \n");
      printf("\n");
      printf(" ----------------------\n");
      printf(" Permutation UnRanking \n");
      printf(" ----------------------\n");
      printf(" 9. Max Flip \n");
      printf(" 10. Min Flip\n");
      printf(" 11. Max Flip (Signed) \n");
      printf(" 12. Min Flip (Signed) \n");

      printf("\n ENTER selection #: "); scanf("%d", &type);

      if (type < 0 || type > 12) {
            printf("\n INVALID ENTRY\n\n");
            exit(0);
      }

      // long int constraints: MAX n seems to be 20 or 16 (Signed)
      printf(" ENTER length n: ");
      scanf("%d", &n);

      if (type % 4 == 0 || type %4 == 3) SIGNED = 1;
      if (type % 4 == 1 || type %4 == 3) MAX = 1;
      if (type % 4 == 0 || type %4 == 2) MIN = 1;

      if (type >= 1 && type <= 4) GEN = 1;
      else if (type >=5 && type <= 8) {
            RANK = 1;
            printf(" ENTER permutation (separated by spaces): ");
            for (i=1; i<=n; i++) scanf("%d", &a[i]);
            for (i=1; i<=n; i++) {
                  if (a[i] < 0) {
                        a[i] = -a[i];
                        sign[i] = 1;
                  }
                  else sign[i] = 0;
            }
      }
      else if (type >=9 && type <= 12) {
            UNRANK = 1;
            if (SIGNED == 0) rank = factorial[n];
            else rank = factorial[n] * pow2[n];
            printf(" ENTER rank (between 1 and %lld): ", rank);
            scanf("%lld", &rank);
      }
      printf("\n");
}
//----------------------------------------
void Init() {
      int j;

      //==============
      // INITIAL PERM
      //==============
      for (j=1; j<=n; j++) a[j] = j;
      for (j=1; j<=n; j++) sign[j] = 0;

      //===========
      // INIT NEXT
      //===========
      for (j=0; j<=n+1; j++) c[j] = 0;
      for (j=0; j<=n+1; j++) f[j] = j;
}
//----------------------------------------
void Print() {
int i;

      for (i=1; i<=n; i++) {
            if (sign[i] == 0 || !SIGNED) printf(" %d ", a[i]);
            else printf("-%d ", a[i]);
      }
      if (!RANK) printf("\n");
      total++;
}
```

```
//--------------------------------------------------
// OPERATIONS ON PERMUTATIONS
//--------------------------------------------------
void RotateRight(int t, int j) {
int i, b[MAX_N];

      for (i=1; i<=t-j; i++) b[i+j] = a[i];
      for (i=1; i<=j; i++) b[i] = a[t-j+i];
      for (i=1; i<=t; i++) a[i] = b[i];

      for (i=1; i<=t-j; i++) b[i+j] = sign[i];
      for (i=1; i<=j; i++) b[i] = sign[t-j+i];
      for (i=1; i<=t; i++) sign[i] = b[i];
}
//--------------------------------------------------
void RotateLeft(int t, int j) {
int i, b[MAX_N];

      for (i=j+1; i<=t; i++) b[i-j] = a[i];
      for (i=1; i<=j; i++) b[t-j+i] = a[i];
      for (i=1; i<=t; i++) a[i] = b[i];

      for (i=j+1; i<=t; i++) b[i-j] = sign[i];
      for (i=1; i<=j; i++) b[t-j+i] = sign[i];
      for (i=1; i<=t; i++) sign[i] = b[i];
}
//--------------------------------------------------
void Flip(int t) {
int i, b[MAX_N];

      for (i=1; i<=t; i++) b[i] = a[t-i+1];
      for (i=1; i<=t; i++) a[i] = b[i];

      //============================
      // Flip Signs for Signed case
      //============================
      for (i=1; i<=t; i++) b[i] = sign[t-i+1];
      for (i=1; i<=t; i++) sign[i] = (b[i]+1) % 2;
}
//--------------------------------------------------
void FlipSign(int t) {
int i;

      for (i=1; i<=t; i++) sign[i] = (sign[i]+1)%2;
}
//----------------------------------------
// UNRANKING
//----------------------------------------
void UnRankMin(long long int rank, int t){
int j,x;

      if (t == 1) { a[1] = 1; return; }

      x = (rank-1) / factorial[t-1];
      rank = rank - x * factorial[t-1];

      UnRankMin(rank, t-1);

      a[t] = t - x;
      for (j=1; j<t; j++) a[j] = 1 + (a[j] + a[t] - 1) % t;
}
//----------------------------------------
void UnRankMinSigned(long long int rank, int t){
int j,x;

      if (t == 1) {
            a[1] = 1;
            if (rank == 2) sign[1] = 1;
            else sign[1] = 0;
            return;
      }
      x = (rank-1) / (factorial[t-1] * pow2[t-1]);
      rank = rank - x * factorial[t-1] * pow2[t-1];

      if (x < t) {
            a[t] = t-x;
            sign[t] = 0;
      }
      else {
            a[t] = 2*t -x;
            sign[t] = 1;
      }

      UnRankMinSigned(rank, t-1);

      for (j=1; j<t; j++) {
            x = 1 + (a[j] + a[t] - 1) % t;
            if ((sign[t] == sign[j] && x > a[t]) || (sign[t] != sign[j] && x < a[t])) sign[j] = 1;
            else sign[j] = 0;
            a[j] = x;
      }
}
//----------------------------------------
void UnRankMax(long long int rank, int t) {
int d;

      //===========
      // BASE CASE
      //===========
      if (t == 2) {
            if(rank == 1) { a[1] = 1; a[2] = 2;}
            if(rank == 2) { a[1] = 2; a[2] = 1;}
            return;
      }

      a[t] = t;
      if (rank % 2 == 0) {
            UnRankMax( 2*((rank-1)/(2*t)) + 2, t-1);
            d = 2*t - (rank % (2*t));
      }
      else {
            UnRankMax( 2*((rank-1)/(2*t)) + 1, t-1);
            d = (rank-1) % (2*t);
      }
      RotateLeft(t,d/2);
}
//----------------------------------------
void UnRankMaxSigned(long long int rank, int t) {
int d;

      a[t] = t; sign[t] = 0;
      //===========
      // BASE CASE
      //===========
      if (t == 1) {
            if (rank == 2) sign[1] = 1;
            return;
      }

      d = (rank-1) % (4*t);
      if (d >= 2*t) {
            d = 4*t - d - 1;
            UnRankMaxSigned( 2*((rank-1)/(4*t)) + 2, t-1);
      }
      else UnRankMaxSigned( 2*((rank-1)/(4*t)) + 1, t-1);

      if (d % 2 == 0) {
            FlipSign(d/2);
            RotateLeft(t,d/2);
      }
      else {
            Flip(t);
            RotateRight(t,d/2);
            FlipSign(d/2);
      }
}
//----------------------------------------
// RANKING
//----------------------------------------
long long int RankMin(int t){
int j;

      if (t == 1) return 1;
      for (j=1; j<t; j++) a[j] = (a[j] - a[t] + t) % t;
      return ((t-a[t]) * factorial[t-1] + RankMin(t-1));
}
//----------------------------------------
long long int RankMinSigned(int t){
int j;

      if (t == 1 && sign[1] == 0) return 1;
      if (t == 1 && sign[1] == 1) return 2;

      for (j=1; j<t; j++) {
            if ((sign[t] == sign[j] && a[t] < a[j]) || (sign[t] != sign[j] && a[t] > a[j])) sign[j] = 1;
            else sign[j] = 0;
            a[j] = (a[j] - a[t] + t) % t;
      }
      if (sign[t] == 0) return ((t-a[t]) * factorial[t-1] * pow2[t-1] + RankMinSigned(t-1));
      else return ((2*t - a[t]) * factorial[t-1] * pow2[t-1] + RankMinSigned(t-1));
}
//----------------------------------------
long long int RankMax(int t) {
long long int i,d,x,p;

      //============
      // BASE CASES
      //============
      if (t == 2 && a[1] ==1) return 1;
      if (t == 2 && a[1] ==2) return 2;

      //=====================================
      // Determine position of largest value
      //=====================================
      for (i=1; i<=t; i++) if (a[i] == t) p = i;

      //===================================================================
      // Rotate to either the first or last permutation in the bracelet class
      //===================================================================
      d = 2*(t-p);
      RotateRight(t,t-p);

      //=================================================================================
      // The parity indicates whether or not the rotate perm is first or last in the class
      //=================================================================================
      x = RankMax(t-1);
      if (x % 2 == 0) return (t*x - d);
      else return (t*(x-1) + d+1);
}
//----------------------------------------
long long int RankMaxSigned(int t) {
long long int i,d,x,p;

      //============
      // BASE CASE
      //============
      if (t == 1 && sign[1] == 0) return 1;
      if (t == 1 && sign[1] == 1) return 2;

      //=====================================
      // Determine position of largest value
      //=====================================
      for (i=1; i<=t; i++) if (a[i] == t) p = i;

      //=======================================================================
      // Rotate/Flip to either the first or last permutation in the bracelet class
      //=======================================================================
      d = 2*(t-p);
      RotateRight(t,t-p);
      FlipSign(t-p);

      if (sign[t] == 1) {
            FlipSign(t);
            d += 2*t;
      }

      //==============================================================================
      // The parity indicates whether or not the rotate perm is first or last in the class
      //==============================================================================
      x = RankMaxSigned(t-1);
      if (x % 2 == 0) return (2*t*x - d);
      else return (2*t*(x-1) + d+1);
}
//-----------------------------------------
// flip-sequence GENERATION
//-----------------------------------------
int NextMin() {
int j;

      j = f[2];
      f[2] = 2;
      c[j]++;
      if (c[j] == j-1 || j == 2) {
            c[j] = 0;
            f[j] = f[j+1];
            f[j+1] = j+1;
      }
      if (j == n+1) return(0);
      return(j);
}
//-----------------------------------------
int NextMinSigned() {
int j;

      j = f[1];
      f[1] = 1;
      c[j]++;
      if (c[j] == 2*j-1 || j == 1) {
            c[j] = 0;
            f[j] = f[j+1];
            f[j+1] = j+1;
      }
      if (j == n+1) return(0);
      return(j);
}
//-----------------------------------------
int NextMax() {
int j;

      j = f[n];
      f[n] = n;
      c[j]++;
      if (c[j] == j || j == n) {
            c[j] = 0;
            f[j] = f[j-1];
            f[j-1] = j-1;
      }
      if (j == 1) return(0);
      return(j);
}
//-----------------------------------------
int NextMaxSigned() {
int j;
```

```
        j = f[n];
        f[n] = n;
        c[j]++;
        if (c[j] == 2*j+1 || j == n) {
              c[j] = 0;
              f[j] = f[j-1];
              f[j-1] = j-1;
        }
        if (j == 0) return(0);
        return(j);
}
//----------------------------------------
void Gen() {
int j;

        do {
              Print();
              if (MIN && !SIGNED) j= NextMin();
              if (MIN && SIGNED) j= NextMinSigned();
              if (MAX && !SIGNED) j= NextMax();
              if (MAX && SIGNED) j= NextMaxSigned();
              Flip(j);

        } while (j>0);

        printf("Total = %lld\n\n", total);
}
//----------------------------------------
int main() {
int j;

        factorial[0] = 1;
        for (j=1; j<=MAX_N; j++) factorial[j] = factorial[j-1] * j;;

        pow2[0] = 1;
        for (j=1; j<=MAX_N; j++) pow2[j] = pow2[j-1] * 2;

        Input();

        if (GEN) {
              Init();
              Gen();
        }
        if (RANK) {
              printf("The rank of permutation ");
              Print();
              if (MIN && !SIGNED) rank = RankMin(n);
              if (MIN && SIGNED) rank = RankMinSigned(n);
              if (MAX && !SIGNED) rank = RankMax(n);
              if (MAX && SIGNED) rank = RankMaxSigned(n);
              printf(" is: %lld\n\n", rank);
        }
        if (UNRANK) {
              if (MIN && !SIGNED) UnRankMin(rank,n);
              if (MIN && SIGNED) UnRankMinSigned(rank,n);
              if (MAX && !SIGNED) UnRankMax(rank,n);
              if (MAX && SIGNED) UnRankMaxSigned(rank,n);
              printf("The permutation of rank %lld is: ", rank);
              Print();
        }
}
```