Necklaces and Lyndon words in colexicographic order

Daniel Gabrić and Joe Sawada
School of Computer Science
University of Guelph
50 Stone Road East
Guelph, ON N1G 2W1
Canada
dgabric@uoguelph.ca
jsawada@uoguelph.ca

Abstract

We present the first constant-amortized-time algorithms for generating all length-n necklaces and Lyndon words over a k-letter alphabet in colexicographic order, for arbitrary $k \geq 2$. Our approach introduces a novel class of words called *quasinecklaces*, which serve as an easily generated superset of necklaces through which all necklaces can be efficiently identified. We derive a formula for the number $\mathrm{Q}_k(n)$ of length-n quasinecklaces and show that $\mathrm{Q}_k(n)$ is proportional to the number of length-n necklaces, which is the key property needed to achieve constant amortized time. We also apply our results to efficiently generate a well-known de Bruijn sequence and efficiently generate necklaces and Lyndon words subject to a weight constraint.

Keywords— Necklace, Lyndon word, colex order, Grandmama de Bruijn sequence, quasinecklace

1 Introduction

Let Σ_k denote the k-letter totally ordered alphabet $\{1, 2, \ldots, k\}$ where $1 < 2 < \cdots < k$. Throughout this paper, we implicitly assume all words have symbols from Σ_k. We call a word w a *necklace*[1] if it is, not necessarily uniquely, lexicographically smallest among all of its nontrivial rotations. The word w is a *Lyndon word* if it is strictly smaller than all of its nontrivial rotations. For example, the word 113213 is both a necklace and a Lyndon word, 121212 is a necklace but not a Lyndon word, and the word 12312 is neither a Lyndon word nor a necklace. Necklaces and Lyndon words are classical objects in combinatorics on words

[1]Necklaces are also commonly referred to as equivalence classes of words under rotation.

and have many practical applications. For example, they appear in string algorithms [1, 8], coding theory [12, 13], bionformatics [3, 15, 17], and even music theory [6].

The study of necklaces and Lyndon words naturally raises the question of how to generate them efficiently. We call a generation algorithm constant amortized time (CAT) if the total work performed divided by the number of objects generated is bounded by a constant. Several CAT algorithms for generating necklaces and Lyndon words in lexicographic order are known. Fredricksen, Kessler, and Maiorana [10, 11] presented the FKM algorithm, which generates necklaces in lexicographic order and was subsequently shown to be CAT by Ruskey et al. [18]. Duval [9] independently discovered a different lexicographic algorithm for necklaces, later proved to run in CAT by Berstel and Pocchiola [2]. Notably, the lexicographically least rotation of a de Bruijn sequence can be obtained by concatenating the Lyndon words generated by Duval's algorithm.

Although necklaces and Lyndon words can already be efficiently generated in lexicographic order, some applications rely on different orderings. For example, the Grandmama de Bruijn sequence [7] is constructed by concatenating the primitive roots of all necklaces arranged in colexicographic (colex) order, and colex orderings of necklaces also arise in the construction of universal cycles for subsets [4]. Sawada et al. [19, 20] introduced a CAT algorithm to generate binary necklaces in colex order. Their method relies on an auxiliary set of words called pseudonecklaces, which are easy to generate in colex order and whose cardinality is proportional to that of binary necklaces. For alphabets of size $k \geq 3$, however, no such CAT generation algorithm has been available.

Main results. In this paper, we resolve this gap by presenting the first CAT algorithms for generating k-ary necklaces and Lyndon words in colex order for arbitrary $k \geq 2$. To do so, we introduce a novel class of words called quasinecklaces. Quasinecklaces are similar in spirit to pseudonecklaces: every pseudonecklace is a quasinecklace, but the converse does not hold in general. We show that the number $\mathrm{Q}_k(n)$ of length-n quasinecklaces over Σ_k is $\mathcal{O}(\mathrm{N}_k(n))$, where $\mathrm{N}_k(n)$ is the number of length-n necklaces over Σ_k, which is the key property needed to achieve constant amortized time. This bound also holds if we add an upper bound on the weight of the strings. We apply these results to efficiently generate the Grandmama de Bruijn sequence and bounded-weight universal cycles. We provide a complete C implementation in Appendix C.

Outline. The remainder of the paper is organized as follows. Section 2 establishes preliminary definitions and notation. Section 3 introduces quasinecklaces, presents a CAT algorithm for generating them in colex order, and derives a formula for their count $\mathrm{Q}_k(n)$. Section 4 describes how to modify the quasinecklace algorithm to generate necklaces or Lyndon words in colex order, analyzes its running time, and extends the method to handle a weight constraint. In Section 5, we apply our results to efficiently a (bounded-weight) de Bruijn sequence. We conclude with a summary and open questions in Section 6. In this paper, we assume the unit-cost RAM model of computation.

2 Preliminaries

Recall that Σ_k denotes the k-letter totally ordered alphabet $\{1, 2, \ldots, k\}$ where $1 < 2 < \cdots < k$. Throughout this paper, we implicitly assume all words have symbols from Σ_k and all words are 1-indexed. We will also use the following naming conventions: the letters a, b, c, d, e are reserved for symbols, the letters f, g, h are reserved for functions, the letter k is reserved for the size of the alphabet, the letters i, j, ℓ, m, n, p, q, r are reserved for integers, and the letters s, t, u, v, w, x, y, z are reserved for words. Note that symbols will also function as integers in the range $1, 2, \ldots, k$.

Let w be a word of length n. Let i and j be integers such that $1 \leq i, j \leq n$. Let $\min(u)$ denote the smallest symbol of u. We denote the i'th symbol of w as $w[i]$. We say that y is a *factor* of w if there exist (possibly empty) words x and z such that $w = xyz$. We will use the notation $w[i : j]$ to denote the factor of w starting at index i and ending at index j where $w[i : j] = \epsilon$ if $i > j$. Any factor of the form $w[1 : i]$ (resp. $w[i : n]$) is called a *prefix* (resp. *suffix*). If $i < n$ (resp. $i > 1$), then we call the prefix (resp. suffix) *proper*. For $p > 0$, let $w^p = ww^{p-1}$ where $w^0 = \epsilon$. The word w is said to be a *power* if $w = u^p$ for some word u and some integer $p > 1$; otherwise, we say that w is *primitive*. The *primitive root* of w is the shortest prefix u of w such that $w = u^p$ for some $p \geq 1$.

Let u and v be words of lengths m and n respectively. Then u comes before v in *lexicographic* order if either u is a prefix of v or if there exists an integer i such that $u[1 : i-1] = v[1 : i-1]$ and $u[i] < v[i]$. Additionally, u comes before v in *colexicographic* (colex) order if either u is a suffix of v or if there exists an integer i such that $u[m-i+1 : m] = v[n-i+1 : n]$ and $u[m-i] < v[n-i]$. If u is lexicographically less than (or equal to) v, we write $u < v$ (resp. $u \leq v$).

Let u and v be words of length n. We say that u is a *rotation* of v if there exist words x, y such that $u = xy$ and $v = yx$. We formalize this notion by introducing the left-shift map σ so that $\sigma^i(u) = yx$ where $u = xy$ and $|x| = i$, where i is an integer such that $0 \leq i \leq |u|$. We say that u is a *nontrivial* rotation of v if $\sigma^i(u) = v$ where i is an integer such that $0 < i < |u|$.

We can now formally define necklaces and Lyndon words. The word w is said to be a *necklace* if $w \leq \sigma^i(w)$ for all integers i such that $0 < i < |w|$. The word w is said to be a *Lyndon word* if $w < \sigma^i(w)$ for all integers i such that $0 < i < |w|$. Equivalently, the word w is a Lyndon word if and only if it is lexicographically smaller than all of its proper suffixes. Note that every Lyndon word is primitive and the primitive root of a necklace is a Lyndon word. Let $\mathbf{N}_k(n)$ (resp. $\mathbf{L}_k(n)$) denote the set of length-n necklaces (resp. Lyndon words) over Σ_k and let $\mathrm{N}_k(n) = |\mathbf{N}_k(n)|$ (resp. $\mathrm{L}_k(n) = |\mathbf{L}_k(n)|$). See sequences A000031, A001037, A001867, A027376, A001868, and A027377 in the *On-Line Encyclopedia of Integer Sequences* [16] for sample values of $\mathrm{N}_2(n)$, $\mathrm{L}_2(n)$, $\mathrm{N}_3(n)$, $\mathrm{L}_3(n)$, $\mathrm{N}_4(n)$, and $\mathrm{L}_4(n)$, respectively. The following two remarks are proved in [5].

Remark 1. *For $n, k \geq 2$, $\mathrm{L}_k(n) \in \Theta(\mathrm{N}_k(n))$.*

Remark 2. *For $n \geq 1$ and $k \geq 2$, $\sum_{t=1}^{n} \mathrm{N}_k(t) \in \Theta(\mathrm{N}_k(n))$.*

Definition 3. Let w be a word of length n, $a = \min(w)$, and ℓ be the length of a largest run of a's in w. We say that w is a *quasinecklace* if the following conditions are satisfied:

(a) The word a^ℓ is a prefix of w.

(b) If $w[i+1:i+\ell] = a^\ell$ for any i where $1 \leq i+\ell < n$, then $w[\ell+1] \leq w[i+\ell+1]$.

(c) If $w[n] = a$, then $w = a^n$.

Let $\mathbf{Q}_k(n)$ denote the set of length-n quasinecklaces and let $\mathrm{Q}_k(n) = |\,\mathbf{Q}_k(n)|$. See Table 1 in Appendix A for a list of Lyndon words, necklaces, and quasinecklaces for $n = 5$, $k = 3$ as the appear in colex order.

A binary word is a *pseudonecklace* if its maximal prefix of the form 1^*2^* is lexicographically no larger than any other maximal factor of the form 1^*2^*. Let $\mathbf{P}(n)$ denote the set of all length-n pseudonecklaces. See Table 2 in Appendix B for a list of quasinecklaces and pseudonecklaces for $n = 8$, $k = 2$.

3 Generating Quasinecklaces

In this section, we outline the approach we take to efficiently generating quasinecklaces in colex order.

The key step in the approach of Sawada et al. [20] to efficiently generate all *binary* necklaces was to identify the set $\mathbf{P}(n)$ of pseudonecklaces. This step is important because pseudonecklaces can easily be generated in colex order and the number of pseudonecklaces is proportional to the number of binary necklaces (i.e., $|\,\mathbf{P}(n)| \in \mathcal{O}(\mathrm{N}_2(n))$). Given these two facts, generating binary necklaces efficiently requires only a simple and "cheap" test to rule out any pseudonecklace that is not a binary necklace. We follow the same approach but use a different, and slightly larger, set of length-n words called quasinecklaces.

Quasinecklaces are very similar to pseudonecklaces. In fact, it is not too hard to show that every pseudonecklace is a quasinecklace and that there are quasinecklaces that are not pseudonecklaces. Our first step is to show that the number of quasinecklaces is proportional to the number of necklaces.

Theorem 4. *Let $k \geq 2$ and $n \geq 1$ be integers. Then $\mathrm{Q}_k(n) \in \mathcal{O}(\mathrm{N}_k(n))$.*

Proof. It is sufficient to show that there is a one-to-one map from $\mathbf{Q}_k(n) - \mathbf{N}_k(n)$ to $\mathbf{N}_k(n)$. Let $w \in \mathbf{Q}_k(n) - \mathbf{N}_k(n)$. Since w is not a necklace, we have that $w \neq a^n$ for all $a \in \Sigma_k$. Therefore, we can write $w = a^\ell b u$ where $a < b$ and $\ell \geq 1$. We define the mapping $w \mapsto w'$ where $w' = a^\ell(b-1)u$. We prove that the mapping is one-to-one, since it is clear that w' is a necklace.

Suppose $w_1, w_2 \in \mathbf{Q}_k(n) - \mathbf{N}_k(n)$ and $w_1 \neq w_2$. We can write $w_1 = a_1^{\ell_1} b_1 u_1$ and $w_2 = a_2^{\ell_2} b_2 u_2$ where $a_1 < b_1$, $a_2 < b_2$, and $\ell_1, \ell_2 \geq 1$. For the sake of a contradiction, suppose $a_1^{\ell_1}(b_1-1)u_1 = a_2^{\ell_2}(b_2-1)u_2$. If $\ell_1 = \ell_2$, then $w_1 = w_2$, a contradiction. Assume $\ell_1 > \ell_2$. Since w_1 is not a necklace, there is a proper suffix v of w_1 that is smaller than w_1. This suffix v must

begin with $a_1^{\ell_1}b_1$ and must be a suffix of u_1. But since $\ell_1 > \ell_2$ and $a_1^{\ell_1}(b_1-1)u_1 = a_2^{\ell_2}(b_2-1)u_2$, we have that u_1 is a suffix of u_2. This implies that $a_1^{\ell_1} = a_2^{\ell_1}$ is a factor of u_2. But this contradicts w_2 being a quasinecklace. Therefore, the mapping is one-to-one. □

Before we present our algorithm to generate quasinecklaces, we introduce two functions, the second of which is central to our algorithm. Let $\text{runs}(u) = (a, \ell, b, \ell_1)$ such that

- $a = \min(u)$,
- ℓ is the maximum number of contiguous a's in u,
- b is the smallest symbol that follows a^ℓ in u where $b = a$ if a^ℓ is a suffix of u, and
- ℓ_1 is the maximum number of contiguous a's in the prefix of u.

Let $\text{MS}(j, u, \text{runs}(u))$ be the largest symbol $c \in \Sigma_k$ such that cu is a suffix of some quasinecklace of length $n = j + |u|$. When expanding $\text{runs}(u)$ in $\text{MS}(j, u, \text{runs}(u))$, we abuse notation and write $\text{MS}(j, u, a, \ell, b, \ell_1)$. For example, if $k = 3$, then $\text{MS}(112, 3, \text{runs}(112)) = \text{MS}(112, 3, 1, 2, 2, 2) = 2$ since 3112 is not the suffix of a length-6 quasinecklace, but 2112 is. In our recursive algorithm, we maintain a suffix of some quasinecklace in $\mathbf{Q}_k(n)$ and use MS to compute the possible symbols we could use to extend this suffix. In Lemma 6, we fully characterize $\text{MS}(j, u, \text{runs}(u))$.

Remark 5. *If u is a suffix of some quasinecklace in $\boldsymbol{Q}_k(n)$, then $1^{n-|u|-1}cu$ is a quasinecklace in $\boldsymbol{Q}_k(n)$ for all symbols c where $1 \le c \le \text{MS}(j, u, \text{runs}(u))$.*

Lemma 6. *Let n and j be integers such that $1 \le j \le n$. Let u be the length-$(n-j)$ suffix of some quasinecklace $w \in \boldsymbol{Q}_k(n)$. Let $(a, \ell, b, \ell_1) = \text{runs}(u)$. Let $b_1 = u[\ell_1 + 1] = w[j + \ell_1 + 1]$ if $\ell_1 + 1 \le n - j$; otherwise let $b_1 = u[1] = w[j + 1]$. Then*

$$\text{MS}(j, u, \text{runs}(u)) = \begin{cases} 1, & \text{if } u \text{ ends with } 1; \quad (1) \\ k, & \text{if } j = n \text{ or } j > \ell + 1 \text{ or both } j > 1 \text{ and } a > 1; \quad (2) \\ b, & \text{if } j = \ell + 1 \text{ or } \ell = n - 1; \quad (3) \\ a, & \text{if } j = 1 \text{ and } w[n] > a \text{ and } b_1 \le b \text{ and } \ell_1 + 1 = \ell; \quad (4) \\ a, & \text{if } j = 1 \text{ and } w[n] > a \text{ and } \ell_1 + 1 > \ell; \quad (5) \\ \max(a - 1, 1), & \text{otherwise.} \quad (6) \end{cases}$$

Proof. Let $c \in \Sigma_k$ be the largest symbol such that cu is a suffix of a quasinecklace in $\mathbf{Q}_k(n)$. We prove that $\text{MS}_n(j, u, \text{runs}(u)) = c$ for each case.

(1) Suppose u ends with 1. There is exactly one quasinecklace in $\mathbf{Q}_k(n)$ that ends with 1 and it is 1^n. Thus $c = 1$.

For the remaining cases we have that u does not end with 1.

(2) If $j = n$, then $c = k$ since the last symbol of a quasinecklace is constrained. If $j > \ell + 1$ or both $j > 1$ and $a > 1$, then $1^{j-1}ku$ is a quasinecklace, so $c = k$.

For the remaining cases we have $j < n$, $j \leq \ell + 1$, and either $j = 1$ or $a = 1$.

(3)
- Suppose $j = \ell + 1$. First we prove that $w = 1^\ell bu$ is a quasinecklace. We must have $b > 1$, since $b = 1$ implies that u ends with 1. We immediately get that w satisfies all conditions of Definition 3. Therefore w is a quasinecklace

  Now we show that b is the largest symbol such that bu is a suffix of some quasinecklace in $\mathbf{Q}_k(n)$. Suppose $d > b$ and consider the word du. Since $j = \ell + 1 > 1$ and either $j = 1$ or $a = 1$, we have that $a = 1$. Any quasinecklace in $\mathbf{Q}_k(n)$ that has du as a suffix must begin with 1^ℓ. Now $w = 1^\ell du \in \mathbf{Q}_k(n)$ by Remark 5. But $1^\ell d$ is a prefix of w, which contradicts condition (b), since $d > b$. Therefore $c = b$.
- Suppose $\ell = n - 1$. Then $u = b^{n-1}$. Clearly $bu = b^n$ is a quasinecklace. It is also clear that $c = b$ is the largest symbol such that $w = cu$ is a quasinecklace.

For the remaining cases we have $j < \ell + 1$ and $\ell < n - 1$. For cases (4) and (5) we have $c \leq a$, since $j = 1$ and any quasinecklace in $\mathbf{Q}_k(n)$ that has u as a suffix must not begin with a symbol larger than a. Thus, to show that $c = a$ for the following two cases, it is sufficient show that $w = au$ is a quasinecklace.

(4) Suppose $j = 1$, $w[n] > a$, $b_1 \leq b$, and $\ell_1 + 1 = \ell$. Then w begins with $a^\ell b_1$. But this means that w satisfies condition (a) of Definition 3. Condition (b) is satisfied because $b_1 \leq b$. Condition (c) is satisfied since $w[n] \neq a$. Therefore w is a quasinecklace.

For the remaining cases we have $j > 1$,$w[n] \leq a$, $b_1 > b$, or $\ell_1 + 1 \neq \ell$.

(5) Suppose $j = 1$, $w[n] > a$, and $\ell_1 + 1 > \ell$. Since $\ell_1 + 1 > \ell$, w begins with at least $\ell + 1$ copies of a, the unique longest run of a's in w. Therefore, w satisfies conditions (a) and (b) of Definition 3. Condition (c) is also satisfied because $w[n] > a$. Thus, w is a quasinecklace.

For the remaining cases we have $j > 1$, $w[n] \leq a$, or $\ell_1 + 1 \leq \ell$.

(6) Suppose all the conditions in cases (1)-(5) do not hold. We show that $c = \max(a-1, 1)$ by showing that if $a = 1$, then $c = 1$, and if $a > 1$, then $c = a - 1$.

- Suppose $a = 1$. From the end of the proof of case (3), we have $j < \ell + 1$, which implies $c \leq a$. By Remark 5 we get that $1^{n-j}u$ is a quasinecklace. Thus, $c = 1$.
- Suppose $a > 1$. Then we know that $j = 1$, since the end of the proof of case (2) shows that $j = 1$ or $a = 1$. We prove that the word $w = du$ where $d = a - 1$ is a quasinecklace. Conditions (a) and (b) of Definition 3 are satisfied since w begins with the unique copy of the smallest symbol in w. Condition (c) is satisfied since $\min(u) = a$ and so $w[n] \geq a > a - 1$. Therefore, w is a quasinecklace and $c \geq a - 1$.

  Now we show that $c \leq a - 1$ by showing that $w = du$ is not a quasinecklace for $d \geq a$. If $w[n] = a$, then condition (c) is not satisfied since $\ell < n - 1$ and so

$w \neq a^n$. So $w[n] > a$. From the end of the proof of case (5) we have that either $\ell_1 + 1 = \ell$ or $\ell_1 + 1 < \ell$. If $\ell_1 + 1 = \ell$, then we must have $b_1 > b$ by the end of case (4). But now we have that $a^\ell b_1$ is a prefix of w, which violates condition (b). So suppose that $\ell_1 + 1 < \ell$. Now we have that w begins with fewer than ℓ copies a's, which violates condition (a). Therefore, we have $c \leq a - 1$.

□

The value $\mathrm{MS}(j, u, \mathrm{runs}(u))$ can clearly be computed in constant time. The function GEN in Algorithm 1 generates the quasinecklaces $\mathbf{Q}_k(n)$ in colex order. The function VISIT processes, or outputs, the current word w. The initial call is $\textsc{Gen}(n, 0, 0, 0, 0)$.

Algorithm 1 The function GEN generates all quasinecklaces in $\mathbf{Q}_k(n)$ in colex order.

```
1: function Gen(j, a, ℓ, b, ℓ_1)
2:     MaxSymbol ← MS(j, w[j+1:n], a, ℓ, b, ℓ_1)
3:     b_1 ← if ℓ_1 + 1 ≤ n − j then w[j + ℓ_1 + 1] else a
4:     if MaxSymbol = 1 or j = 0 then Visit( )
5:     else
6:         for c ← 1 to MaxSymbol do
7:             w[j] ← c
8:             if j = n then Gen(n − 1, c, 1, c, 1)
9:             else if c < a then Gen(j − 1, c, 1, w[j + 1], 1)
10:            else if c = a and (ℓ_1 + 1 > ℓ or ℓ_1 + 1 = ℓ and b_1 ≤ b) then Gen(j − 1, a, ℓ_1 + 1, b_1, ℓ_1 + 1)
11:            else if c = a then Gen(j − 1, a, ℓ, b, ℓ_1 + 1)
12:            else Gen(j − 1, a, ℓ, b, 0)
13:        w[j] ← 1
```

We briefly justify that this algorithm generates each quasinecklace in constant-amortized time. Throughout this proof sketch we refer to the recursion tree of the algorithm where each node represents a suffix of some quasinecklace in $\mathbf{Q}_k(n)$, and each leaf is a quasinecklace in $\mathbf{Q}_k(n)$. Simply put, a generation algorithm runs in constant-amortized time if the total amount of work done divided by the number of objects generated is bounded by a constant. Therefore, it is sufficient to show that the total number of nodes in the recursion tree is proportional to the total number of leaf nodes and that the amount of work done by each node is bounded by a constant. Let $MaxSymbol = \mathrm{MS}(j, w[j+1:n], \mathrm{runs}(w[j+1:n]))$. When $MaxSymbol = 1$, the algorithm immediately stops and processes w, as there is only one quasinecklace with suffix $w[j+1:n]$, namely $1^j w[j+1:n]$. We avoid any additional work involved with filling in 1^j by initializing w to 1^n and maintaining the prefix to be 1^j after all recursive calls on Line 13. When $MaxSymbol > 1$, the algorithm performs $MaxSymbol$ recursive calls to add $MaxSymbol$ different symbols to the beginning of $w[j+1:n]$. Therefore, each internal node of the recursion tree has at least two children, which implies that the total number of nodes is proportional to the total number of leaves. By inspection, it is clear that every node in the recursion tree is responsible for a constant amount of work. Since the depth of the recursion is at most n, the algorithm uses $\mathcal{O}(n)$ space. Thus, we get the following theorem.

Theorem 7. *Algorithm 1 generates the quasinecklaces $\boldsymbol{Q}_k(n)$ in colex order in constant amortized time using $\mathcal{O}(n)$ space.*

It is important to note that Theorem 7 assumes the algorithm is not printing out the quasinecklaces, as printing a word w is inherently $\mathcal{O}(|w|)$.

3.1 Enumeration

In this section, we provide a simple recurrence for the number $\mathrm{Q}_k(n)$ of quasinecklaces. First we define a quantity that we use to count $\mathrm{Q}_k(n)$. Let $A_{a,\ell,b}(n)$ be the number of quasinecklaces in $\mathbf{Q}_k(n)$ that begin with $a^\ell b$ where a, b are symbols such that $a < b$.

Lemma 8. *Let n, ℓ be positive integers a, b be symbols such that $a < b$. Then*

$$A_{a,\ell,b}(n) = \begin{cases} 0, & \text{if } n \leq \ell; \\ 1, & \text{if } n = \ell + 1; \\ (k-a)\sum_{i=1}^{\ell} A_{a,\ell,b}(n-i) + (k-b+1)A_{a,\ell,b}(n-\ell-1), & \text{otherwise.} \end{cases}$$

Proof. Let w be a quasinecklace in $\mathbf{Q}_k(n)$ beginning with $a^\ell b$. If $n \leq \ell$, then any length-n word cannot begin with $a^\ell b$, thus $A_{a,\ell,b}(n)$. If $n = \ell+1$, then there is exactly one quasinecklace in $\mathbf{Q}_k(n)$ that begins with $a^\ell b$, namely $w = a^\ell b$, thus $A_{a,\ell,b}(n) = 1$.

Suppose $n > \ell + 1$. Write $w = a^\ell u$ for some word u that begins with b. We look at the suffixes of u. Since w cannot have the factors $a^{\ell+1}$ or $a^\ell b'$ for some $b' < b$, we have that either $u = vca^\ell d$ where v is a word and c, d are symbols such that $c > a$ and $d > b$ or $u = v'ca^{i-1}d'$ where v' is a word, i is some integer such that $1 \leq i \leq \ell$, and $d > a$ is a symbol. But now both $a^\ell vc$ and $a^\ell v'c$ are both quasinecklaces since they are prefixes of the quasinecklace w and $c > a$. There are $A_{a,\ell,b}(n-\ell-1)$ quasinecklaces of the form $a^\ell vc$ and $k-b+1$ choices for the symbol d. Therefore, there are $A_{a,\ell,b}(n-\ell-1)$ quasinecklaces in $\mathbf{Q}_k(n)$ of the form $w = a^\ell vca^\ell d$. Additionally, there are $A_{a,\ell,b}(n-i)$ quasinecklaces of the form $a^\ell v'c$ and $k-a$ choices for the symbol d', Thus, there are $(k-a)A_{a,\ell,b}(n-i)$ quasinecklaces in $\mathbf{Q}_k(n)$ of the form $w = a^\ell v'ca^\ell d'$. Summing $(k-a)A_{a,\ell,b}(n-i)$ over all possible i in addition to summing $A_{a,\ell,b}(n-\ell-1)$, we get the desired formula for $n > \ell + 1$. □

Theorem 9. *Let n and k be integers such that $n \geq 1$ and $k \geq 2$. Then*

$$Q_k(n) = k + \sum_{a=1}^{k-1} \sum_{b=a+1}^{k} \sum_{\ell=1}^{n-1} A_{a,\ell,b}(n).$$

Proof. A quasinecklace in $\mathbf{Q}_k(n)$ is either of the form a^n or begins with the word $a^\ell b$ where a and b are symbols such that $a < b$. There are k quasinecklaces of the form a^n. There are $A_{a,\ell,b}(n)$ quasinecklaces that begin with $a^\ell b$. Summing $A_{a,\ell,b}(n)$ over all possible a, ℓ, b, in addition to summing k, we get the desired formula. □

4 Generating Necklaces and Lyndon words

In this section, we describe how to modify Algorithm 1 to generate necklaces and Lyndon words in colex order. Applying an $\mathcal{O}(n)$ test on each quasinecklace to determine if it is a necklace leads to a $\mathcal{O}(n)$-amortized time algorithm. To obtain constant-amortized time, we maintain two additional pieces of information as each quasinecklace is generated:

- $\text{suf}(w)$: the length of the lexicographically smallest suffix of w, and
- $\text{psuf}(w, r)$: the length of the longest suffix of w whose primitive root is of length r.

By applying the values $\text{suf}(w)$ and $\text{psuf}(w, r)$ we can test if a word w is a Lyndon word or necklace in constant time applying the following lemma.

Lemma 10. *Let w be a length-n word where $r = \text{suf}(w)$ and $p = \text{psuf}(w, r)$. Then*

(1) w is a Lyndon word if and only if $r = n$, and

(2) w is a necklace if and only if $p = n$.

Proof. We prove both statements:

(1) Follows from the definition of a Lyndon word.

(2) Suppose the word w is a necklace. Then $w = u^j$ for some $j \geq 1$ where u is a Lyndon word. The word $v = w[n - r + 1 : n]$ is a Lyndon word, since it is the smallest suffix of w. We must have $u = v$, for otherwise we would have that either u or v are not Lyndon words. Therefore, w is the longest suffix of w whose primitive root is of length r, and so $p = n$. The steps of this proof are reversible, so the backwards direction has also been covered.

□

Lemma 10 implies that it is enough to know the values of $\text{suf}(w)$ and $\text{psuf}(w, \text{suf}(w))$ to determine whether w is a Lyndon word or necklace. Computing these values for every quasinecklace w generated is computationally expensive. A better approach is to incrementally update these values for each new suffix generated. Lemmas 11 and 13 show how to compute $\text{suf}(w)$ and $\text{psuf}(w, \text{suf}(w))$ incrementally. Corollaries 12 and 14 show how to compute $\text{suf}(w)$ and $\text{psuf}(w, \text{suf}(w))$ if $w = 1^j u$ for some $j \geq 1$ and we only know $\text{suf}(u)$ and $\text{psuf}(u, \text{suf}(u))$.

Lemma 11. *Let n and j be integers such that $n \geq j \geq 1$. Let $u = w[j + 1 : n]$ be the length-$(n - j)$ suffix of a quasinecklace $w \in \boldsymbol{Q}_k(n)$, $r = \text{suf}(u)$, and c be an integer such that $1 \leq c \leq \text{MS}(j, u, \text{runs}(u))$. Then*

$$\text{suf}(cu) = \begin{cases} |cu|, & \text{if } cw[j + 1 : j + r - 1] < w[n - r + 1 : n]; \\ r, & \text{otherwise.} \end{cases}$$

Proof. If $cw[j+1:j+r-1] < w[n-r+1:n]$, then cu is the smallest suffix of cu and therefore is primitive, so $\mathrm{suf}(cu) = |cu|$. Otherwise, we have that $cw[j+1:j+r-1] \geq w[n-r+1:n]$, which implies $\mathrm{suf}(cu) = \mathrm{suf}(u) = r$. □

The next result follows from j applications of the previous lemma.

Corollary 12. *Let n and j be integers such that $n \geq j \geq 1$. Let $u = w[j+1:n]$ be the length-$(n-j)$ suffix of a quasinecklace $w \in \boldsymbol{Q}_k(n)$, $r = \mathrm{suf}(u)$, and $w' = 1^j w[j+1:r]$. Then*

$$\mathrm{suf}(1^j u) = \begin{cases} n, & \text{if } w'[1:r] < w[n-r+1:n]; \\ r, & \text{otherwise.} \end{cases}$$

Lemma 13. *Let n and j be integers such that $n \geq j \geq 1$. Let $u = w[j+1:n]$ be the length-$(n-j)$ suffix of a quasinecklace $w \in \boldsymbol{Q}_k(n)$. Let $r = \mathrm{suf}(u)$ and $p = \mathrm{psuf}(u, \mathrm{suf}(u))$. Let c be an integer such that $1 \leq c \leq \mathrm{MS}(j, u, \mathrm{runs}(u))$. Then*

$$\mathrm{psuf}(cu, r) = \begin{cases} |cu|, & \text{if } cw[j+1:j+r-1] < w[n-r+1:n]; \\ |cu|, & \text{if } cw[j+1:j+r-1] = w[n-r+1:n] \text{ and } |cu| - p = r; \\ p, & \text{otherwise.} \end{cases}$$

Proof. If $cw[j+1:j+r-1] < w[n-r+1:n]$, then cu is the smallest suffix of cu and is therefore primitive, so $\mathrm{psuf}(cu, r) = |cu|$. If $cw[j+1:j+r-1] = w[n-r+1:n]$ and $|cu| - p = r$, then $cw[j+1:j+r-1] = v$ and $w[j+r:n] = v^i$ for some $i \geq 1$ where $v = w[n-r+1:n]$. Since v is a Lyndon word by definition and $cu = v^{i+1}$, we have that cu is the longest suffix of cu whose primitive root is of length r. Therefore $\mathrm{psuf}(cu, r) = |cu|$. Otherwise, we have that $cw[j+1:j+r-1] > w[n-r+1:n]$ or $|cu| - p \neq r$. In the case that $cw[j+1:j+r-1] > w[n-r+1:n]$, we have that the word $v = w[n-r+1:n]$ is not a prefix of cu. Therefore, the longest suffixes of cu and u that have a primitive root of length r are the same, and so $\mathrm{psuf}(cu, r) = p$. If $cw[j+1:j+r-1] = w[n-r+1:n]$ and $|cu| - p \neq r$, then the primitive root of cu is not of length r. Thus, the longest suffixes of cu and u that have a primitive root of length r are the same, and so $\mathrm{psuf}(cu, r) = p$. □

The next result follows from j applications of the previous lemma. It is important to note that Corollary 14 does not hold when $u = 1$, but Lemma 13 does. Corollary 14 assumes that u is of the form $v'v^i$ where $r = \mathrm{suf}(u) = |v|$ and $p = \mathrm{psuf}(u, r) = |v^i|$. The corollary's second case tests whether prepending 1^j completes exactly one additional copy of v (i.e., is $1^j u = v^{i+1}$?). When $u = 1$, prepending 1^j adds j new copies of v instead of just one, a scenario the second case cannot detect. Therefore, the corollary falls through to the last case and gives the wrong value.

Corollary 14. *Let n and j be integers such that $n \geq j \geq 1$. Let $u = w[j+1:n] \neq 1$ be the length-$(n-j)$ suffix of a quasinecklace $w \in \boldsymbol{Q}_k(n)$, $r = \mathrm{suf}(u)$, $p = \mathrm{psuf}(u, \mathrm{suf}(u))$, and $w' = 1^j w[j+1:r]$. Then*

$$\mathrm{psuf}(1^j u, r) = \begin{cases} n, & \text{if } w'[1:r] < w[n-r+1:n]; \\ n, & \text{if } w'[1:r] = w[n-r+1:n] \text{ and } n - p = r; \\ p, & \text{otherwise.} \end{cases}$$

The computation of $\mathrm{suf}(cu)$ and $\mathrm{psuf}(cu, \mathrm{suf}(cu))$ requires the knowledge of whether $cw[j+1:j+r-1] < w[n-r+1:n]$ and $cw[j+1:j+r-1] = w[n-r+1:n]$. In Algorithm 2, we describe a function CMP which has four parameters, p, ℓ, q, ℓ_1, and returns the result of a comparison between $w[p:n-q+p]$ and $w[q:n]$. This function has the following preconditions:

- $p < q$,
- $w[q:n]$ is the smallest suffix of $w[p+1:n]$,
- $w[p:n]$ begins with a^{ℓ_1} where a is a symbol, and
- if $q < n$, then $w[q:n]$ begins with a^{ℓ}.

The function CMP is largely straightforward: it first compares the lengths of the runs of a at the beginning of both words, and then performs a symbol-by-symbol comparison between the two words. However, Lines 2 and 6 require closer examination. In the case that $q = n$, we have that $w[q:n] = a$ but the integer ℓ may not be 1; Line 2 handles this special case separately. If $p+i = q$, then $w[p:n] = uuv$ where $w[q:n] = uv$. Since $w[q:n]$ is the smallest suffix of $w[p+1:n]$, we have $uv < v$, which implies $uuv < uv$ and $w[p:n-q+p] < w[q:n]$. Thus, on Line 6 we immediately return that $w[p:n-q+p] < w[q:n]$ when $p+i = q$.

Algorithm 2 Returns -1 if $w[p:n-q+p] < w[q:n]$, 0 if $w[p:n-q+p] = w[q:n]$, and 1 if $w[p:n-q+p] > w[q:n]$.

```
1: function CMP(p, ℓ1, q, ℓ)
2:     if q = n then return 0
3:     if ℓ1 < ℓ then return 1
4:     if ℓ1 > ℓ then return −1
5:     for i ← ℓ to n − q do
6:         if p + i = q then return −1
7:         if w[p + i] > w[q + i] then return 1
8:         else if w[p + i] < w[q + i] then return −1
9:     return 0
```

4.1 Modifying Algorithm 1

Assume that w, j, c, a, ℓ, b, and ℓ_1 are as in Algorithm 1. The following modifications can be applied to this algorithm to generate necklaces and Lyndon words:

1. Add two new parameters r and p to Gen such that $r = \mathrm{suf}(w[j+1:n])$ and $p = \mathrm{psuf}(w[j+1:n], r)$.
2. Update r and then p before each recursive call.

3. Before calling the function VISIT, update ℓ_1, r, and p, since we may have $j > 0$ which would mean ℓ_1, r, and p are not up-to-date.

   If $j > 0$, we first update ℓ_1 in the following way. If $w[j+1:n] = 1$, we set $p \coloneqq n$. If $\min(w[j+1:n]) = 1$, set $\ell_1 \coloneqq \ell_1 + j$; otherwise set $\ell_1 \coloneqq j$. Then we update r using Corollary 12 and p using Corollary 14 if $w[j+1:n] \neq 1$.

4. In the function VISIT, process w as a Lyndon word if $r = n$, and as a necklace if $p = n$.

5. The initial call is $\textsc{Gen}(n, 0, 0, 0, 0, 0, 0)$ where the additional two zeroes at the end represent values for parameters r and p.

6. Change the call to GEN on Line 8 to $\textsc{Gen}(n-1, c, 1, c, 1, 1, 1)$.

See Appendix C for a C program that implements these algorithms to generate quasinecklaces, necklaces, and Lyndon words.

4.2 Analysis

Recall that each node in the recursion tree for GEN represents a suffix of some quasinecklace in $\mathbf{Q}_k(n)$, or equivalently a recursive call to GEN. We have already proved that the total number of nodes in the recursion tree of GEN is proportional to $\mathrm{Q}_k(n)$, which is itself proportional to $\mathrm{N}_k(n)$ by Theorem 4. From Remark 1, $\mathrm{L}_k(n)$ is $\Theta(\mathrm{N}_k(n))$. Additionally, apart from the calls to CMP, the work done by each recursive call is constant. Therefore, to prove that our algorithm for generating necklaces or Lyndon words in colex order runs constant-amortized time, it is sufficient to show that the total amount of work done by all calls to CMP is proportional to $\mathrm{N}_k(n)$, or equivalently, $\mathrm{Q}_k(n)$.

We analyze the work done by all calls to $\mathrm{CMP}(p, \ell_1, q, \ell)$. Let w be a length-n word. Recall that CMP returns the result of the comparison between $w[p : n-q+p]$ and $w[q : n]$ given that the preconditions outlined before Algorithm 1 are satisfied. In particular, $w[q : n]$ is the lexicographically smallest suffix of $w[2 : n]$ and $p < q$. Since the total number of nodes in the recursion tree for GEN is proportional to $\mathrm{Q}_k(n)$, any constant amount of work done by a call to CMP is, when summed over all nodes, proportional to $\mathrm{Q}_k(n)$. Therefore, in our analysis of CMP we omit Lines 2 through 4, the first iteration of the **for** loop, and the last iteration of the **for** loop. The remaining iterations of the **for** loop compare $w[p+\ell+1 : p+i]$ and $w[q+\ell+1 : q+i]$ symbol-by-symbol for some integer i such that $\ell < i < \min(n-q, q-p)$. We create a mapping that maps at most two comparisons to a single element of $\mathbf{Q}_k(n-p+1)$. For simplicity, and without loss of generality, we assume $p = 1$.

First, we define the domain of our mapping. We describe the words w that are processed by the **for** loop when the index variable i is such that $\ell < i < \min(n-q, q-1)$. Since i is not the last iteration of the **for** loop, the conditions in the first $i-\ell+1$ iterations all evaluated to false. Additionally, all the conditions on Lines 2 through 4 evaluated to false. Therefore, we have $w[1 : i+1] = w[q : q+i]$ and furthermore $w[1 : \ell+1] = a^\ell b$ and $w[q : q+\ell] = a^\ell b$, where $a = \min(w)$ and $a < b$. Since $w[q : n]$ is the lexicographically smallest suffix of $w[2 : n]$, w is

a quasinecklace. We explicitly define the domain in the following way. Let $\mathbf{D}(n)$ denote the set of all pairs (w, i) of quasinecklaces $w \in \mathbf{Q}_k(n)$ and integers i such that:

1. $w[1 : \ell + 1] = w[q : q + \ell] = a^\ell b$ where $\ell \geq 1$,
2. $w[1 : i + 1] = w[q : q + i]$, where $\ell < i < \min(n - q, q - 1)$, and
3. $w[q : n]$ is the lexicographically smallest suffix of $w[2 : n]$.

Now we are ready to present our mapping. We define $f : \mathbf{D}(n) \to \mathbf{Q}_k(n)$ by

$$f(w, i) = \begin{cases} a^i w[i + 1 : n], & \text{if } w[i + 1] > a = \min(w); \\ a^i(a + 1)w[i + 2 : n], & \text{otherwise.} \end{cases}$$

The mapping f is constructed to map quasinecklaces to quasinecklaces. Since we know that the number of quasinecklaces is proportional to the number of necklaces, it is sufficient to show that the size of the preimage of every quasinecklace is bounded above by a constant. We do this by treating both cases as separate functions with their own restricted domains. If the size of the preimage of every quasinecklace is bounded above by a constant for both cases, then the same is true for the mapping f.

Lemma 15. *Let* $\mathbf{D}_1 = \{(w, i) \in \mathbf{D}(n) \mid w[i + 1] > \min(w)\}$. *Then* $f(w, i)$, *restricted to* $(w, i) \in \mathbf{D}_1$, *is one-to-one.*

Proof. Suppose $f(u, i) = f(v, j)$ where $(u, i), (v, j) \in \mathbf{D}_1$. Since $f(u, i) = f(v, j)$, we have $\min(u) = \min(v) = a$. But this means that $i = j$ since $f(u, i)$ begins with $a^j v[j + 1]$ where $v[j + 1] > a$. Now we have $a^i u[i + 1 : n] = a^i v[i + 1 : n]$, which implies that the smallest suffix of $u[2 : n]$ is equal to the smallest suffix of $v[2 : n]$. Then $u[1 : i] = v[1 : i]$, and thus $u = v$. Therefore, f is one-to-one. □

Note the second case in the definition of f implies that $w[i + 1] = \min(w)$.

Lemma 16. *Let* $\mathbf{D}_2 = \{(w, i) \in \mathbf{D}(n) \mid w[i + 1] = \min(w)\}$. *Then* $f(w, i)$, *restricted to* $(w, i) \in \mathbf{D}_2$, *is one-to-one.*

Proof. Suppose $f(u, i) = f(v, j)$ where $(u, i), (v, j) \in \mathbf{D}_2$. Since $f(u, i) = f(v, j)$, we have $\min(u) = \min(v) = a$. But this means that $i = j$ since $f(u, i)$ begins with $a^j(a + 1)$. Now we have $a^i(a + 1)u[i + 2 : n] = a^i(a + 1)v[i + 2 : n]$, which implies that the smallest suffix of $u[2 : n]$ is equal to the smallest suffix of $v[2 : n]$. Then $u[1 : i + 1] = v[1 : i + 1]$, and thus $u = v$. Therefore, f is one-to-one. □

Lemmas 15 and 16 show that, when restricted to their respective domains, the first and second cases of f are each one-to-one. Consequently, each of these cases could, in principle, map different elements of their domains to the same quasinecklace without violating injectivity within the case itself. Taking all cases together, we have that f can map at most two distinct elements of $\mathbf{D}(t)$ to a single quasinecklace in $\mathbf{Q}_k(t)$. Since the elements of $\mathbf{D}(t)$ correspond to all but a constant number of comparison operations performed by CMP, each quasinecklace is therefore charged for at most two comparisons. Therefore, since

- $\mathrm{Q}_k(t) \in \mathcal{O}(\mathrm{N}_k(t))$ by Theorem 4,
- $\sum_{t=1}^{n} \mathrm{N}_k(t) \in \mathcal{O}(\mathrm{N}_k(n))$ by Remark 2, and
- $\mathrm{L}_k(n) \in \Theta(\mathrm{N}_k(n)$,

the total number of comparison operations performed by CMP across all recursive calls, considering all suffix lengths $1 \leq t \leq n$, is $\mathcal{O}(\mathrm{L}_k(n))$.

Theorem 17. *The necklaces $\boldsymbol{N}_k(n)$ and Lyndon words $\boldsymbol{L}_k(n)$ can each be listed in colex order in constant amortized time using $\mathcal{O}(n)$ space.*

4.3 Weight constraint

The *weight* of a word is the sum of its symbols. To generate the necklaces $\mathbf{N}_k(n)$ or Lyndon words $\mathbf{L}_k(n)$ with weight at most w in colex order, we need only make the following additional modifications to those outlined in Section 4.1 that modified Algorithm 1:

1. Add a parameter that maintains the weight of the current suffix $w[j+1:n]$ that is updated appropriately at each recursive call.
2. The function MS is modified so that it returns the maximum value that will not violate the weight constraint, accounting for the minimum prefix 1^{j-1}.

Corollary 18. *The necklaces $\boldsymbol{N}_k(n)$ and the Lyndon words $\boldsymbol{L}_k(n)$ with weight at most w can each be listed in colex order in constant amortized time using $\mathcal{O}(n)$ space.*

Proof. The weight of the current suffix can easily be maintained at each recursive call with constant work. Similarly, the second modification also requires an extra constant amount of work to the function MS. In the proof of Theorem 4, observe that the one-to-one mapping of non-necklaces to necklaces preserves the weight constraint. Furthermore, in the analysis from Section 4.2, they key function f maps a suffix and an index to a quasinecklace with the same or smaller weight. Thus, the same analysis applies to this bounded weight variant. □

5 Application: de Bruijn sequences

A *de Bruijn sequence* is a cyclic sequence of length k^n that contains every length n-word over Σ_k as a subword exactly once. The *Grandmama* de Bruijn sequence [7], can be constructed by concatenating together the primitive roots of the necklaces in $\mathbf{N}_k(n)$ as they appear in colex order. For example, the colex listing of $\mathbf{N}_3(3)$ is:

$$111, 112, 122, 222, 132, 113, 123, 223, 133, 233, 333.$$

By concatenating together the primitive roots of each necklace, we obtain the Grandmama de Bruijn sequence:

$$1 \cdot 112 \cdot 122 \cdot 2 \cdot 132 \cdot 113 \cdot 123 \cdot 223 \cdot 133 \cdot 233 \cdot 3$$

of length $3^3 = 27$, where $\cdot$ denotes concatenation. By applying Theorem 17, we obtain the following result.

Corollary 19. *The Grandmama de Bruijn sequence for $n, k \geq 1$ can be constructed in constant amortized time per symbol using $\mathcal{O}(n)$ space.*

This is an improvement by a factor of n over the bound established in [7]. For $k = 2$, the result was previously established in [20].

Recently, it has been demonstrated that concatenating the primitive roots of all necklaces in $\mathbf{N}_k(n)$ *with weight at most w* as they appear in colex order yields a bounded weight de Bruijn sequence [4]. Such sequences have found application in the construction of universal cycles for k-subsets and k-multisets [4]. The result in Corollary 19 also holds for this bounded-weight variant of the Grandmama de Bruijn sequence.

6 Final remarks

In this paper we give a CAT algorithm to list the necklaces $\mathbf{N}_k(n)$ and Lyndon words $\mathbf{L}_k(n)$ (with an upper bound on their weight) as they appear in colex order. When these words are listed in lexicographic order, efficient ranking and unranking algorithms are known [14]. Thus, it is an interesting open question to efficiently rank/unrank necklaces and Lyndon words as they appear in colex order. Such algorithms would likely lead to an efficient algorithm to decode the Grandmama de Bruijn sequence and its bounded-weight variant.

7 Acknowledgment

Joe Sawada (grant RGPIN-2025-03961) and Daniel Gabrić (grant RGPIN-2026-04863) gratefully acknowledge research support from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta. The "runs" theorem. *SIAM J. Comput.* **46**(5) (2017), 1501–1514.

[2] J. Berstel and M. Pocchiola. Average cost of Duval's algorithm for generating Lyndon words. *Theoret. Comput. Sci.* **132**(1) (1994), 415–425.

[3] S. Bonomo, S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. Sorting conjugates and suffixes of words in a multiset. *Internat. J. Found. Comp. Sci.* **25**(08) (2014), 1161–1175.

[4] C. Campbell, L. Janik-Jones, and J. Sawada. Universal cycles for k-subsets and k-multisets of an n-set. *Discrete Mathematics* (2026), to appear.

[5] K. Cattell, F. Ruskey, J. Sawada, M. Serra, and C.R. Miers. Fast algorithms to generate necklaces, unlabeled necklaces, and irreducible polynomials over GF(2). *J. Algorithms* **37**(2) (2000), 267–282.

[6] M. Chemillier. Periodic musical sequences and Lyndon words. *Soft Computing* **8**(9) (2004), 611–616.

[7] P. B. Dragon, O. I. Hernandez, J. Sawada, A. Williams, and D. Wong. Constructing de Bruijn sequences with co-lexicographic order: The k-ary Grandmama sequence. *European J. Combinatorics* **72** (2018), 1–11.

[8] J.-P. Duval. Factorizing words over an ordered alphabet. *J. Algorithms* **4**(4) (1983), 363–381.

[9] J.-P. Duval. Génération d'une section des classes de conjugaison et arbre des mots de Lyndon de longueur bornée. *Theoret. Comput. Sci.* **60**(3) (1988), 255–283.

[10] H. Fredricksen and I. J. Kessler. An algorithm for generating necklaces of beads in two colors. *Discrete Math.* **61**(2) (1986), 181–188.

[11] H. Fredricksen and J. Maiorana. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discrete Math.* **23**(3) (1978), 207–210.

[12] S. W. Golomb. *Shift Register Sequences*. Holden-Day, Inc. San Francisco, CA, 1967.

[13] S. W. Golomb, B. Gordon, and L. R. Welch. Comma-free codes. *Canad. J. Math.* **10** (1958), 202–209.

[14] T. Kociumaka, J. Radoszewski, and W. Rytter. Efficient ranking of Lyndon words and decoding lexicographically minimal de Bruijn sequence. *SIAM J. Disc. Math.* **30**(4) (2016), 2027–2046.

[15] I. Martayan, B. Cazaux, A. Limasset, and C. Marchet. Conway–Bromage–Lyndon (CBL): an exact, dynamic representation of k-mer sets. *Bioinformatics* **40** (2024), i48–i57.

[16] N. J. A. Sloane et al. OEIS Foundation Inc. (2022), The On-Line Encyclopedia of Integer Sequences, https://oeis.org.

[17] J. Olbrich. Fast and Memory-Efficient BWT Construction of Repetitive Texts Using Lyndon Grammars. In Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman, editors, *33rd Annual European Symposium on Algorithms (ESA 2025)*, Vol. 351 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 60:1–60:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[18] F. Ruskey, C. Savage, and T. Min Yih Wang. Generating necklaces. *J. Algorithms* **13**(3) (1992), 414–430.

[19] J. Sawada and A. Williams. A Gray code for fixed-density necklaces and Lyndon words in constant amortized time. *Theoret. Comput. Sci.* **502** (2013), 46–54.

[20] J. Sawada, A. Williams, and D. Wong. Necklaces and Lyndon words in colexicographic and binary reflected Gray code order. *J. Discrete Algorithms* **46-47** (2017), 25–35.

A Table of Lyndon words, necklaces, and quasinecklaces

Table 1: Lyndon words, necklaces, and quasinecklaces in colex order for $n = 5$, $k = 3$. Colour coding: blue (non-Lyndon), red (non-necklace). Read the lists top down, then left to right.

Lyndon words $\mathbf{L}_3(5)$		**Necklaces $\mathbf{N}_3(5)$**		**Quasinecklaces $\mathbf{Q}_3(5)$**	
11112	12313	11111	12313	11111	11313
11212	11123	11112	11123	11112	12313
11312	12123	11212	12123	11212	13313
11122	11223	11312	11223	12212	11123
12122	12223	11122	12223	11312	12123
11222	22223	12122	22223	12312	11223
12222	13223	11222	13223	11122	12223
13222	11323	12222	11323	12122	22223
11322	12323	22222	12323	11222	13223
12322	22323	13222	22323	12222	11323
13322	13323	11322	13323	22222	12323
11132	11133	12322	11133	13222	22323
12132	12133	13322	12133	11322	13323
13132	13133	11132	13133	12322	23323
11232	11233	12132	11233	13322	11133
12232	12233	13132	12233	11132	12133
13232	22233	11232	22233	12132	13133
11332	13233	12232	13233	13132	11233
12332	23233	13232	23233	11232	12233
13332	11333	11332	11333	12232	22233
11113	12333	12332	12333	13232	13233
11213	22333	13332	22333	11332	23233
12213	13333	11113	13333	12332	11333
11313	23333	11213	23333	13332	12333
		12213	33333	11113	22333
		11313		11213	13333
				12213	23333
				13213	33333

B Table of pseudonecklaces and quasinecklaces

Table 2: Pseudonecklaces and quasinecklaces in colex order for $n = 8$, $k = 2$. Quasinecklaces in red are not pseudonecklaces. Read the lists top down, then left to right.

Pseudonecklaces P(8)		**Quasinecklaces $\mathbf{Q}_2$**(8)	
11111111	11112122	11111111	11221122
11111112	11212122	11111112	11112122
11121112	11122122	11121112	11212122
11112112	12122122	11112112	12212122
11212112	11222122	11212112	11122122
11122112	11111222	11122112	12122122
11111212	11211222	11222112	11222122
11211212	11121222	11111212	12222122
11121212	12121222	11211212	11111222
12121212	11221222	11121212	11211222
11221212	12221222	12121212	11121222
11112212	11112222	11221212	12121222
11212212	11212222	12221212	11221222
11122212	12212222	11112212	12221222
12122212	11122222	11212212	11112222
11222212	12122222	12212212	11212222
11111122	11222222	11122212	12212222
11121122	12222222	12122212	11122222
11221122	22222222	11222212	12122222
		12222212	11222222
		11111122	12222222
		11121122	22222222

C Generation Algorithm

```
//---------------------------------------------------------------------------
// NECKLACES, LYNDON WORDS, OR QUASINECKLACES IN COLEX ORDER IN O(1)-AMORTIZED TIME
//---------------------------------------------------------------------------
#include <stdio.h>
#define MAX 100

int type,N,k,maxW,w[MAX],NECK=0,LYNDON=0,DB=0;
long long int total=0;

//---------------------------------------
// RETURNS -1 IF w[p:n-q+p] < w[q:n],
//          0 IF w[p:n-q+p] = w[q:n], and
//          1 IF w[p:n-q+p] > w[q:n].
//---------------------------------------
int CMP(int p, int l1, int q, int l) {
    int i;
    if (q == N) return 0;
    if (l1 < l) return 1;
    if (l1 > l) return -1;
    for (i=l; i<=N-q; i++) {
        if (p+i == q) return -1;
        if (w[p+i] > w[q+i]) return 1;
        else if (w[p+i] < w[q+i]) return -1;
    }
    return 0;
}
//-----------------------------------------------------
// COMPUTE suf(w[j:n])
//-----------------------------------------------------
int Suf(int j, int r, int cmp) {
    if (cmp == -1) return N-j+1;
    return r;
}
//-------------------------------------------------
// COMPUTE PSuf(w[j:n], r)
//-------------------------------------------------
int PSuf(int j, int r, int p, int cmp) {
    if (cmp == -1) return N-j+1;
    if (cmp == 0 && N-j+1-p == r) return N-j+1;
    return p;
}
//---------------------------------------------------------------------------
// RETURNS LARGEST c SUCH THAT cw[j+1:n] IS THE SUFFIX OF SOME LENGTH-N QUASINECKLACE
//---------------------------------------------------------------------------
int MaxSymbol(int j, int a, int l, int b, int l1) {
    int b1 = (l1+1 <= N-j) ? w[j+l1+1] : a;
    if (j == 0 || w[N] == 1 && j < N) return 1;
    if (j == N || j > l+1 || (j > 1 && a > 1)) return k;
    if (j == l + 1 || l == N-1) return b;
    if (j == 1 && b1 <= b && l1+1 == l && w[N] != a) return a;
    if (j == 1 && l1+1 > l && w[N] != a) return a;
    return (a-1 >= 1) ? a-1 : 1;
}
//-------------------------------------------------
// PRINT EACH QUASINECK/NECK/LYN
//-------------------------------------------------
void Visit(int r, int p) {
    int i;
    //TEST IF A WORD IS A NECK/LYN
    if (NECK && p < N) return;
    if (LYNDON && r < N) return;
    //OUTPUT DB SEQUENCE OR QUASINECK/NECK/LYN
    if (DB) for (i=1; i<=r; i++) printf("%d", w[i]);
    else {
        total++;
        for (i=1; i<=N; i++) printf("%d", w[i]);
        printf("\n");
    }
}
```

```
//-------------------------------------------------------------------------
// LIST NECKLACES (LYNDON WORDS/QUASINECKLACES) IN COLEX ORDER.
//-------------------------------------------------------------------------
void Gen(int j, int a, int l, int b, int l1, int r, int p, int wt) {
    int c,MS,cmp,nr,np,b1;
    MS = MaxSymbol(j,a,l,b,l1);
    b1 = (l1+1 <= N-j) ? w[j+l1+1] : a;
    if (wt-(MS+(j-1)) < 0) MS = wt-(j-1);
    if (MS == 1) {
        if (j == 0) Visit(r,p);
        else if (w[N] == 1) Visit(1,N);
        else if (a > 1) Visit(N,N);
        else {
            //UPDATE r AND p
            cmp = CMP(1,l1+j,N-r+1,l);
            nr = Suf(1,r,cmp);
            np = PSuf(1,r,p,cmp);
            Visit(nr,np);
        }
    }else {
        for(c=1; c<=MS; ++c){
            w[j] = c;
            if (j == N) Gen(N-1,c,1,c,1,1,1,wt-c);
            else if (c < a) Gen(j-1,c,1,w[j+1],1,N-j+1,N-j+1,wt-c);
            else if (c == a && (l1+1 > l || l1+1 == l && b1  <= b)) {
                cmp = CMP(j,l1+1,N-r+1,l);
                nr = Suf(j,r,cmp);
                np = PSuf(j,r,p,cmp);
                Gen(j-1,a,l1+1,b1,l1+1,nr,np,wt-c);
            } else if(c == a) Gen(j-1,a,l,b,l1+1,r,p,wt-c);
            else Gen(j-1,a,l,b,0,r,p,wt-c);
        }
        w[j] = 1;
    }
}
//-------------------------------------------------------------
void Input() {
    int i;
    printf("      Colex order                          Bounded weight    \n");
    printf("  -------------------------------      -----------------    \n");
    printf("  1.  Necklaces                        5. Necklaces         \n");
    printf("  2.  Lyndon words                     6. Lyndon words      \n");
    printf("  3.  Quasinecklaces                   7. Quasinecklaces    \n");
    printf("  4.  Grandmama De Bruijn sequence\n");
    printf("\n  ENTER object #: ");    scanf("%d", &type);

    if (type == 1 || type == 4 || type == 5) NECK = 1;
    if (type == 2 || type == 6) LYNDON = 1;
    if (type == 4) DB = 1;

    printf("\n  ENTER length n: ");    scanf("%d", &N);
    printf("  ENTER k: ");             scanf("%d", &k);

    if(type >= 5) {
        printf("  ENTER Weight (%d - %d): ",N,k*N); scanf("%d", &maxW);
    } else maxW = k*N;

    printf("\n");

    for(i=1; i<=N; ++i) w[i] = 1;
}
//-------------------------------------------------
int main() {
    Input();
    Gen(N,0,0,0,0,0,0,maxW);
    if(!DB) printf("Total = %lld\n",total);
    else printf("\n\n");
    return 0;
}
```