Algorithms for the Lyndon unique maximal factorization.

David E. Daykin.
Deptartment of Mathematics, University of Reading, U.K.

Address for all correspondence:
david.daykin@googlemail.com
Sunnydene, Tuppenny Lane, Emsworth, Hants, England, PO10 8HG.

Abstract. Let Σ be a totally ordered set. We work on finite strings $b = b_1b_2 \ldots b_m$ of b_i from Σ. Such a b is a *lyn* (Lyndon word) if $m \geq 1$, and b is the unique first in *lex* (lexicographic order) among the m rows of the $m \times m$ circulant matrix with b as first row.

A classic result is that every string b has a *unique max factorization umf(b)* into *lyns*, each *lyn* of maximum possible size in b.

In 1983 J. P. Duval [6] published Algorithm 1, which finds *umf(b)*. It was studied in 1991 by A. Apostolico and M. Crochemore [1]. Then their work was studied in 1994 by J.W. Daykin, C.S. Iliopoulos and W.F. Smyth [5].

Since Duval used a programming language, we start by giving a new simple account of his Algorithm 1. Then our Algorithm 2 given here modifies Duval's Algorithm 1 to find *umf(a)*, when a is a string $a = A_1A_2 \ldots A_p$ of lyns A_i.

Our Algorithm 3 is also for a string $a = A_1A_2 \ldots A_p$ of lyns A_i. It is completely different to Algorithms 1,2. It snakes right, left, right, and so on. It revealed the fact that *lyns* have a special structure. We give an example where Algorithm 3 needs almost $2m$ tests, we think that is the most needed, but cannot give a rigorous proof.

We find interesting properties of *lyns*, some of which may be new.

Keywords: algorithm, complexity, factorization, Lyndon word, string.

Footnote. The author thanks the referees for reading this paper carefully, and for making a correction and suggestions that improved it. {The author's supervisor G. Kreisel told him in 1958, "I have the utmost respect for my fellow mathematicians." On the other hand his co-author Rudy Ahlswede said in 1983, "We are not as good as all that you know David!"} **End of Footnote.**

1. Introduction to lyns and umfs.

Mostly integers are fed into digital computers, so without loss of generality, we let Σ be the integers with their usual order $\ldots -2<-1<0<+1<+2 \ldots$. We work on strings (words) $a = a_1a_2 \ldots a_n$ of integers a_i, and use *lex* (lexicographic order) between them. We put $dim(a) = |a| = n$. The string a is a *lyn* (Lyndon word) if $n \geq 1$, and a is the unique first in lexicographic order between the rows of the $n \times n$ circulant matrix with a as first row. For example the string *211* is not a *lyn*, but *112* is a *lyn* as the *3* rows are $112 < 121 < 211$ in *lex*. Clearly a *lyn* is not the empty string λ, and $|\lambda| = 0$. Every (single) integer is a lyn. Also a *lyn* is non-periodic, otherwise the circulant would have two rows the same, and we would not get uniqueness. (Other writers say "primitive" instead of non-periodic. Also, instead of talking about the circulant, they say "minimal in its conjugacy class". In [4] Daykin and Daykin find all factorization families consisting of one row from each non-periodic circulant. So we like circulants.)

If a,b,c are strings and $a = bc$ we have $a \geq_{IS} b$ in *lex*, where IS = *Initial Section.* If $a = def$, $b = dgh$ and $|e| = |g| = 1$, so e,g are integers, then $a >_{FD} b$ when $e > g$, where FD = *First Difference.*

If $a = bc = db$ with $b,c,d \neq \lambda$, then b is said to be a *border* of the string a. If a has no border it is *border-free.* Lemmas 1,2,3,4 here below are found on page 365 of Duval's 1983 paper [6].

Lemma 1. *Every lyn is border-free.*

Lemma 2. *A string a is a lyn iff for all $a = bc$ with $b,c \neq \lambda$ we have $a < c$.*

Lemma 3. *If A,B are lyns then AB is a lyn iff $A < B$.*

Lemma 4. *If A,B are lyns and $A < B$ then AB and all $AA...ABB...B$ are lyns.*

Observe that, when we use a capital letter for a string, it denotes that it was given as a *lyn.* Having chosen a letter for a string, we do not change it.

We do not use it here, but we mention a generalization of Lemma 2.

Lemma 5. (Easy). *Let $a = A_1A_2 \ldots A_p$ where the A_i are lyns. Then a is a lyn iff*

$$a <_{FD} A_iA_{i+1} \ldots A_p \text{ for } 1 < i \leq p.$$

The following more revealing form of Lemma 2 appeared in [3].

Lemma 6. *The string $a = a_1a_2 \ldots a_n$ is a lyn iff*

(1) $$a_1a_2 \ldots a_i <_{FD} a_{n-i+1}a_{n-i+2} \ldots a_n, \text{ for } 1 \leq i < n.$$

(Note that in (1) we have |left side| = |right side| = i, and there is no border.)

From Lemma 6 one easily gets Lemmas 7,8.

Lemma 7. *If string $b \neq \lambda$ is an initial section of a lyn, then b is a lyn iff it is border-free.*

Lemma 8. *Let $A = a_1a_2 \ldots a_n$ be a lyn. If $2 \leq r \leq n$ and e is an integer with $e > a_r$ then the string $a_1a_2 \ldots a_{r-1}e$ is a lyn.* (Note *1213* is a lyn, but *1313,1413,...* are not.)

Lemma 9. (Easy) (Daykin-Daykin [3]) *If xy,yz are lyns with $y \neq \lambda$ then xyz is a lyn.*

Definition 1. *Let $a = a_1a_2 \ldots a_n$ be a string. Let b be a section $b = a_ia_{i+1}a_{i+2} \ldots a_j$ of a. When b is a lyn we say b is max in a if we cannot find a lyn different from b by decreasing i, or increasing j, or by doing both.*

Now suppose $a = A_1A_2 \ldots A_p$ where the A_i are *lyns.* If $A_2 < A_3$ then Lemma 3 makes A_2A_3 a *lyn*, and A_2 is not max in a. If any A_i is not max in a, by xyz Lemma 9, we can join together some of the *lyns* in the factorization of a. This proves

Theorem 1. (Classic). *Every string $a \neq \lambda$ has a unique max factorization*

$umf(a) = [A_1][A_2] \dots [A_p]$ where $a = A_1A_2 \dots A_p$ and each A_i is a lyn max in a. Further we have $A_1 \geq A_2 \geq \dots$
$\geq A_p$.

Lemma 10. *If $a = A_1A_2 \dots A_p$ where the A_i are lyns, and $A_1 \geq_{IS} A_2 \geq_{IS} \dots \geq_{IS} A_p$, then $umf(a) = [A_1][A_2] \dots [A_p]$.*

Proof. Suppose $p \geq 2$. Then A_p is an *IS* of A_1. So A_p is a border of *a* and *a* is not a *lyn*. Next suppose some A_i is not max in *a*. Then by xyz Lemma 9 we would get a section of the *A*'s forming a *lyn*. This is impossible by the first argument. □

This Lemma 10 strengthens Theorem 1 as follows.

Theorem 2. *If $a = A_1A_2 \dots A_p$ where the A_i are lyns then*

$$umf(a) = [A_1][A_2] \dots [A_p] \text{ iff } A_1 \geq A_2 \geq \dots \geq A_p.$$

Lemma 11. (Cut) *If $a = A_1A_2 \dots A_pb$ where the A_i are lyns, and string $b \neq \lambda$, and $A_1 \geq_{IS} A_2 \geq_{IS} \dots \geq_{IS} A_p > b$, then $umf(a) = [A_1][A_2] \dots [A_p]umf(b)$.*

Proof. Suppose the lemma is false, so we do not get the Cut between A_p and *b*. This means there is a *lyn D*, which is a section of *a* and meets both A_p and *b*. Then by the xyz Lemma 9, there is a *lyn* $C = c_1c_2$, where $c_1 = A_i A_{i+1} \dots A_p$ for some *i* in $1 \leq i \leq p$, and $c_2 \neq \lambda$ is an *IS* of *b*.

Case 1. $A_p >_{IS} b$. Here $A_i \geq_{IS} A_{i+1} \geq_{IS} \dots \geq_{IS} A_p >_{IS} c_2$ making c_2 a border of C, which is impossible.

Case 2. $A_p >_{FD} b$. We are given that $A_p = def$ and $b = dgh$ for strings *d,e,f,g,h* with $|e| = |g| = 1$ and $e > g$. Observe that $A_p = def$ is an *IS* of c_1. We cannot have c_2 an *IS* of *d* for that would make c_2 a border of *lyn C*. So *dg* is an *IS* of c_2. Then $e > g$ contradicts (1) for the *lyn* C when the right side of (1) is c_2. □

Lemma 12. *If A,Ab are lyns, and string $b \neq \lambda$ then $A < b$.* (Note A =12 < 51 = b.)

Proof. Trivially $A \neq b$. We assume *A* is not an *IS* of *b*, for otherwise $A < b$ by definition. Then we cannot have *b* an *IS* of *A*, for that would make *b* a border of *lyn Ab*. From circulant *Ab* we have $Ab < bA$. So in this case $A <_{FD} b$.

□

2. Algorithm 1. Duval's Lyndon factorization of a string of integers.

In 1983 [6] Duval gave his algorithm in a programming language, and in contrast to his sophistication we present here a simplified version. We are given a string $b = b_1b_2 \dots b_m$. We assume that we have found that $b = (B)^{\mu}c$, which means that *B* is a *lyn*, and that *b* starts with $\mu \geq 1$ copies of *B* followed by string $c = c_1c_2 \dots c_r$. (Thus *B* is an *IS* of *b*.) Initially *B* is b_1 and $\mu = 1$ and $c = b_2b_3 \dots b_m$.

Case 1. $c = \lambda$. We are finished because $umf(b) = [B][B] \dots [B]$, with μ copies of *B*.

Case 2. $\lambda \neq c = c_1c_2 \dots c_r$. We test *B ? c*, testing $b_i ? c_i$ for $i = 1,2,3,\dots$ in turn.

Case 2.1. (Cut) $B > c$. By Lemma 11 we have $umf(b) = [B][B] \dots [B]\ umf(c)$, with μ copies of *B*. We restart to find $umf(c)$.

Case 2.2. $B \leq_{IS} c$. There is a copy C of B at the start of c. We call this C a newlyn. We increase μ by 1, get a new c, and restart.
Case 2.3. $B <_{FD} c$. This holds because we are not in Case 2.2. Hence there is a least integer $j \geq 1$ with $b_j \neq c_j$, so $b_j < c_j$. If $j \geq 2$ we apply Lemma 8 to find $C = c_1c_2 ... c_j$ is a *lyn* with $B < C$. If $j = 1$ then $C = c_1$ is a *lyn* with $B < C$. In either case we call this C a newlyn. Now Lemma 4 says $(B)^{\mu}C$ is a *lyn*. We restart with *lyn* $(B)^{\mu}C$ in
place of *lyn* B, and $\mu = 1$.

□

It occurred to the author, that if, instead of being given a string of integers, we were given a string of *lyns*, we must get an algorithm, generally requiring fewer tests. In the extreme example, a string AB of two *lyns* A, B with different first integers is factored $[A][B]$ or $[AB]$ with only one integer test.

3. Algorithm 2. We modify Duval's Algorithm 1 for a string of *lyns*.

Our input now is a string $b = A_1A_2 ... A_n$ of *lyns* A_i. In a nutshell, whenever Duval finds a newlyn C, we locate it in our input string. Let ω be the last integer of C.
Case (i). This ω is the end of an A_i, let it be A_{end}. Here we can only follow Duval.
Case(ii). The ω is in an A_i but is not the end of A_i, let it be A_{mid}. Here xyz Lemma 9 on C and A_{mid} gives a *lyn* C^+ longer than C. Now C^+ ends A_{mid}, and it starts at or before the start of C. We will use C^+.
Algorithm 2. Let $b = A_1A_2 ... A_n$ where the A_i are *lyns*. We assume that we know that $b = \{A\}^{\mu}c$, where First A is a *lyn*, Second $A = A_1A_2 ... A_h$ for some h in $1 \leq h < n$, Third μ is an integer $\mu \geq 1$, and Fourth $c = A_kA_{k+1}A_{k+2} ... A_n$ for some k in $1 < k \leq n$. (Note that $\{A\}^{\mu} = A_1A_2 ... A_{k-1}$, but we may not have n = $\mu h + k$.)
Initially $A = A_1$ and $\mu = 1$ and $c = A_2A_3 ... A_n$.
Case 1. $c = \lambda$. We are finished because *umf(b)* = [A][A] ... [A], with μ copies of A.
Case 2. $c \neq \lambda$. We test A ? c. If $A = a_1a_2 ... a_u$ and $c = c_1c_2 ... c_v$, this means testing the integers a_i ? c_i for $i = 1,2,3,...$ in turn.
Case 2.1. (Cut) $A > c$. By Lemma 11 we have *umf(b)* = [A][A] ... [A] *umf(c)*, with μ copies of A. We restart to find *umf(c)*.
Case 2.2. $A \leq_{IS} c$. There is a copy C of A at the start of c. This C is a newlyn.
Case (i) above. We increase μ by 1, and restart on $b = \{A\}^{\mu+1}c^*$, where $Ac^* = c$. (Note that here and below the four start conditions hold.)
Case (ii) above. If C^+ starts at C, then by Lemma 4 we get $L = \{A\}^{\mu}C^+ = AA ... AC^+$ is a *lyn*, and $b = \{L\}d$ for some d. We restart on $\{L\}^1d$. The same holds true, using xyz Lemma 9, if the start of C^+ is before C.
Case 2.3. $A <_{FD} c$. This holds because we are not in Case 2.2. Again put $A = a_1a_2 ... a_u$ and $c = c_1c_2 ... c_v$. Then there is a least integer $j \geq 1$ with $a_j \neq c_j$, so

$a_j < c_j$. If $j \geq 2$ we apply Lemma 8 to find $C = c_1c_2 \ldots c_j$ is a *lyn* with $A < C$. If $j = 1$ then $C = c_1$ is a *lyn* with $A < C$. Now Lemma 4 says $(A)^{\mu}C$ is a newlyn.
Case (i) above. We restart on $b = \{(A)^{\mu}C\}^{\upsilon}e$ with $\upsilon = 1$, where $b = (A)^{\mu}Ce$.
Case (ii) above. We use xyz Lemma 9 on $(A)^{\mu}C$ and A_{mid} to get a *lyn* L. We restart with $b = (L)^{\upsilon}g$ with $\upsilon = 1$, where $b = Lg$.

□

4. Algorithm 3. Our Lyndon factorization of a string of *lyns*.

Suppose A, B, C are *lyns* with $A \geq_{IS} B < C$. Then BC is a *lyn* but $A\ ?\ BC$ can be anything, as shown by Example 1 below.

Example 1. Let A be each of the *lyns* $13, 12, 1, 14, 134$ in turn. Let B be the *lyn* 1, so $A \geq_{IS} B$. Let C be the *lyn* 3 so $B < C$ and $BC = 13$ is a *lyn*. Then $A\ ?\ BC$ is in turn
$=, <_{FD}, <_{IS}, >_{FD}, >_{IS}$. (One can get the same with binary strings.)

□

In view of this Example 1, our Algorithm 3 below has to snake left and right.

Algorithm 3. The input is any string $a = A_1A_2 \ldots A_p$ of *lyns* A_i with $p \geq 2$. We want $umf(a)$. We may have some or all $|A_i| = 1$. We use up the A_i one at a time when we move right. So we assume we know that $A_1 \geq_{IS} A_2 \geq_{IS} \ldots \geq_{IS} A_q$ for some q in $1 \leq q \leq p$.

Case 1. (Stop) $q = p$. We are finished by Theorem 2, (or Lemma 10.)

Case 2. $q < p$. Test $A_q\ ?\ A_{q+1}$. (This means finding the *lex* order between them.)

Case 2.1. (Cut) $A_1 \geq_{IS} A_2 \geq_{IS} \ldots \geq_{IS} A_q >_{FD} A_{q+1}$. Use lemma 11 and restart.

Case 2.2. (Go Right) $A_q \geq_{IS} A_{q+1}$. Increase q and restart.

Case 2.3. (Go Left) $A_1 \geq_{IS} A_2 \geq_{IS} \ldots \geq_{IS} A_q < A_{q+1}$. Here we use Lemma 3. The two *lyns* A_q and A_{q+1} are replaced by the single *lyn* A_qA_{q+1}. If $q = 1$ we just restart. If $1 < q$ then Example 1 shows we do not know $A_{q\ 1}\ ?\ A_qA_{q+1}$. So this is our first test, when we restart at Case 2.

□

Before the algorithm looked at A_{q+1}, it found $umf(A_1A_2 \ldots A_q)$, but this may not be the start of $umf(a)$.

The cost of finding $umf(a)$ is the number of tests $e\ ?\ g$ which the algorithm made. Here e, g are integers, and one test finds $e < g$, $e = g$, or $e > g$. The cost of (Cut) is $|A_q|$ or less. The cost of (Go Right) equals $|A_{q+1}|$. We bound the cost of (Go Left) in Example 2 below.

Suppose Algorithm 3 has run on a string $a_1a_2 \ldots a_n$. Let $a_h\ ?\ a_i$ and $a_j\ ?\ a_k$ be two of the integer tests it performed. It seems that test $a_h\ ?\ a_i$ was performed before test $a_j\ ?\ a_k$ if $i < k$ or $i = k$ and $h < j$.

5. The structure of a Lyndon word.

We say a *lyn* D is *allislyn* if all *IS* of D are *lyns* (this means $a_1 < a_i$ for $1 < i$).

Lemma 13. (Two slopes zip up.) *Let C be a lyn which is not allislyn. Then $C = A_1A_2 \dots A_pB_1B_2 \dots B_q$ with $p \geq 2$, and $q \geq 1$, and all A_i and B_j lyns, and $A_1 \geq_{IS} A_2 \geq_{IS} \dots \geq_{IS} A_p$, and $B_1 \geq B_2 \geq \dots \geq B_q$, and $A_p < B_1$.*

Proof. Let $a \neq \lambda$ be the *IS* of C, with a not a *lyn*, and with $dim(a)$ maximal. So $a \neq C$ and $umf(a) = [A_1][A_2] \dots [A_p]$ with $A_1 \geq A_2 \geq \dots \geq A_p$ and $p \geq 2$. We cannot have $A_i >_{FD} A_{i+1}$ for that would give a cut in C, which is a *lyn*. So $A_1 \geq_{IS} A_2 \geq_{IS} \dots \geq_{IS} A_p$. Let $C = ab$, so $b \neq \lambda$. Then $umf(b) = [B_1][B_2] \dots [B_q]$ with $B_1 \geq B_2 \geq \dots \geq B_q$ and $q \geq 1$ and B_1 is a *lyn*. Next $A_1 A_2 \dots A_pB_1$ is a *lyn*, by definition of a. Hence we must have $A_p < B_1$.

Now A_pB_1 and $E = A_1A_2 \dots A_pB_1$ are *lyns*. So $A_{p-1} < A_pB_1$ making $A_{p-1}A_pB_1$ a *lyn*, and so on. Having zipped E up, we get $E < B_2$ starting the zip up of C. □

A Chinese proverb says, "A single picture is worth a thousand words." So rather than using masses of symbols, we study typical examples.

Into our Algorithm 3 we put a string $a = A_1A_2 \dots A_p$ of *lyns* A_i with $p \geq 2$, and we now discuss what happens.

Case 3. The algorithm starts Goes Left. Here A_1A_2 is a *lyn*. If $p \geq 3$ we replace A_1, A_2 by A_1A_2 and restart.

Case 4. The algorithm always Goes Right. So $A_1 \geq A_2 \geq \dots \geq A_p$, with cost $\leq |A_2A_3 \dots A_p|$, and $umf(a) = [A_1][A_2] \dots [A_p]$ by Theorem 2.

Case 5. The algorithm starts Go Right but has a first Go Left. Each Go Left forms a new *lyn*, as shown in Case 2.3 above. In Example 2 below we go Right five times, then Left six times, and this produces a *lyn*. It should be compared to Lemma 13. This Example 2 is typical of how the algorithm starts to behave, except it does not have equalities like $L_2 = L_3$, (which is $e = \lambda$.)

Example 2. Let $K = L_1L_2L_3L_4L_5L_6L_7$ where K and the L_i are *lyns*. Suppose further

$L_1 = Abcdef >_{IS} L_2 = Abcde >_{IS} L_3 = Abcd >_{IS} L_4 = Abc >_{IS} L_5 = Ab >_{IS} L_6 = A < L_7 = H,$

where b, c, d, e, f are non-empty strings of integers, not necessarily *lyns*.

We feed $L_1, L_2, L_3, L_4, L_5, L_6, L_7$ into the algorithm. First it tests $L_1 ? L_2$. Since $L_1 >_{IS} L_2$ the cost of this test between *lyns* is $|L_2|$. It then does $L_2 ? L_3$ with cost $|L_3|$, and so on till $L_5 ? L_6$. These five Go Right tests cost $|L_2| + \dots + |L_6|$.

The test $L_6 ? L_7$ finds $L_6 < L_7$ so L_6L_7 is a *lyn*. Because K is a *lyn*, by Lemmas 10 and 11, we cannot have $L_5 \geq_{IS} L_6L_7$ or $L_5 >_{FD} L_6L_7$ so $L_5 < L_6L_7$. Thus $L_5L_6L_7$ is a *lyn*. In this way $L_i < L_{i+1}L_{i+2} \dots L_7$ and $L_iL_{i+1} \dots L_7$ is a *lyn* for $i = 6, 5, \dots, 1$. Thus we have six Go Left tests.

Now Ab, A are *lyns*, so Lemma 12 gives $A < b$. In the same way it gives $Ab < c$ and $Abc < d$ and $Abcd < e$ and $Abcde < f$.

Next we consider the costs of the Go Left tests. First we had $A < H$ with cost $\leq |A|$. Second we had $Ab < AH$ which is $b < H$, and hence $A < b < H$ with cost
$\leq |b|$. It tells us $AbAH$ is a *lyn*. Third we had $Abc < AbAH$ so $Ab < c < AH$. Put
$c = (A)c^*$ then $b < c^* < H$ with cost $\leq |c^*|$. Fourth we had $Abcd < AbcAbAH$ so $Abc = AbAc^* < d < AbAH$. Put $d = (Ab)(A)d^*$ then $A < b < c^* < d^* < H$ with cost
$\leq |d^*|$. Continuing $e = (Abc)(Ab)(A)e^*$ and $f = (Abcd)(Abc)(Ab)(A)f^*$ with $A < b < c^* < d^* < e^* < f^* < H$. The total cost of going left is cheap at $\leq |Abc^*d^*e^*f^*| \leq |Abcdef| = |L_1|$.

□

Example 3. Suppose our input *lyns* are $H_1, H_2 \ldots, H_{99}$. Before the algorithm looks at H_{33}, it finds $umf(H_1 \ldots H_{32})$. If this *umf* has an $>_{FD}$ it has a cut, the *lyns* on the left of this cut are done, and will not affect those on its right. So for our purposes, we can assume we have $H_1H_2 \ldots H_{32} = L_1L_2L_3L_4L_5L_6L_7$, as in Example 2, with

$$L_1 \geq_{IS} L_2 \geq_{IS} L_3 \geq_{IS} L_4 \geq_{IS} L_5 \geq_{IS} L_6 = A < L_7 = H = H_{32}.$$

□

Theorem 3. *Consider the typical lyn K in Example 2 above. This K is not allislyn. In the notation there we have* $A < b < c^* < d^* < e^* < f^* < H$. *We put* $w = AbA$, *then*

$$L_1 = wc^*wd^*wc^*we^*wc^*wd^*wc^*wf^* = L_2L_3L_4wf^* \text{ and}$$
$$L_2 = wc^*wd^*wc^*we^* = L_3L_4we^* \text{ and}$$
$$L_3 = wc^*wd^* = L_4wd^* \text{ and}$$
$$L_4 = wc^*$$

with initial conditions $L_5 = Ab$ *and* $L_6 = A$ *and* $L_7 = H$.

Further $K = L f^*LH$, *where* $L = L_2L_3L_4L_5L_6$ *and L has border A.*

□

Notice that in Theorem 3 the number of w in the successive lyns is $8,4,2,1$, for c^* it is $4,2,1,1$, for d^* it is $2,1,1$, for e^* it is $1,1$, and for f^* it is just 1. We are dealing with powers of 2. So if τ is the number of tests, then

$$\tau \leq |Abc^*d^*e^*f^*| + 16|A| + 8|b| + 4|c^*| + 2|d^*| + |e^*| + |H|,$$
$$|K| = 32|A| + 16|b| + 8|c^*| + 4|d^*| + 2|e^*| + |f^*| + |H|.$$

If A,b,c^*,d^*,e^*,f^*,H are integers then $\tau \leq 38$ tests, and $|K| = 64$.

The fact that *lyns* are so strongly structured surprised the author. Let $K^{(1)}, K^{(2)}$ be two *lyns*, each like the K above, with corresponding first *lyns* $L_1^{(1)}, L_1^{(2)}$. Imagine one is testing $K^{(1)} ? K^{(2)}$. One begins by testing $L_1^{(1)} ? L_1^{(2)}$. In view of $L_1 = wc^*wd^*wc^*we^*wc^*wd^*wc^*wf^*$ seen in Theorem 3, it will not cost much to make this test. These facts explain in part why the algorithm is efficient.

Interestingly the *lyn* 1213121415 factors [1213] $\geq_{IS}$ [12] < [1415] and [12131214] $\geq_{IS}$ [1] < [5].

6. Complexity of Algorithm 3.

For each $n \geq 1$, let $\beta(n)$ be the maximum number of integer tests $e\ ?\ g$ used by the algorithm, to find $umf(a)$, over all strings a of integers with $dim(a) = n$. So $\beta(n)/n$ is our complexity, and we think has constant 2.

Suppose we have a string a of γ *lyns*, each of dimension δ, over the 26 letters of the alphabet, so $|a| = \gamma\delta$. The expected cost of testing two of these *lyns* is about 1. If Algorithm 3 has complexity 2, the complexity for dimension δ *lyns* is $2/\delta$.

To get a string a needing as many tests as possible, we want a *lyn* which snakes Right, Left, Right,... as many times as possible. So for a given n, we want the $A, b, c^*, d^*, e^*, f^*, H$ in Example 2 and Theorem 3 above, to be as small as possible, so they might as well be 1,2,3, ... This idea is used in Example 4, which is the best example the author could find.

Example 4. Let a be the initial section of $\pi(1)$ below, with $|a| = n = 2^p$ and $p \geq 4$,

$$\pi(1) =$$
$$1|2|1,3|1,2,1,4|1,2,1,3,1,2,1,5|1,2,1,3,1,2,1,4,1,2,1,3,1,2,1,6|\ldots$$

Let $A = p,q,r,s = 1,2,1,3$. On A the algorithm goes $p?q$ so 12 is a *lyn*, then $p?r$ so $12 \geq 1$, then $r?s$ so 13 is a *lyn*, finally $q?s$ shows 1213 is a *lyn*, at cost 4.

The next 6 tests are crucial. Let $B = t,u,v,w = 1,2,1,4$. It goes $p?t$ so $A \geq 1$, then $t?u$ so 12 is *lyn*, then $q?u$ so $A \geq 12$, then $t?v$ so $A \geq 12 \geq 1$, then $v?w$ so 14 is *lyn*, then $u?w$ so B is *lyn*, but we do not yet know $A?B$. These 6 tests are used for 1213, 1214, 1215 and so on, in fact for all tests $x?y$ with at least one of x, y equal to 1 or 2.

We delete every 1 and 2 from $\pi(1)$ to obtain $\pi(3) = 3,4,3,5,3,4,3,6,\ldots$. In other words we delete all runs 121. By the above working, each deleted run costs 6 towards the cost of finding $umf(a)$. The effect of the algorithm on tests $x?y$ which do not involve a 1 or a 2 is the same as finding $umf(\pi(3))$. To obtain $\pi(5)$ we delete every 3 and 4 from $\pi(3)$. In other words we delete all runs 343, each of which cost 6. The effect of the algorithm on tests $x?y$ which do not involve a $1,2,3$ or 4 is the same as finding $umf(\mu(5))$. Since we delete runs of three with cost of 6, the total number of tests is nearly $2n$, but $< 2n$. □

Example 5. Consider the case $\Sigma = \{0,1\}$ with $0 < 1$. Let $A = 00\ldots011$ and $B = 00\ldots001$ with $|A| = |B| = n$. Then $umf(AB) = [A][B]$. Our Algorithm 1 finds, First A is a lyn, Second there is a cut between A and B, and Third that B is a lyn, each with $n-1$ tests. So the binary complexity is $\geq 3(n-1)/2n$. The author could not do better.

Binary *lyns* may warrant further investigation. They could be studied as vectors of even dimension, so (3,2) is *lyn* 00011, and (3,2,1,4) is *lyn* 0001101111.

7. Short Cuts.

These would improve Algorithm 1.

Shortcut 1. Assume $A_1 >_{IS} A_2 = A_3 = \ldots = A_r < A_{r+1}$, with $3 \leq r$. So A_rA_{r+1} is a *lyn*, and we are in the Go Left case. Our next test would normally be A_{r-1} ? A_rA_{r+1}. However Lemma 4 says that $B = A_2A_3 \ldots A_rA_{r+1}$ is a *lyn*. So we can short cut to the test A_1 ? B. (The algorithm would prove that B is a *lyn* in any case.) □

Shortcut 2. Assume $a = A_1A_2 \ldots A_rb$ where A_1 is a *lyn*, and $A_1 = A_2 = \ldots = A_r$, and string $b \neq \lambda$. Let $A_1 = a_1a_2 \ldots a_n$ and $b = b_1b_2 \ldots b_p$. We start testing a_i ? b_i for $i = 1,2,\ldots$ Suppose there is a least i with $a_i \neq b_i$. If this i has $a_i < b_i$ then $A_1A_2 \ldots A_rb_1b_2 \ldots b_i$ is a *lyn* by Lemmas 4,8. If this i has $a_i > b_i$ then there is a cut $umf(a) = [A_1][A_2] \ldots [A_r]umf(b)$. (The algorithm would find these.)
□

8. Further research.

We want more UMFF's (Unique Max Factorization Families, see [3], [4] for definitions and theory). We always take exactly one row from each non-periodic circulant matrix, and nothing else. An UMFF behaves like the family of *lyns*. It was lexicographic order that yielded the *lyns*. In [3] are more than 30 other orders that yield UMFF's. For these, it seems we can adjust our algorithms to get the unique max factorizations. In [4] are all UMFF's, but there are so many, that it may be possible to get an algorithm for only certain ones.

References.

[1] A. Apostolico and M. Crochemore, Optimal canonization of all substrings of a string, *Inform. and Comput.* 95 (1991) 76-95.

[2] K. T. Chen, R. H. Fox and R. C. Lyndon, Free differential calculus, IV - The quotient groups of the lower central series, *Ann. Math.* 68 (1958) 81-95.

[3] D.E. Daykin and J.W. Daykin, Lyndon-like and V-order factorizations of strings, *J. Discrete Algorithms* 1 (2003) 357-365.

[4] D.E.Daykin and J.W. Daykin, Properties and construction of unique maximal factorization families for strings, *Internat. J. Found. Comput. Sci.* Vol. 19, No. 4 (2008) 1073-1084.

[5] J.W. Daykin, C.S. Iliopoulos and W.F. Smyth, Parallel RAM algorithms for factorizing words, *Theoret. Comput. Sci.* 127 (1) (1994) 53-67.

[6] J. P. Duval, Factorizing words over an ordered alphabet, *J. Algorithms* 4 (1983) 363-381.
[7] M. Lothaire, *Combinatorics on Words*, Addison-Wesley, Reading, MA, 1983; 2nd Edition, Cambridge University Press, Cambridge, 1997.
[8] R. C. Lyndon, On Burnside's problem I, *Trans. Amer. Math. Soc.* 77 (1954) 202-215.

Thanks and best wishes to the referees, and to A. Apostolico, M. Crochemore, J.W. Daykin, C.S. Iliopoulos, W.F. Smyth, J. Spurr, and above all to J. P. Duval (some of whose foundational results have been mentioned here.)