Fast Algorithms for Abelian Periods in Words and Greatest Common Divisor Queries

Tomasz Kociumaka

University of Warsaw

High School of Economics, Moscow, January 25, 2014

Joint work with Jakub Radoszewski and Wojciech Rytter
published at STACS 2013
Outline

1. Greatest Common Divisor Queries

2. Abelian Periods
   - Introduction
   - Solutions for constant alphabets
   - Solutions for large alphabets
   - Conclusions
Problem (Greatest Common Divisor)

For a positive integer \( n \) build a data structure that given integers \( x, y \in \{1, \ldots, n\} \) computes \( \gcd(x, y) \).
Problem (Greatest Common Divisor)

For a positive integer $n$ build a data structure that given integers $x, y \in \{1, \ldots, n\}$ computes $\gcd(x, y)$.

RAM model with word size $w = \Omega(\log n)$.
Problem (Greatest Common Divisor)

For a positive integer $n$ build a data structure that given integers $x, y \in \{1, \ldots, n\}$ computes $\gcd(x, y)$.

RAM model with word size $w = \Omega(\log n)$

- memory is a large array,
- each cell contains a single word, i.e. it represents a $w$-bit integer,
- cells are addressed by a consecutive range of $w$-bit integers,
- space complexity is measured in words, i.e. the length of that range,
- arithmetic, comparison and bitwise operations on $w$-bit integers as well as reading and writing to a cell given its address are performed in constant time.
## Previous & our results

<table>
<thead>
<tr>
<th>Euclid’s algorithm</th>
<th>space</th>
<th>construction</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Theorem (Gries & Misra, 1978)

In $O(n)$ time we can find, for all positive integers $k \leq n$, the smallest prime divisor $p$ of $k$ and the largest exponent $\alpha$ such that $p^\alpha$ is a divisor of $k$.

Fact

The number of distinct prime divisors of $k \leq n$ is $O(\log n \log \log n)$. 

Tomasz Kociumaka

Fast Algorithms for Abelian Periods and GCD Queries
Previous & our results

<table>
<thead>
<tr>
<th></th>
<th>space</th>
<th>construction</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclid’s algorithm</td>
<td>-</td>
<td>-</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>precompute answers</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>use factorization</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O\left(\frac{\log n}{\log \log n}\right)$</td>
</tr>
</tbody>
</table>

Theorem (Gries & Misra, 1978)

In $O(n)$ time we can find, for all positive integers $k \leq n$, the smallest prime divisor $p$ of $k$ and the largest exponent $\alpha$ such that $p^\alpha$ is a divisor of $k$.

Fact

The number of distinct prime divisors of $k \leq n$ is $O\left(\frac{\log n}{\log \log n}\right)$. 
Previous & our results

<table>
<thead>
<tr>
<th></th>
<th>space</th>
<th>construction</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclid’s algorithm</td>
<td>-</td>
<td>-</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>precompute answers</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>use factorization</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O\left(\frac{\log n}{\log \log n}\right)$</td>
</tr>
<tr>
<td>this work</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Theorem (Gries & Misra, 1978)

In $O(n)$ time we can find, for all positive integers $k \leq n$, the smallest prime divisor $p$ of $k$ and the largest exponent $\alpha$ such that $p^\alpha$ is a divisor of $k$.

Fact

The number of distinct prime divisors of $k \leq n$ is $O\left(\frac{\log n}{\log \log n}\right)$. 

Computing $gcd(x, y)$ is sometimes easy:

- we can precompute $gcd[x', y']$ for every $x', y' \leq \sqrt{n}$ and then for $x \leq \sqrt{n}$ we can use the precomputed answer $gcd[x, y \mod x]$,
- if $x$ is prime it suffices to check whether $x$ divides $y$. 

**Definition**

Let $k$ be a positive integer. Then $(k_1, k_2, k_3)$ is a special decomposition of $k$ if $k = k_1 \cdot k_2 \cdot k_3$ and each $k_i$ is prime or does not exceed $\sqrt{k}$.

$(2, 64, 64)$ is a special decomposition of $8192$.

$(1, 18, 479)$, $(2, 9, 479)$ and $(3, 6, 479)$ are up to permutations all special decompositions of $8622$. 

Tomasz Kociumaka

Fast Algorithms for Abelian Periods and GCD Queries
Computing $gcd(x, y)$ is sometimes easy:

- we can precompute $gcd[x', y']$ for every $x', y' \leq \sqrt{n}$ and then for $x \leq \sqrt{n}$ we can use the precomputed answer $gcd[x, y \mod x]$,
- if $x$ is prime it suffices to check whether $x$ divides $y$.

**Definition**

Let $k$ be a positive integer. Then $(k_1, k_2, k_3)$ is a special decomposition of $k$ if $k = k_1k_2k_3$ and each $k_i$ is prime or does not exceed $\sqrt{k}$. 

(2, 64, 64) is a special decomposition of 8192. 
(1, 18, 479), (2, 9, 479) and (3, 6, 479) are up to permutations all special decompositions of 8622.
Computing \( \gcd(x, y) \) is sometimes easy:

- we can precompute \( \gcd[x', y'] \) for every \( x', y' \leq \sqrt{n} \) and then for \( x \leq \sqrt{n} \) we can use the precomputed answer \( \gcd[x, y \mod x] \),
- if \( x \) is prime it suffices to check whether \( x \) divides \( y \).

**Definition**

Let \( k \) be a positive integer. Then \((k_1, k_2, k_3)\) is a special decomposition of \( k \) if \( k = k_1k_2k_3 \) and each \( k_i \) is prime or does not exceed \( \sqrt{k} \).

- \((2, 64, 64)\) is a special decomposition of 8192.
- \((1, 18, 479), (2, 9, 479)\) and \((3, 6, 479)\) are up to permutations all special decompositions of 8622.
Lemma

Let $\ell > 1$ be a positive integer, $p$ be the smallest prime divisor of $\ell$ and $k = \frac{\ell}{p}$. A decomposition of $\ell$ can be obtained from a decomposition of $k$ by multiplying the smallest factor by $p$. 
Lemma

Let $\ell > 1$ be a positive integer, $p$ be the smallest prime divisor of $\ell$ and $k = \frac{\ell}{p}$. A decomposition of $\ell$ can be obtained from a decomposition of $k$ by multiplying the smallest factor by $p$.

\[ 2 \quad 2 \quad 2 \quad 3 \quad 5 \quad 7 \quad 853 \]

$\ell = 716520$

$\ell_1 = 1$  $\ell_2 = 1$  $\ell_3 = 1$
Let $\ell > 1$ be a positive integer, $p$ be the smallest prime divisor of $\ell$ and $k = \frac{\ell}{p}$. A decomposition of $\ell$ can be obtained from a decomposition of $k$ by multiplying the smallest factor by $p$.

\[
\begin{align*}
\ell &= 716520 \\
\ell_1 &= 853 \\
\ell_2 &= 1 \\
\ell_3 &= 1
\end{align*}
\]
Lemma

Let \( \ell > 1 \) be a positive integer, \( p \) be the smallest prime divisor of \( \ell \) and \( k = \frac{\ell}{p} \). A decomposition of \( \ell \) can be obtained from a decomposition of \( k \) by multiplying the smallest factor by \( p \).

\[
\ell = 716520
\]

\[
\ell_1 = 853 \quad \ell_2 = 7 \quad \ell_3 = 1
\]
Let $\ell > 1$ be a positive integer, $p$ be the smallest prime divisor of $\ell$ and $k = \frac{\ell}{p}$. A decomposition of $\ell$ can be obtained from a decomposition of $k$ by multiplying the smallest factor by $p$. 

\[ \ell = 716520 \]

\[ \ell_1 = 853 \quad \ell_2 = 7 \quad \ell_3 = 5 \]
Lemma

Let $\ell > 1$ be a positive integer, $p$ be the smallest prime divisor of $\ell$ and $k = \frac{\ell}{p}$. A decomposition of $\ell$ can be obtained from a decomposition of $k$ by multiplying the smallest factor by $p$.

\[ 2 \ 2 \ 2 \ 3 \ 5 \ 7 \ 853 \]
\[
\ell = 716520
\]

\[
\ell_1 = 853 \quad \ell_2 = 7 \quad \ell_3 = 15
\]
Lemma

Let $\ell > 1$ be a positive integer, $p$ be the smallest prime divisor of $\ell$ and $k = \frac{\ell}{p}$. A decomposition of $\ell$ can be obtained from a decomposition of $k$ by multiplying the smallest factor by $p$.

\[ 2 \ 2 \ 2 \ 3 \ 5 \ 7 \ 853 \]

\[ \ell = 716520 \]

\[ 853 \quad 2 \ 7 \quad 3 \ 5 \]

\[ \ell_1 = 853 \quad \ell_2 = 14 \quad \ell_3 = 15 \]
Lemma

Let $\ell > 1$ be a positive integer, $p$ be the smallest prime divisor of $\ell$ and $k = \frac{\ell}{p}$. A decomposition of $\ell$ can be obtained from a decomposition of $k$ by multiplying the smallest factor by $p$.

\[
2 \ 2 \ 2 \ 3 \ 5 \ 7 \ 853
\]

\[
\ell = 716520
\]

\[
\ell_1 = 853 \quad \ell_2 = 28 \quad \ell_3 = 15
\]
Lemma

Let \( \ell > 1 \) be a positive integer, \( p \) be the smallest prime divisor of \( \ell \) and \( k = \frac{\ell}{p} \). A decomposition of \( \ell \) can be obtained from a decomposition of \( k \) by multiplying the smallest factor by \( p \).

\[
\begin{align*}
\ell &= 716520 \\
\ell_1 &= 853 \\
\ell_2 &= 28 \\
\ell_3 &= 30
\end{align*}
\]
Lemma

Let $\ell > 1$ be a positive integer, $p$ be the smallest prime divisor of $\ell$ and $k = \frac{\ell}{p}$. A decomposition of $\ell$ can be obtained from a decomposition of $k$ by multiplying the smallest factor by $p$.

Proof.

Assume that $k = k_1 k_2 k_3$ and $k_1 \leq k_2 \leq k_3$.

- If $k_1 = 1$ then $k_1 \cdot p = p$ is prime.
- Otherwise, $k_1$ is a divisor of $\ell$ and by the definition of $p$ we have $p \leq k_1$. Therefore:

$$ (k_1 p)^2 = k_1^2 p^2 \leq k_3 p \leq k_1 k_2 k_3 p = \ell. $$

Consequently $k_1 p \leq \sqrt{\ell}$.

In both cases $(k_1 p, k_2, k_3)$ is a special decomposition of $\ell$.  \qed
The data structure consists of:

1. precomputed answers for any $x, y \leq \sqrt{n}$, computed with dynamic programming based on the Euclid’s algorithm.
The data structure consists of:

1. precomputed answers for any $x, y \leq \sqrt{n}$,
   - computed with dynamic programming based on the Euclid’s algorithm

2. a special decomposition of each $x \in \{1, \ldots, n\}$.
   - computed with dynamic programming using the Lemma and the algorithm of Gries & Misra to compute the smallest prime divisors.
Algorithm $gcd(x, y)$

$(x_1, x_2, x_3) := \text{decomp}[x];$

g := 1;

for $i := 1$ to 3 do
    if $x_i \leq \sqrt{n}$ then
        $d := gcd[x_i, y \mod x_i];$
    else if $x_i | y$ then $d := x_i;$
    else $d := 1;$

    $g := g \cdot d;$
    $y := y/d;$

return $g;$
Algorithm $\text{gcd}(x, y)$

$(x_1, x_2, x_3) := \text{decomp}[x];$

$g := 1;$

for $i := 1$ to $3$ do

if $x_i \leq \sqrt{n}$ then

$d := \text{gcd}[x_i, y \mod x_i];$

else if $x_i \mid y$ then $d := x_i;$

else $d := 1;$

$g := g \cdot d;$

$y := y/d;$

return $g;$

$x_1 = 28$

$x_2 = 30$

$x_3 = 853$

$y = 337788$
Algorithm $gcd(x, y)$

$(x_1, x_2, x_3) := decomp[x]$;

$g := 1$;

for $i := 1$ to 3 do
    if $x_i \leq \sqrt{n}$ then
        $d := gcd[x_i, y \mod x_i]$;
    else if $x_i \mid y$ then
        $d := x_i$;
    else
        $d := 1$;
    $g := g \cdot d$;
    $y := y / d$;

return $g$;
Algorithm \( \text{gcd}(x, y) \)

\((x_1, x_2, x_3) := \text{decomp}[x]; \)

\(g := 1;\)

\textbf{for} \(i := 1 \text{ to } 3 \textbf{ do}\)

\textbf{if} \(x_i \leq \sqrt{n}\) \textbf{ then}\n
\(d := \text{gcd}[x_i, y \mod x_i];\)

\textbf{else if} \(x_i \mid y\) \textbf{ then}\n
\(d := x_i;\)

\textbf{else}\n
\(d := 1;\)

\(g := g \cdot d;\)

\(y := y / d;\)

\textbf{return} \(g;\)
Algorithm $gcd(x, y)$

$(x_1, x_2, x_3) := \text{decomp}[x];$

$g := 1;$

\textbf{for} $i := 1$ \textbf{to} 3 \textbf{do}

\hspace{1em} \textbf{if} $x_i \leq \sqrt{n}$ \textbf{then}

\hspace{2em} $d := gcd[x_i, y \mod x_i];$

\hspace{1em} \textbf{else if} $x_i \mid y$ \textbf{then} $d := x_i;$

\hspace{1em} \textbf{else} $d := 1;$

$g := g \cdot d;$

$y := y / d;$

\textbf{return} $g;$
Algorithm $gcd(x, y)$

$(x_1, x_2, x_3) := \text{decomp}[x]$;

$g := 1$;

for $i := 1$ to 3 do
    if $x_i \leq \sqrt{n}$ then
        $d := gcd(x_i, y \mod x_i)$;
    else if $x_i \mid y$ then $d := x_i$;
    else $d := 1$;

$g := g \cdot d$;

$y := y/d$;

return $g$;
Algorithm $gcd(x, y)$

$(x_1, x_2, x_3) := \text{decomp}[x]$;

$g := 1$;

for $i := 1$ to 3 do

if $x_i \leq \sqrt{n}$ then

$d := gcd[x_i, y \mod x_i]$;

else if $x_i \mid y$ then $d := x_i$;

else $d := 1$;

$g := g \cdot d$;

$y := y/d$;

return $g$;
Algorithm \( \text{gcd}(x, y) \)

\[
(x_1, x_2, x_3) := \text{decomp}[x];
\]

\[
g := 1;
\]

\[
\text{for } i := 1 \text{ to } 3 \text{ do }
\]

\[
\text{if } x_i \leq \sqrt{n} \text{ then }
\]

\[
d := \gcd[x_i, y \mod x_i];
\]

\[
\text{else if } x_i | y \text{ then } d := x_i;
\]

\[
\text{else } d := 1;
\]

\[
g := g \cdot d;
\]

\[
y := y / d;
\]

\[
\text{return } g;
\]
Final remarks

**Theorem**

Assume $n$ is known in advance. A sequence of $q$ queries $\gcd(x, y)$ with $x, y \in \{0, \ldots, n\}$ can be answered online in $O(n + q)$ total time.

Euclid’s algorithm gives $O(q \log n)$ time which might be better.
Assume $n$ is known in advance. A sequence of $q$ queries $\gcd(x, y)$ with $x, y \in \{0, \ldots, n\}$ can be answered online in $O(n + q)$ total time.

Euclid’s algorithm gives $O(q \log n)$ time which might be better. We can combine our approach with Euclid’s algorithm. Moreover, we actually do not need to know $n$ in advance.

A sequence of $q$ queries $\gcd(x, y)$ with $x, y \in \{0, \ldots, n\}$ can be answered online in $O(q \max(1, \log \frac{n}{q}))$ total time.
Outline

1. Greatest Common Divisor Queries

2. Abelian Periods
   - Introduction
   - Solutions for constant alphabets
   - Solutions for large alphabets
   - Conclusions
Definition

Let $w$ be a word over $\Sigma$. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in $w$.

$$w = a \ b \ b \ a \ c \quad \mathcal{P}(w) = (2, 2, 1)$$
Commutative equivalence and Parikh vectors

**Definition**

Let $w$ be a word over $\Sigma$. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in $w$.

$$w = a\, b\, b\, a\, c \quad \mathcal{P}(w) = (2, 2, 1)$$
Definition

Let \( w \) be a word over \( \Sigma \). A Parikh vector \( P(w) \) counts for each letter \( a \in \Sigma \) its number of occurrences in \( w \).

\[
w = \text{a b b a c} \quad P(w) = (2, 2, 1)
\]
**Definition**

Let $w$ be a word over $\Sigma$. A Parikh vector $P(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in $w$.

\[
w = a \ b \ b \ a \ c \quad P(w) = (2, 2, 1)\]
Commutative equivalence and Parikh vectors

**Definition**

Let \( w \) be a word over \( \Sigma \). A Parikh vector \( \mathcal{P}(w) \) counts for each letter \( a \in \Sigma \) its number of occurrences in \( w \).

\[
\begin{align*}
  w &= a b b a c \\
  \mathcal{P}(w) &= (2, 2, 1)
\end{align*}
\]

**Definition**

Words \( u, w \) are *commutatively equivalent* if \( \mathcal{P}(u) = \mathcal{P}(w) \).

\[
\begin{align*}
  a b b a c &\approx a c b a b \\
  b a b &\not\approx a b a
\end{align*}
\]
Abelian Periods

Definition

Let $w$ be a word. An integer $q$ is:

- a **full** Abelian period of $w$ if $w$ can be partitioned into commutatively equivalent factors of length $q$,

```
| a b a b a c a b | a a b c b a a b |
```

$q = 8 \quad P = (4, 3, 1)$
Abelian Periods

Definition

Let $w$ be a word. An integer $q$ is:

- a **full** Abelian period of $w$ if $w$ can be partitioned into commutatively equivalent factors of length $q$,

- an Abelian period of $w$ if $q$ is a full Abelian period of some extension **to the right** of $w$.

\[
\begin{array}{cccccc}
  a & b & b & a & c & a \\
  a & b & a & a & b & c \\
  b & a & a & b & a & c
\end{array}
\]

\[
q = 6 \quad \mathcal{P} = (3, 2, 1)
\]
Abelian Periods

Definition

Let $w$ be a word. An integer $q$ is:

- a **full** Abelian period of $w$ if $w$ can be partitioned into commutatively equivalent factors of length $q$,

- an Abelian period of $w$ if $q$ is a full Abelian period of some extension to the right of $w$,

- a **weak** Abelian period of $w$ if $q$ is a full Abelian period of some extension of $w$.

\[ q = 5 \quad \mathcal{P} = (2, 2, 1) \]
## Previous & our results

<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Variant</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>Fici et al.</td>
<td>weak</td>
<td>$O(n^2\sigma)$</td>
</tr>
<tr>
<td>2012</td>
<td>Fici et al.</td>
<td>standard</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>$O(n \log \log n)$</td>
</tr>
<tr>
<td>2013</td>
<td>Crochemore et al.</td>
<td>weak</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

*Assumptions: $\Sigma = \{1, \ldots, \sigma\}$, each letter actually occurs (can be achieved by sorting and renaming letters). RAM model (with $w = \Omega(\log n)$).*
<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Variant</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>Fici et al.</td>
<td>weak</td>
<td>$O(n^2\sigma)$</td>
</tr>
<tr>
<td>2012</td>
<td>Fici et al.</td>
<td>standard</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>$O(n \log \log n)$</td>
</tr>
<tr>
<td>2013</td>
<td>Crochemore et al.</td>
<td>weak</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>2013</td>
<td>this work</td>
<td>standard</td>
<td>$O(n \log \log n)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>randomized</td>
<td>$O(n \log \log n + n \log \sigma)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>deterministic</td>
<td>$O(n)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
## Previous & our results

<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Variant</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>Fici et al.</td>
<td>weak</td>
<td>$O(n^2 \sigma)$</td>
</tr>
<tr>
<td>2012</td>
<td>Fici et al.</td>
<td>standard</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>$O(n \log \log n)$</td>
</tr>
<tr>
<td>2013</td>
<td>Crochemore et al.</td>
<td>weak</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>2013</td>
<td>this work</td>
<td>standard</td>
<td>$O(n \log \log n)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>randomized</td>
<td>$O(n \log \log n + n \log \sigma)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>deterministic</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

**Assumptions:**

- $\Sigma = \{1, \ldots, \sigma\}$, each letter actually occurs (can be achieved by sorting and renaming letters)
- RAM model (with $w = \Omega(\log n)$).
Outline

1. Greatest Common Divisor Queries
2. Abelian Periods
   - Introduction
   - Solutions for constant alphabets
   - Solutions for large alphabets
   - Conclusions
Definition

Let $P_i$ be the Parikh vector of $w[1..i]$. We write $i \sim j$ if there exists $c \in \mathbb{R}$ such that $P_i[s] = cP_j[s]$ for each $s \in \Sigma$. 

![Diagram showing proportionality between paths]

3 \sim 9
Definition

Let $P_i$ be the Parikh vector of $w[1..i]$. We write $i \sim j$ if there exists $c \in \mathbb{R}$ such that $P_i[s] = cP_j[s]$ for each $s \in \Sigma$. 

![Diagram](image)
Fact

Let $A = \{ k : k \sim n \}$. An integer $q \mid n$ is a full Abelian period $\iff q \mid k$ and $k \leq n$ implies $k \in A$.

$A = \{2, 4, 6, 8, 12\}$
Fact

Let \( A = \{ k : k \sim n \} \). An integer \( q \mid n \) is a full Abelian period \( \iff q \mid k \) and \( k \leq n \) implies \( k \in A \).

\[ A = \{2, 4, 6, 8, 12\} \]

4 is a full Abelian period.
Fact

Let \( A = \{ k : k \sim n \} \). An integer \( q \mid n \) is a full Abelian period \( \iff q \mid k \) and \( k \leq n \) implies \( k \in A \).

\[ A = \{2, 4, 6, 8, 12\} \]

6 is a full Abelian period.
Fact

Let $A = \{ k : k \sim n \}$. An integer $q \mid n$ is a full Abelian period $\iff q \mid k$ and $k \leq n$ implies $k \in A$.

$A = \{2, 4, 6, 8, 12\}$

2 is not a full Abelian period.
Full Abelian Periods

Fact

Let \( A = \{k : k \sim n\} \). An integer \( q \mid n \) is a full Abelian period \( \iff q \mid k \) and \( k \leq n \) implies \( k \in A \).

Observation

There exists \( k \notin A \) such that \( q \mid k \iff \) there exists \( d \mid n \) such that \( q \mid d \) and \( d = \gcd(k, n) \) for some \( k \notin A \).
Fact

Let $A = \{ k : k \sim n \}$. An integer $q \mid n$ is a full Abelian period $\iff q \mid k$ and $k \leq n$ implies $k \in A$.

Observation

There exists $k \notin A$ such that $q \mid k \iff$ there exists $d \mid n$ such that $q \mid d$ and $d = \gcd(k, n)$ for some $k \notin A$.

```plaintext
D := ∅; X := \{ q : q \mid n \};
foreach k \notin A do D := D \cup \{ \gcd(k, n) \};
foreach q \mid n, d \mid n do
    if q \mid d and d \in D then
        X := X \setminus \{ q \};
return X;
```

The number of pairs $(q, d)$ is $o(n)$, since the number of divisors of $n$ is $o(\sqrt{n} \log n)$.
**Fact**

Let \( A = \{ k : k \sim n \} \). An integer \( q \mid n \) is a full Abelian period \( \iff q \mid k \) and \( k \leq n \) implies \( k \in A \).

**Observation**

There exists \( k \notin A \) such that \( q \mid k \iff \) there exists \( d \mid n \) such that \( q \mid d \) and \( d = \gcd(k, n) \) for some \( k \notin A \).

\[
D := \emptyset; \quad X := \{ q : q \mid n \};
\]

\[
\text{foreach } k \notin A \text{ do } \quad D := D \cup \{ \gcd(k, n) \};
\]

\[
\text{foreach } q \mid n, d \mid n \text{ do }
\]

\[
\text{if } q \mid d \text{ and } d \in D \text{ then }
\]

\[
X := X \setminus \{ q \};
\]

\[
\text{return } X;
\]

The number of pairs \((q, d)\) is \( o(n) \), since the number of divisors of \( n \) is \( o(n^\varepsilon) \).
A positive integer $q \leq n$ is a candidate if $q \sim kq$ for each $k \in \{1, \ldots, \left\lfloor \frac{n}{q} \right\rfloor\}$. 

10 is a candidate
A positive integer $q \leq n$ is a candidate if $q \sim kq$ for each $k \in \{1, \ldots, \left\lfloor \frac{n}{q} \right\rfloor \}$.

8 is a candidate.
A positive integer $q \leq n$ is a candidate if $q \sim kq$ for each $k \in \{1, \ldots, \left\lfloor \frac{n}{q} \right\rfloor\}$.
Fact

A candidate $q$ is an Abelian period of $w$ if

$$
\mathcal{P}(w[1 \ldots q]) \geq \mathcal{P}(w[kq + 1 \ldots n]), \text{ where } k = \left\lfloor \frac{n}{q} \right\rfloor
$$

and $\geq$ denotes the component-wise order.

10 is an Abelian period
Fact

A candidate $q$ is an Abelian period of $w$ if
\[ P(w[1\ldots q]) \geq P(w[kq + 1\ldots n]), \text{ where } k = \left\lfloor \frac{n}{q} \right\rfloor \]
and $\geq$ denotes the component-wise order.

8 is not an Abelian period
**Fact**

A candidate $q$ is an Abelian period of $w$ if

$$P(w[1 \ldots q]) \geq P(w[kq + 1 \ldots n]),$$

where $k = \left\lfloor \frac{n}{q} \right\rfloor$

and $\geq$ denotes the component-wise order.

Consequently in $O(n)$ time we can select Abelian periods among all candidates.
Lemma

The set $C$ of all candidates can be computed in $O(n \log \log n)$. 

\begin{align*}
C := & \{1, \ldots, n\}; \\
\text{for } q := n \text{ down to } 1 \text{ do } \quad \text{(⋆)} \\
\quad \text{foreach } p \in \text{Primes}, \quad \text{if } q \not\sim pq \text{ or } pq \not\in C \text{ then } \quad \text{C := C \setminus \{q\}; } \\
\quad \text{return } C; \\
\end{align*}

For a fixed $p \in \text{Primes}$ (⋆) is executed at most $n^p$ times, in total we have

$\sum_{p \in \text{Primes}, p \leq n} n^p = O(n \log \log n)$. 

Tomasz Kociumaka
Fast Algorithms for Abelian Periods and GCD Queries 18/31
Computing candidates

**Lemma**

The set $C$ of all candidates can be computed in $O(n \log \log n)$.

**Observation**

$q \in C \iff \forall k \in \mathbb{Z}_+ : kq \leq n \quad q \sim kq \iff \forall p \in \text{Primes} : pq \leq n \quad (q \sim pq \land pq \in C)$. 
Computing candidates

**Lemma**

The set $C$ of all candidates can be computed in $O(n \log \log n)$.

**Observation**

$q \in C \iff \forall k \in \mathbb{Z}_+: kq \leq n \iff q \sim kq \iff \forall p \in \text{Primes}: pq \leq n (q \sim pq \land pq \in C)$.

\[
C := \{1, \ldots, n\}; \\
\text{for } q := n \text{ downto } 1 \text{ do} \\
\quad \text{foreach } p \in \text{Primes}, \ pq \leq n \text{ do} \\
\quad \quad \text{if } q \not\sim pq \text{ or } pq \notin C \text{ then} \\
\quad \quad \quad C := C \setminus \{q\}; \\
\text{return } C ;
\]
Computing candidates

Lemma

The set $C$ of all candidates can be computed in $O(n \log \log n)$.

Observation

$q \in C \iff \forall k \in \mathbb{Z}_+: \ kq \leq n \quad q \sim kq \iff \forall p \in \text{Primes}: \ pq \leq n \ (q \sim pq \land pq \in C)$. 

$C := \{1, \ldots, n\}$;

for $q := n$ downto 1 do

foreach $p \in \text{Primes}, \ pq \leq n$ do

(*): if $q \not\sim pq$ or $pq \not\in C$ then

$C := C \setminus \{q\}$;

return $C$;

For a fixed $p \in \text{Primes}$ (*) is executed at most $\frac{n}{p}$ times, in total we have $\sum_{p \in \text{Primes}, \ p \leq n} \frac{n}{p} = O(n \log \log n)$. 
Theorem

Full Abelian periods of a word of length $n$ over a constant-size alphabet can be computed in $O(n)$ time.

Theorem

Standard Abelian periods of a word of length $n$ over a constant-size alphabet can be computed in $O(n \log \log n)$ time using $O(n)$ space.
Outline

1 Greatest Common Divisor Queries

2 Abelian Periods
   - Introduction
   - Solutions for constant alphabets
   - Solutions for large alphabets
   - Conclusions
Issues with previous solutions

- We cannot afford storing all $P_i = P(w[1..i])$ explicitly.
Issues with previous solutions

- We cannot afford storing all $P_i = P(w[1..i])$ explicitly.
- For Full Abelian periods we need to:
  - test proportionality with the whole word ($k \sim n$).
Issues with previous solutions

- We cannot afford storing all $P_i = P(w[1..i])$ explicitly.
- For Full Abelian periods we need to:
  - test proportionality with the whole word ($k \sim n$).
- For Standard Abelian periods we need to:
  - test proportionality of arbitrary prefixes,
  - check whether a candidate is a period.
Parikh vector $P_{i+1}$ differs from $P_i$ only at a single coordinate.

<table>
<thead>
<tr>
<th>w</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Parikh vector $P_{i+1}$ differs from $P_i$ only at a single coordinate.

**Definition**

A sequence $\xi = (\sigma_1, \ldots, \sigma_r)$ of elementary operations of the form "$v[j] := x$" is a diff-representation of vector sequence $\bar{v} = (\bar{v}_0, \ldots, \bar{v}_r)$, where $\bar{v}_0 = \bar{0}$ and $\bar{v}_{i+1}$ is obtained from $\bar{v}_i$ by applying $\sigma_i$.
Diff-representation of sequences

Parikh vector $P_{i+1}$ differs from $P_i$ only at a single coordinate.

**Definition**

A sequence $\xi = (\sigma_1, \ldots, \sigma_r)$ of elementary operations of the form "$v[j] := x$" is a diff-representation of vector sequence $\bar{v} = (\bar{v}_0, \ldots, \bar{v}_r)$, where $\bar{v}_0 = \bar{0}$ and $\bar{v}_{i+1}$ is obtained from $\bar{v}_i$ by applying $\sigma_i$.

$\xi$ is also regarded a diff-representation of all subsequences of $\bar{v}$. Now vector sequence has a diff-representation, potentially much longer than the sequence itself.

<table>
<thead>
<tr>
<th>$w$</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
We *normalize* the Parikh vectors to test equality instead of proportionality. The diff-representation should remain small.

<table>
<thead>
<tr>
<th>$w$</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
We normalize the Parikh vectors to test equality instead of proportionality. The diff-representation should remain small.

We could consider $\frac{1}{i}P_i$, but the diff-representation could be $\Theta(n\sigma)$.

<table>
<thead>
<tr>
<th>$w$</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>a</td>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
</tr>
<tr>
<td>a</td>
<td>1/2</td>
<td>1/4</td>
<td>1/4</td>
</tr>
<tr>
<td>a</td>
<td>3/5</td>
<td>1/5</td>
<td>1/5</td>
</tr>
<tr>
<td>b</td>
<td>1/2</td>
<td>1/3</td>
<td>1/6</td>
</tr>
<tr>
<td>a</td>
<td>4/7</td>
<td>2/7</td>
<td>1/7</td>
</tr>
<tr>
<td>c</td>
<td>1/8</td>
<td>1/4</td>
<td>1/4</td>
</tr>
<tr>
<td>a</td>
<td>5/9</td>
<td>2/9</td>
<td>2/9</td>
</tr>
<tr>
<td>a</td>
<td>3/5</td>
<td>1/5</td>
<td>1/5</td>
</tr>
<tr>
<td>c</td>
<td>6/11</td>
<td>2/11</td>
<td>3/11</td>
</tr>
<tr>
<td>b</td>
<td>1/2</td>
<td>1/4</td>
<td>1/4</td>
</tr>
</tbody>
</table>
Instead, we fix a letter \( s \in \Sigma \) and consider vectors \( \gamma_i = \frac{1}{P_i[s]} P_i \).

If \( s \) is the least frequent letter, the diff-representation is of size \( O(n + \sigma \frac{n}{\sigma}) = O(n) \).

<table>
<thead>
<tr>
<th>( w )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b )</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( c )</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( a )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( a )</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( a )</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( b )</td>
<td>3/2</td>
<td>1</td>
<td>1/2</td>
</tr>
<tr>
<td>( a )</td>
<td>4/2</td>
<td>1</td>
<td>1/2</td>
</tr>
<tr>
<td>( c )</td>
<td>4/2</td>
<td>1</td>
<td>2/2</td>
</tr>
<tr>
<td>( a )</td>
<td>5/2</td>
<td>1</td>
<td>2/2</td>
</tr>
<tr>
<td>( a )</td>
<td>6/2</td>
<td>1</td>
<td>2/2</td>
</tr>
<tr>
<td>( c )</td>
<td>6/2</td>
<td>1</td>
<td>3/2</td>
</tr>
<tr>
<td>( b )</td>
<td>6/3</td>
<td>1</td>
<td>3/3</td>
</tr>
</tbody>
</table>
Instead, we fix a letter \( s \in \Sigma \) and consider vectors \( \gamma_i = \frac{1}{P_i[s]} P_i \).

If \( s \) is the least frequent letter, the diff-representation is of size \( O(n + \sigma \frac{n}{\sigma}) = O(n) \).

\( \gamma_i \) is undefined for \( i < i_0 \), where the first occurrence of \( s \) at \( i_0 \), but a period must contain all letters, so testing proportionality of vectors \( P_i, P_j \) with \( i, j < i_0 \) is unnecessary.
Instead, we fix a letter \( s \in \Sigma \) and consider vectors \( \gamma_i = \frac{1}{P_i[s]} P_i \).

If \( s \) is the least frequent letter, the diff-representation is of size \( O(n + \sigma \frac{n}{\sigma}) = O(n) \).

\( \gamma_i \) is undefined for \( i < i_0 \), where the first occurrence of \( s \) at \( i_0 \), but a period must contain all letters, so testing proportionality of vectors \( P_i, P_j \) with \( i, j < i_0 \) is unnecessary.

Reduced fractions can be replaced by pairs of integers: the nominator and the denominator. Gcd queries are used to reduce fractions.
Efficient equality testing

We are given a diff-representation of length \( r = O(n) \) of a vector sequence, where each vector has dimension \( m = O(\sigma) \) and nonnegative integer values up to \( n \).

Problem

For a fixed vector find which vectors in the sequence are equal to that vector.

A simple \( O(r + m) \) time solution: maintain the number of coordinates which are equal.

Problem

Preprocess the sequence to answer queries: is the \( i \)-th vector in the sequence equal to the \( j \)-th one?

Idea: compute a fingerprint (an integer of polynomial magnitude) of each vector and compare fingerprints instead of vectors.
Efficient equality testing

We are given a diff-representation of length $r = O(n)$ of a vector sequence, where each vector has dimension $m = O(\sigma)$ and nonnegative integer values up to $n$.

**Problem**

*For a fixed vector find which vectors in the sequence are equal to that vector.*
Efficient equality testing

We are given a diff-representation of length $r = O(n)$ of a vector sequence, where each vector has dimension $m = O(\sigma)$ and nonnegative integer values up to $n$.

### Problem

*For a fixed vector find which vectors in the sequence are equal to that vector.*

A simple $O(r + m)$ time solution: maintain the number of coordinates which are equal.
Efficient equality testing

We are given a diff-representation of length $r = O(n)$ of a vector sequence, where each vector has dimension $m = O(\sigma)$ and nonnegative integer values up to $n$.

**Problem**

For a fixed vector find which vectors in the sequence are equal to that vector.

A simple $O(r + m)$ time solution: maintain the number of coordinates which are equal.

**Problem**

Preprocess the sequence to answer queries: is the $i$-th vector in the sequence equal to the $j$-th one?
Efficient equality testing

We are given a diff-representation of length $r = O(n)$ of a vector sequence, where each vector has dimension $m = O(\sigma)$ and nonnegative integer values up to $n$.

Problem

For a fixed vector find which vectors in the sequence are equal to that vector.

A simple $O(r + m)$ time solution: maintain the number of coordinates which are equal.

Problem

Preprocess the sequence to answer queries: is the $i$-th vector in the sequence equal to the $j$-th one?

Idea: compute a fingerprint (an integer of polynomial magnitude) of each vector and compare fingerprints instead of vectors.
Karp-Rabin fingerprints

With a vector $\vec{v} = (v_0, \ldots, v_{m-1})$ we associate a polynomial $Q_v(x) = \sum_{i=0}^{m-1} v_i x^i$. 

Lemma

Let $\vec{v}_1, \ldots, \vec{v}_r$ be vectors in $\{0, \ldots, n\}^m$. Let $p > \max(n, (m+r)c+3)$ be a number, where $c$ is a positive constant, and let $x_0 \in \mathbb{Z}_p$ be chosen uniformly at random. Then $Q_{\vec{v}_i}(x_0)$ (computed in $\mathbb{Z}_p$) gives collision-free fingerprints with probability at least $1 - \frac{1}{(m+r)c}$. 

Proof.

Single collision has probability at most $m/p$, then apply the union bound.

To efficiently determine $Q_{\vec{v}_{i+1}}$ using $Q_{\vec{v}_i}$ we precompute $x_j^0$ for all $j \in \{0, \ldots, m-1\}$. 

Tomasz Kociumaka

Fast Algorithms for Abelian Periods and GCD Queries 25/31
Karp-Rabin fingerprints

With a vector $\vec{v} = (v_0, \ldots, v_{m-1})$ we associate a polynomial $Q_v(x) = \sum_{i=0}^{m-1} v_i x^i$.

**Lemma**

Let $\vec{v}_1, \ldots, \vec{v}_r$ be vectors in $\{0, \ldots, n\}^m$. Let $p > \max(n, (m + r)^{c+3})$ be a number, where $c$ is a positive constant, and let $x_0 \in \mathbb{Z}_p$ be chosen uniformly at random. Then $Q_{\vec{v}_i}(x_0)$ (computed in $\mathbb{Z}_p$) gives collision-free fingerprints with probability at least $1 - \frac{1}{(m+r)^c}$. 

Tomasz Kociumaka

Fast Algorithms for Abelian Periods and GCD Queries 25/31
Karp-Rabin fingerprints

With a vector $\bar{v} = (v_0, \ldots, v_{m-1})$ we associate a polynomial

$$Q_v(x) = \sum_{i=0}^{m-1} v_i x^i.$$ 

Lemma

Let $\bar{v}_1, \ldots, \bar{v}_r$ be vectors in $\{0, \ldots, n\}^m$. Let $p > \max(n, (m + r)^{c+3})$ be a number, where $c$ is a positive constant, and let $x_0 \in \mathbb{Z}_p$ be chosen uniformly at random. Then $Q_{\bar{v}_i}(x_0)$ (computed in $\mathbb{Z}_p$) gives collision-free fingerprints with probability at least $1 - \frac{1}{(m+r)^c}$.

Proof.

Single collision has probability at most $\frac{m}{p}$, then apply the union bound.
With a vector $\bar{v} = (v_0, \ldots, v_{m-1})$ we associate a polynomial $Q_v(x) = \sum_{i=0}^{m-1} v_i x^i$.

**Lemma**

Let $\bar{v}_1, \ldots, \bar{v}_r$ be vectors in $\{0, \ldots, n\}^m$. Let $p > \max(n, (m + r)^{c+3})$ be a number, where $c$ is a positive constant, and let $x_0 \in \mathbb{Z}_p$ be chosen uniformly at random. Then $Q_{\bar{v}_i}(x_0)$ (computed in $\mathbb{Z}_p$) gives collision-free fingerprints with probability at least $1 - \frac{1}{(m+r)^c}$.

**Proof.**

Single collision has probability at most $\frac{m}{p}$, then apply the union bound.

To efficiently determine $Q_{\bar{v}_{i+1}}$ using $Q_{\bar{v}_i}$ we precompute $x_0^j$ for all $j \in \{0, \ldots, m - 1\}$.
Efficient proportionality testing

Lemma

The set \( \{ k : k \sim n \} \) can be determined in \( O(n) \) time.

Lemma

After \( O(n) \)-time randomized preprocessing, tests \( i \sim j \) (unless \( i, j < i_0 \)) can be performed in \( O(1) \) time.

The preprocessing is Monte Carlo and works with high probability (i.e. the error probability is inverse polynomial).
Efficient proportionality testing

Lemma

The set \( \{ k : k \sim n \} \) can be determined in \( O(n) \) time.

Lemma

After \( O(n) \)-time randomized preprocessing, tests \( i \sim j \) (unless \( i, j < i_0 \)) can be performed in \( O(1) \) time.

The preprocessing is Monte Carlo and works with high probability (i.e. the error probability is inverse polynomial).

Lemma

After \( O(n \log \sigma) \)-time deterministic preprocessing, tests \( i \sim j \) (unless \( i, j < i_0 \)) can be performed in \( O(1) \) time.
Sketch of the idea

Lemma

After $O(n \log \sigma)$-time deterministic preprocessing, tests $i \sim j$ (unless $i, j < i_0$) can be performed in $O(1)$ time.

Proof idea – divide and conquer:

- split coordinates into two halves,
- split the diff-representation into operations involving halves,
- recurse to obtain fingerprints of halves,
- use linear-time sorting to obtain a (small) fingerprint out of a pair of fingerprints for halves.
Fact

A candidate $q$ is an Abelian period of $w$ if

$$
P(w[1\ldots q]) \geq P(w[kq + 1\ldots n]), \text{ where } k = \left\lfloor \frac{n}{q} \right\rfloor$$

and $\geq$ denotes the component-wise order.
Fact

A candidate $q$ is an Abelian period of $w$ if

$$\mathcal{P}(w[kq - q + 1 \ldots kq]) \geq \mathcal{P}(w[kq + 1 \ldots n]),$$

where $k = \left\lfloor \frac{n}{q} \right\rfloor$

and $\geq$ denotes the component-wise order.
Testing candidates

**Fact**

A candidate $q$ is an Abelian period of $w$ if

$$\mathcal{P}(w[kq - q + 1 \ldots kq]) \geq \mathcal{P}(w[kq + 1 \ldots n]),$$

where $k = \left\lfloor \frac{n}{q} \right\rfloor$

and $\geq$ denotes the component-wise order.

We define the tail table

$$T[i] = \max\{j : \mathcal{P}(w[j \ldots i]) \geq \mathcal{P}(w[i + 1 \ldots n])\}.$$
Testing candidates

**Fact**

A candidate \( q \) is an Abelian period of \( w \) if

\[
P(w[kq - q + 1 \ldots kq]) \geq P(w[kq + 1 \ldots n]), \text{ where } k = \left\lfloor \frac{n}{q} \right\rfloor
\]

and \( \geq \) denotes the component-wise order.

We define the tail table

\[
T[i] = \max\{j : P(w[j \ldots i]) \geq P(w[i + 1 \ldots n])\}.
\]

The tail table can be computed using a simple sliding window approach as \( T[i + 1] \geq T[i] \).

```
a b a b a c a b a a b c b a a a b
```
Testing candidates

Fact

A candidate \( q \) is an Abelian period of \( w \) if
\[
\mathcal{P}(w[kq - q + 1 \ldots kq]) \geq \mathcal{P}(w[kq + 1 \ldots n]), \quad \text{where } k = \left\lfloor \frac{n}{q} \right\rfloor
\]
and \( \geq \) denotes the component-wise order.

We define the tail table

\[
T[i] = \max\{j : \mathcal{P}(w[j \ldots i]) \geq \mathcal{P}(w[i + 1 \ldots n])\}.
\]

The tail table can be computed using a simple sliding window approach as \( T[i + 1] \geq T[i] \).

\[
\begin{array}{cccccccccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{c} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} \\
T[i] & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{i}
\end{array}
\]
A candidate $q$ is an Abelian period of $w$ if 

$$\mathcal{P}(w[kq - q + 1 \ldots kq]) \geq \mathcal{P}(w[kq + 1 \ldots n]),$$

where $k = \left\lfloor \frac{n}{q} \right\rfloor$ and $\geq$ denotes the component-wise order.

We define the tail table

$$T[i] = \max\{j : \mathcal{P}(w[j \ldots i]) \geq \mathcal{P}(w[i + 1 \ldots n])\}.$$

The tail table can be computed using a simple sliding window approach as $T[i + 1] \geq T[i]$.
Testing candidates

**Fact**

A candidate $q$ is an Abelian period of $w$ if

$$P(w[kq - q + 1 \ldots kq]) \geq P(w[kq + 1 \ldots n])$$

where $k = \left\lfloor \frac{n}{q} \right\rfloor$ and $\geq$ denotes the component-wise order.

We define the tail table

$$T[i] = \max\{j : P(w[j \ldots i]) \geq P(w[i + 1 \ldots n])\}.$$

The tail table can be computed using a simple sliding window approach as $T[i + 1] \geq T[i]$. 

```
a b a b a c a b b a a b c b a a b
T[i] i
```
Testing candidates

**Fact**

A candidate $q$ is an Abelian period of $w$ if
\[ P(w[kq - q + 1 \ldots kq]) \geq P(w[kq + 1 \ldots n]), \text{ where } k = \left\lfloor \frac{n}{q} \right\rfloor \]
and $\geq$ denotes the component-wise order.

We define the tail table

\[ T[i] = \max\{j : P(w[j \ldots i]) \geq P(w[i + 1 \ldots n])\}. \]

The tail table can be computed using a simple sliding window approach as $T[i + 1] \geq T[i]$.
Fact

A candidate $q$ is an Abelian period of $w$ if
\[ P(w[kq - q + 1\ldots kq]) \geq P(w[kq + 1\ldots n]), \]
where $k = \left\lfloor \frac{n}{q} \right\rfloor$ and $\geq$ denotes the component-wise order.

We define the tail table

\[ T[i] = \max\{ j : P(w[j \ldots i]) \geq P(w[i + 1 \ldots n]) \}. \]

The tail table can be computed using a simple sliding window approach as $T[i + 1] \geq T[i]$.
A candidate $q$ is an Abelian period of $w$ if
\[
\mathcal{P}(w[kq - q + 1 \ldots kq]) \geq \mathcal{P}(w[kq + 1 \ldots n]), \text{ where } k = \left\lfloor \frac{n}{q} \right\rfloor
\]
and $\geq$ denotes the component-wise order.

We define the tail table

\[
T[i] = \max\{j : \mathcal{P}(w[j \ldots i]) \geq \mathcal{P}(w[i + 1 \ldots n])\}.
\]

The tail table can be computed using a simple sliding window approach as $T[i + 1] \geq T[i]$. 
A candidate $q$ is an Abelian period of $w$ if

$$\mathcal{P}(w[kq - q + 1 \ldots kq]) \geq \mathcal{P}(w[kq + 1 \ldots n]),$$

where $k = \left\lfloor \frac{n}{q} \right\rfloor$ and $\geq$ denotes the component-wise order.

We define the tail table

$$T[i] = \max\{j : \mathcal{P}(w[j \ldots i]) \geq \mathcal{P}(w[i + 1 \ldots n])\}.$$

The tail table can be computed using a simple sliding window approach as $T[i + 1] \geq T[i]$.
A candidate $q$ is an Abelian period of $w$ if
\[ P(w[kq - q + 1 \ldots kq]) \geq P(w[kq + 1 \ldots n]), \text{ where } k = \left\lfloor \frac{n}{q} \right\rfloor \]
and $\geq$ denotes the component-wise order.

We define the tail table
\[ T[i] = \max\{j : P(w[j \ldots i]) \geq P(w[i + 1 \ldots n])\}. \]

The tail table can be computed using a simple sliding window approach as $T[i + 1] \geq T[i]$. 

\[
\begin{array}{cccccccccccc}
  a & b & a & b & a & c & a & b & a & a & b & c & b & a & a & b \\
\end{array}
\]
1 Greatest Common Divisor Queries

2 Abelian Periods
   - Introduction
   - Solutions for constant alphabets
   - Solutions for large alphabets
   - Conclusions
Conclusions

**Theorem**

Let \( w \) be a word of length \( n \) over the alphabet \( \{1, \ldots, \sigma\} \).

Full Abelian periods of \( w \) can be computed in \( O(n) \) time.

**Theorem**

Let \( w \) be a word of length \( n \) over the alphabet \( \{1, \ldots, \sigma\} \).

There exist an \( O(n \log \log n + n \log \sigma) \) time deterministic and an \( O(n \log \log n) \) time randomized algorithm that compute all Abelian periods of \( w \). Both algorithms require \( O(n) \) space.
Thank you for your attention!
Thank you

Thank you for your attention!

Questions?