Part I

Greatest Common Divisor Queries
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 \( \Omega(\log n) \), i.e. constant-time arithmetic operations on \( O(\log n) \)-bit integers.
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 \( \Omega(\log n) \), i.e. constant-time arithmetic operations on \( O(\log n) \)-bit integers.

<table>
<thead>
<tr>
<th></th>
<th>space</th>
<th>construction</th>
<th>query time</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>
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 $y$. 

\[ \textbf{Definition} \]

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

Tomasz Kociumaka
Fast Algorithms for Abelian Periods and GCD Queries 4/17
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 \( k = k_1 k_2 k_3 \) and each \( k_i \) is prime or does not exceed \( \sqrt{k} \).
Queries

The data structure consists of:

- precomputed answers for any \( x, y \leq \sqrt{n} \),
- a special decomposition of each \( x \in \{1, \ldots, n\} \).
The data structure consists of:

- precomputed answers for any \( x, y \leq \sqrt{n} \),
- a special decomposition of each \( x \in \{1, \ldots, n\} \).

**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** \( g; \)
The data structure consists of:

- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

**Algorithm $gcd(x, y)$**

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

\[
g := 1;
\]

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

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

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

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

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

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

\[
\quad y := y / d;
\]

\[
\text{return } g;
\]
The data structure consists of:
- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

**Algorithm $gcd(x, y)$**

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

\[
g := 1;
\]

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

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

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

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

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

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

\[
y := y/d;
\]

\[
\text{return } g;
\]

\[
\text{Algorithm example:}
\]

\[
x_1 = 28
\]

\[
x_2 = 30
\]

\[
x_3 = 853
\]

\[
g = 1
\]

\[
y = 337788
\]

\[
d = 4
\]

\[
x_1 = 28
\]

\[
x_2 = 30
\]

\[
x_3 = 853
\]
The data structure consists of:

- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

**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 $g$;

$x_1 = 28$

$x_2 = 30$

$x_3 = 853$

$y = 84447$

$g = 4$
The data structure consists of:

- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

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

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

- else
  
  $d := 1$;

$g := g \cdot d$;

$y := y / d$;

return $g$;
The data structure consists of:
- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

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 $x_i | y$ then $d := x_i;$

else $d := 1;$

$g := g \cdot d;$

$y := y/d;$

return $g;$
The data structure consists of:

- precomputed answers for any \( x, y \leq \sqrt{n} \),
- a special decomposition of each \( x \in \{1, \ldots, n\} \).

**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** \( g; \)
The data structure consists of:
- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

**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 $g$;

\[x_3 = 853\]
\[x_2 = 30\]
\[x_1 = 28\]
\[y = 33\]

\[2 \quad 2 \quad 3 \quad 853\]

\[g = 10236\]
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 \( p \).

Theorem (Gries & Misra, 1978)

The smallest prime divisors for all positive integers up to \( n \) can be computed in \( O(n) \) time.
Part II

Abelian Periods
Commutative equivalence and Parikh vectors

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 $w$.

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

**Definition**

Let $w$ be a word over $\Sigma$. A Parikh vector $P(w)$ 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 $w$.

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

$$a\ b\ b\ a\ c \approx a\ c\ b\ a\ b\ b\ a\ b \not\approx a\ b\ a$$

Tomasz Kociumaka

Fast Algorithms for Abelian Periods and GCD Queries 8/17
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**

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

$$a \, b \, b \, a \, c \approx a \, c \, b \, a \, b \quad b \, a \, b \not\approx a \, b \, a$$
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 $q$,

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

\[
q = 8 \quad \mathcal{P} = (4, 3, 1)
\]
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 $w$,

\[
\begin{align*}
\text{a b a b a c | a b a a b c | b a a b a c} \\
q = 6 \quad \mathcal{P} = (3, 2, 1)
\end{align*}
\]
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 $w$.

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

$q = 5 \quad \mathcal{P} = (2, 2, 1)$
## Previous 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>Year</td>
<td>Authors</td>
<td>Variant</td>
<td>Time complexity</td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
<td>----------</td>
<td>--------------------------</td>
</tr>
<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></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:

$\Sigma = \{1, \ldots, \sigma\}$

standard RAM model (arrays, arithmetic of $O(\log n)$-bit integers)
<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></td>
<td></td>
<td></td>
<td></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></td>
<td></td>
<td></td>
<td></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></td>
<td>randomized $O(n \log \log n + n \log \sigma)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Assumptions:

- $\Sigma = \{1, \ldots, \sigma\}$
- standard RAM model
  (arrays, arithmetic of $O(\log n)$-bit integers)
Proportionality

**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-url)
Definition

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

\[5 \sim 10\]
Lemma

After $O(n)$ randomized or $O(n \log \sigma)$ deterministic time preprocessing $\sim$ can be tested in constant time.

Fact

The set $[n]_\sim = \{k : k \sim n\}$ can be constructed in $O(n)$ time.
Fact

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

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

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

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

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

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

6 is a full Abelian period.
Fact

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

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

2 is not a full Abelian period.
Fact

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

Observation

There is no $k \notin A$ such that $q \mid k$ $\iff$ there is no
$q'$ such that $q \mid q'$ and $q' = \gcd(k, n)$ for some $k \notin A$. 
Full Abelian Periods

Fact

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

Observation

There is no $k \notin A$ such that $q \mid k \iff$ there is no $q'$ such that $q \mid q'$ and $q' = \gcd(k, n)$ for some $k \notin A$.

1. $A' := \{k : k \not\sim n\}$
2. $X := \{q' : \exists_{k \notin A} \gcd(k, n) = q'\}$
   (iterating over $k \notin A$ and using fast gcd queries)
3. For each $q \mid n$ check whether there exists $q' \in X$ such that $q \mid q'$
Full Abelian Periods

Fact

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

Observation

There is no \( k \not\in A \) such that \( q \mid k \iff \) there is no \( q' \) such that \( q \mid q' \) and \( q' = \gcd(k, n) \) for some \( k \not\in A \).

1. \( A' := \{ k : k \not\sim n \} \)
2. \( X := \{ q' : \exists_{k \not\in A} \gcd(k, n) = q' \} \)
   (iterating over \( k \not\in A \) and using fast gcd queries)
3. For each \( q \mid n \) check whether there exists \( q' \in X \) such that \( q \mid q' \)

The number of pairs \((q, q')\) 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
\[
k \in \left\{ 1, \ldots, \left\lfloor \frac{n}{q} \right\rfloor \right\}.
\]

8 is a candidate.
Definition

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

9 is not a candidate
A simple application of the techniques from weak Abelian periods algorithm gives an $O(n)$ time algorithm computing the set of Abelian periods given the set of candidates.

10 is an Abelian period
A simple application of the techniques from weak Abelian periods algorithm gives an $O(n)$ time algorithm computing the set of Abelian periods given the set of candidates.

8 is not an Abelian period
Lemma

The set $C$ of all candidates can be computed in $O(n \log \log n)$ time provided that $\sim$ can be tested in constant time.
Computing candidates

**Lemma**

The set $C$ of all candidates can be computed in $O(n \log \log n)$ time provided that $\sim$ can be tested in constant time.

**Observation**

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

Recall that primes up to $n$ can be generated in $O(n)$ time.
Computing candidates

**Lemma**

The set $C$ of all candidates can be computed in $O(n \log \log n)$ time provided that $\sim$ can be tested in constant time.

**Observation**

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

Recall that primes up to $n$ can be generated in $O(n)$ time. A fixed $p \in \text{Primes}$ is processed for at most $\frac{n}{p}$ values of $q$, so the total number of operations is bounded by

$$
\sum_{p \in \text{Primes}, p \leq n} \frac{n}{p} = O(n \log \log n).
$$
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 \( O(n \log \log \log n) \) time randomized algorithm that compute all Abelian periods of \( w \). Both algorithms require \( O(n) \) space.
Thank you

Thank you for your attention!