# Quantum Speed-ups for String Synchronizing Sets, Longest Common Substring, and $k$-mismatch Matching

Ce Jin[1] **Jakob Nogler**[2]

[1]MIT

[2]ETH Zurich

March 15, 2024

# Overview

Our contribution are quantum speed-ups for several string problems:

# Overview

Our contribution are quantum speed-ups for several string problems:

- Input string $S$ given as a quantum **oracle** $O_s : |i, b\rangle \mapsto |i, b \oplus S[i]\rangle$

# Overview

Our contribution are quantum speed-ups for several string problems:

- Input string $S$ given as a quantum **oracle** $O_s : |i, b\rangle \mapsto |i, b \oplus S[i]\rangle$
- Stronger than a classical oracle since queries can be made in **superposition**

# Overview

Our contribution are quantum speed-ups for several string problems:

- Input string $S$ given as a quantum **oracle** $O_s : |i, b\rangle \mapsto |i, b \oplus S[i]\rangle$
- Stronger than a classical oracle since queries can be made in **superposition**
- **Query complexity** $Q(n)$: counts number of queries to oracle

# Overview

Our contribution are quantum speed-ups for several string problems:

- Input string $S$ given as a quantum **oracle** $O_s : |i, b\rangle \mapsto |i, b \oplus S[i]\rangle$
- Stronger than a classical oracle since queries can be made in **superposition**
- **Query complexity** $Q(n)$: counts number of queries to oracle
- **Time complexity** $T(n)$: also counts the number of elementary gates

# Longest Common Substring (LCS) Problem

**Input:** two strings $S_1, S_2 \in \Sigma^n$
**Output:** maximum length $d$ such that $S_1[i\,..\,i+d-1] = S_2[j\,..\,j+d-1]$ for some $i, j$

$S_1$ b a c b c a c b b a c a b a n a n a n a s c b a b c b b b a a

$S_2$ b c b a a a b $\underbrace{\text{b a n a n a n a s}}_{d\,=\,9}$ b c a a b a c c a c b c a b a

# Longest Common Substring (LCS) Problem

**Input:** two strings $S_1, S_2 \in \Sigma^n$
**Output:** maximum length $d$ such that $S_1[i\,..\,i+d-1] = S_2[j\,..\,j+d-1]$ for some $i, j$

$$\begin{array}{ll} S_1 & \texttt{b a c b c a c b b a c a } \underline{\texttt{b a n a n a n a s}} \texttt{ c b a b c b b b a a} \\ S_2 & \texttt{b c b a a a b } \underbrace{\texttt{b a n a n a n a s}}_{d=9} \texttt{ b c a a b a c c a c b c a b a} \end{array}$$

Well studied in classical settings:

# Longest Common Substring (LCS) Problem

**Input:** two strings $S_1, S_2 \in \Sigma^n$
**Output:** maximum length $d$ such that $S_1[i \mathinner{.\,.} i+d-1] = S_2[j \mathinner{.\,.} j+d-1]$ for some $i, j$

$$
\begin{array}{ll}
S_1 & \texttt{b a c b c a c b b a c a } \textcolor{red}{\texttt{b a n a n a n a s}} \texttt{ c b a b c b b b a a} \\
S_2 & \texttt{b c b a a a b } \underbrace{\textcolor{red}{\texttt{b a n a n a n a s}}}_{d=9} \texttt{ b c a a b a c c a c b c a b a}
\end{array}
$$

Well studied in classical settings:

- Linear-time algorithm via suffix tree (Weiner'73, Farach'97)

# Longest Common Substring (LCS) Problem

**Input:** two strings $S_1, S_2 \in \Sigma^n$
**Output:** maximum length $d$ such that $S_1[i \,..\, i+d-1] = S_2[j \,..\, j+d-1]$ for some $i, j$

| | |
|---|---|
| $S_1$ | b a c b c a c b b a c a b a n a n a n a s c b a b c b b b a a |
| $S_2$ | b c b a a a b b a n a n a n a s b c a a b a c c a c b c a b a |

$d = 9$

Well studied in classical settings:

- Linear-time algorithm via suffix tree (Weiner'73, Farach'97)
- Time-space trade-off (SV'13, KSV'14, BGKK'20)

# Longest Common Substring (LCS) Problem

**Input:** two strings $S_1, S_2 \in \Sigma^n$
**Output:** maximum length $d$ such that $S_1[i\,..\,i+d-1] = S_2[j\,..\,j+d-1]$ for some $i, j$

$S_1$ b a c b c a c b b a c a b a n a n a n a s c b a b c b b b a a

$S_2$ b c b a a a b $\underbrace{\text{b a n a n a n a s}}_{d=9}$ b c a a b a c c a c b c a b a

Well studied in classical settings:

- Linear-time algorithm via suffix tree (Weiner'73, Farach'97)
- Time-space trade-off (SV'13, KSV'14, BGKK'20)
- Dynamic data structures (ACPR'19, ACPR'20, CGP'20)

# Longest Common Substring (LCS) Problem

**Input:** two strings $S_1, S_2 \in \Sigma^n$
**Output:** maximum length $d$ such that $S_1[i\,..\,i+d-1] = S_2[j\,..\,j+d-1]$ for some $i, j$

$S_1$ b a c b c a c b b a c a b a n a n a n a s c b a b c b b b a a
$S_2$ b c b a a a b b a n a n a n a s b c a a b a c c a c b c a b a

$$\underbrace{\text{b a n a n a n a s}}_{d=9}$$

Well studied in classical settings:

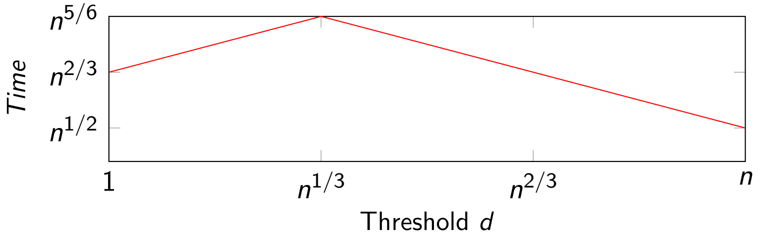
- Linear-time algorithm via suffix tree (Weiner'73, Farach'97)
- Time-space trade-off (SV'13, KSV'14, BGKK'20)
- Dynamic data structures (ACPR'19, ACPR'20, CGP'20)
- Small-alphabet input (CKPR'21)

# Quantum algorithms for LCS

Binary search on **decisional** problem with threshold $1 \leq d \leq n$: does $\mathrm{LCS}(S_1, S_2) \geq d$ hold?

# Quantum algorithms for LCS

Binary search on **decisional** problem with threshold $1 \leq d \leq n$: does $\mathrm{LCS}(S_1, S_2) \geq d$ hold?

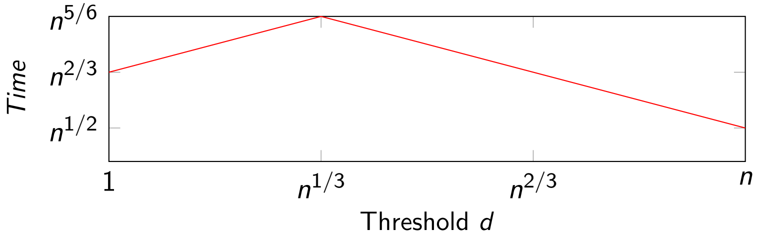



| | LCS with threshold[1] | LCS[1] |
|---|---|---|
| Le Gall and Seddighin (ITCS'22) | $\widetilde{O}(\min\{n^{2/3} \cdot d^{1/2}, n/d^{1/2}\})$ | |

[1]Quantum query complexity and time complexity

# Quantum algorithms for LCS

Binary search on **decisional** problem with threshold $1 \leq d \leq n$: does $\mathrm{LCS}(S_1, S_2) \geq d$ hold?

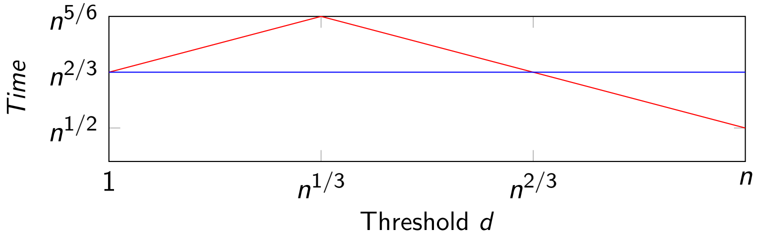



| | LCS with threshold[1] | LCS[1] |
|---|---|---|
| Le Gall and Seddighin (ITCS'22) | $\widetilde{O}(\min\{n^{2/3} \cdot d^{1/2}, n/d^{1/2}\})$ | $\widetilde{O}(n^{5/6})$ |

[1] Quantum query complexity and time complexity

# Quantum algorithms for LCS

Binary search on **decisional** problem with threshold $1 \leq d \leq n$: does $\mathrm{LCS}(S_1, S_2) \geq d$ hold?




| | LCS with threshold[1] | LCS[1] |
|---|---|---|
| Le Gall and Seddighin (ITCS'22) | $\widetilde{O}(\min\{n^{2/3} \cdot d^{1/2}, n/d^{1/2}\})$ | $\widetilde{O}(n^{5/6})$ |
| Akmal and Jin (SODA'22) | $\widetilde{O}(n^{2/3})$ | $\widetilde{O}(n^{2/3})$ |

[1]Quantum query complexity and time complexity

# LCS with threshold

$d = 1 \longrightarrow d = n$

**(Bipartite) Element Distinctness**

**Unstructured Search**

# LCS with threshold

$d = 1$ $\longrightarrow$ $d = n$

**(Bipartite) Element Distinctness**

- Are there $i, j$ such that $S_1[i] = S_2[j]$?

**Unstructured Search**

# LCS with threshold

$d = 1$ $\longrightarrow$ $d = n$

**(Bipartite) Element Distinctness**

- Are there $i, j$ such that $S_1[i] = S_2[j]$?
- $Q(n) = \Theta(n^{2/3})$

**Unstructured Search**

# LCS with threshold

$d = 1$ $\longrightarrow$ $d = n$

**(Bipartite) Element Distinctness**

- Are there $i, j$ such that $S_1[i] = S_2[j]$?
- $Q(n) = \Theta(n^{2/3})$
- Tight lower bound for [AJ'22]

**Unstructured Search**

# LCS with threshold

$d = 1$ ⟶ $d = n$

**(Bipartite) Element Distinctness**

- Are there $i, j$ such that $S_1[i] = S_2[j]$?
- $Q(n) = \Theta(n^{2/3})$
- Tight lower bound for [AJ'22]

**Unstructured Search**

- Does $S_1[i] = S_2[i]$ hold for all $i \in [n]$?

# LCS with threshold

$d = 1 \longrightarrow d = n$

**(Bipartite) Element Distinctness**

- Are there $i, j$ such that $S_1[i] = S_2[j]$?
- $Q(n) = \Theta(n^{2/3})$
- Tight lower bound for [AJ'22]

**Unstructured Search**

- Does $S_1[i] = S_2[i]$ hold for all $i \in [n]$?
- $Q(n) = \Theta(\sqrt{n})$

# LCS with threshold

$d = 1$ ●———————————————→● $d = n$

**(Bipartite) Element Distinctness**

- Are there $i, j$ such that $S_1[i] = S_2[j]$?
- $Q(n) = \Theta(n^{2/3})$
- Tight lower bound for [AJ'22]

**Unstructured Search**

- Does $S_1[i] = S_2[i]$ hold for all $i \in [n]$?
- $Q(n) = \Theta(\sqrt{n})$

By composing the two problems for the intermediate case $1 < d < n$ we obtain:

**Theorem (quantum query lower bound), based on [BHKKLS'11]**

Deciding LCS with threshold $d$ requires $\Omega(n^{2/3}/d^{1/6})$ quantum queries.

# Our results for LCS with threshold

**Theorem (an almost-tight upper bound) [Jin & N'23]**

Given $S_1, S_2 \in \Sigma^n$, there is a quantum algorithm that determines whether $\mathrm{LCS}(S_1, S_2) \geq d$ holds in

$$\widetilde{O}(n^{2/3}/d^{1/6-o(1)})$$

quantum query and time complexity.

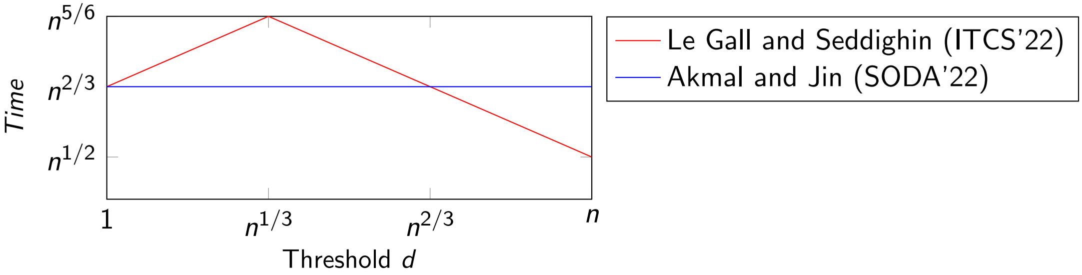
# Our results for LCS with threshold

**Theorem (an almost-tight upper bound) [Jin & N'23]**

Given $S_1, S_2 \in \Sigma^n$, there is a quantum algorithm that determines whether $\mathrm{LCS}(S_1, S_2) \geq d$ holds in

$$\widetilde{O}(n^{2/3}/d^{1/6-o(1)})$$

quantum query and time complexity.



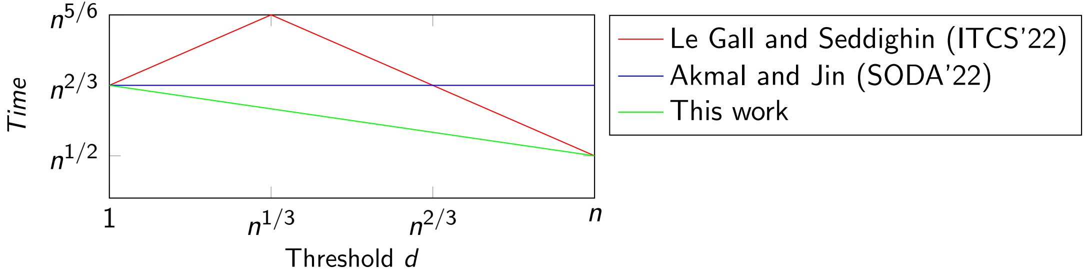
# Our results for LCS with threshold

**Theorem (an almost-tight upper bound) [Jin & N'23]**

Given $S_1, S_2 \in \Sigma^n$, there is a quantum algorithm that determines whether $\mathrm{LCS}(S_1, S_2) \geq d$ holds in

$$\widetilde{O}(n^{2/3}/d^{1/6-o(1)})$$

quantum query and time complexity.



# LCS via Anchoring Technique

An *anchor set* C should satisfy: if $\mathrm{LCS}(S_1, S_2) \geq d$, then $S_1$ and $S_2$ must have a length-$d$ common substring anchored by C.

At least one common position must be included in C

$S_1$ ··· b a c b c a a b a n a n a n a s c b a c b b a c b c b b b a a ···

$S_2$ ··· b c b a a a b b a n a n a n a s b c a a b a c c a c b c a b a ···

# LCS via Anchoring Technique

An *anchor set* C should satisfy: if $\mathrm{LCS}(S_1, S_2) \geq d$, then $S_1$ and $S_2$ must have a length-$d$ common substring anchored by C.

At least one common position must be included in C

$S_1$ ⋯ b a c b c a a b a n a n a n a s c b a c b b a c b c b b b a a ⋯

$S_2$ ⋯ b c b a a a b b a n a n a n a s b c a a b a c c a c b c a b a ⋯

**Theorem [Akmal & Jin'22]**

Given a size-$m$ anchor set such that the $i$-th anchor can be reported in $\mathcal{T}$ quantum time, we can decide $\mathrm{LCS}(S_1, S_2) \geq d$ in $\widetilde{O}(m^{2/3} \cdot (\sqrt{d} + \mathcal{T}))$ quantum time.

# LCS via Anchoring Technique

An *anchor set* C should satisfy: if $\mathrm{LCS}(S_1, S_2) \geq d$, then $S_1$ and $S_2$ must have a length-$d$ common substring anchored by C.

At least one common position must be included in C

$S_1$ $\cdots$ b a c $\underline{\text{b}}$ c a a $\underline{\text{b}}$ a n $\underline{\text{a}}$ n a n $\underline{\text{a}}$ s c b a $\underline{\text{c}}$ b b a c $\underline{\text{b}}$ c b b b a a $\cdots$

$S_2$ $\cdots$ b c b a a a $\underline{\text{b}}$ b a n $\underline{\text{a}}$ n $\underline{\text{a}}$ n a s $\underline{\text{b}}$ c a $\underline{\text{a}}$ b a c c a c b $\underline{\text{c}}$ a b a $\cdots$

**Theorem [Akmal & Jin'22]**

Given a size-$m$ anchor set such that the $i$-th anchor can be reported in $\mathcal{T}$ quantum time, we can decide $\mathrm{LCS}(S_1, S_2) \geq d$ in $\widetilde{O}(m^{2/3} \cdot (\sqrt{d} + \mathcal{T}))$ quantum time.

- [CKPR'21]: Such an anchor set for LCS can be constructed from String Synchronizing Sets [KK'19].

# LCS via Anchoring Technique

An *anchor set* C should satisfy: if $\mathrm{LCS}(S_1, S_2) \geq d$, then $S_1$ and $S_2$ must have a length-$d$ common substring anchored by C.

At least one common position must be included in C

$S_1$ $\cdots$ b a c <u>b</u> c a a <u>b</u> a n <u>a</u> n a n <u>a</u> s c b a <u>c</u> b b a c <u>b</u> c b b b a a $\cdots$

$S_2$ $\cdots$ b c b a a a <u>b</u> b a n <u>a</u> n <u>a</u> n a s <u>b</u> c a <u>a</u> b a c c a c b <u>c</u> a b a $\cdots$

**Theorem [Akmal & Jin'22]**

Given a size-$m$ anchor set such that the $i$-th anchor can be reported in $\mathcal{T}$ quantum time, we can decide $\mathrm{LCS}(S_1, S_2) \geq d$ in $\widetilde{O}(m^{2/3} \cdot (\sqrt{d} + \mathcal{T}))$ quantum time.

- [CKPR'21]: Such an anchor set for LCS can be constructed from String Synchronizing Sets [KK'19].
- Classical construction of synchronizing sets is slow.

# Our results

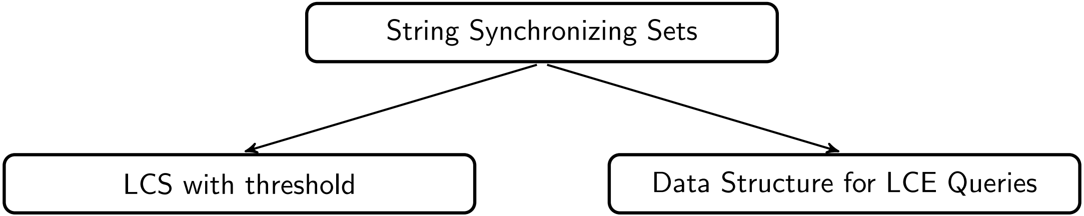
String Synchronizing Sets

# Our results

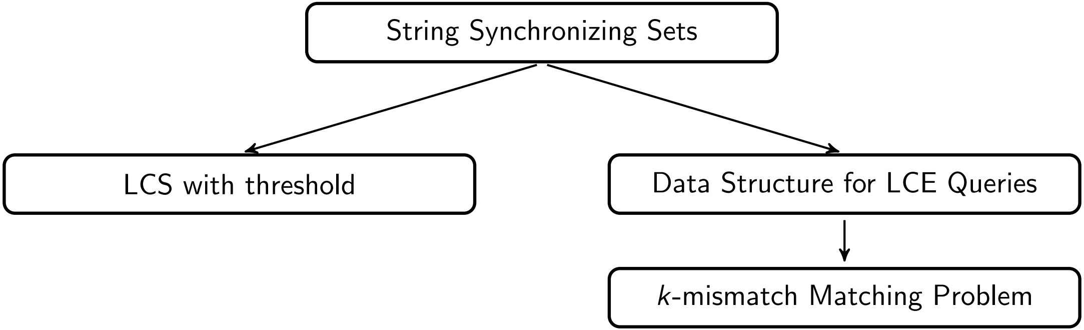
String Synchronizing Sets

LCS with threshold

# Our results


String Synchronizing Sets
LCS with threshold
Data Structure for LCE Queries

# Our results


String Synchronizing Sets
LCS with threshold
Data Structure for LCE Queries
$k$-mismatch Matching Problem

# String Synchronizing Sets [Kempa & Kociumaka'19]

For a string $T[1\,.\,.\,n]$ and a positive integer $1 \le \tau \le n/2$, we say $\mathsf{A} \subseteq [1\,.\,.\,n-2\tau+1]$ is a *$\tau$-synchronizing set of $T$* if it satisfies the following properties:

Example: $\tau = 3$
red: positions in A

$T$ $\cdots$ b a c b c a a a a a a a a a a a a a a b b b b c a c b c a a a $\cdots$

# String Synchronizing Sets [Kempa & Kociumaka'19]

For a string $T[1\,..\,n]$ and a positive integer $1 \leq \tau \leq n/2$, we say $\mathsf{A} \subseteq [1\,..\,n-2\tau+1]$ is a *$\tau$-synchronizing set of $T$* if it satisfies the following properties:

- **Consistency:** If $T[i\,..\,i+2\tau) = T[j\,..\,j+2\tau)$, then $i \in \mathsf{A}$ if and only if $j \in \mathsf{A}$.

Example: $\tau = 3$
red: positions in A

$T$ $\cdots$ b a c b c a a a a a a a a a a a a a a a b b b b c a c b c a a a $\cdots$

# String Synchronizing Sets [Kempa & Kociumaka'19]

For a string $T[1\,..\,n]$ and a positive integer $1 \le \tau \le n/2$, we say $\mathsf{A} \subseteq [1\,..\,n-2\tau+1]$ is a *$\tau$-synchronizing set of $T$* if it satisfies the following properties:

- **Consistency:** If $T[i\,..\,i+2\tau) = T[j\,..\,j+2\tau)$, then $i \in \mathsf{A}$ if and only if $j \in \mathsf{A}$.

Example: $\tau = 3$
red: positions in A

consistency

$T$ $\cdots$ b $\boxed{\text{a c b c a a}}$ a a a a a a a a a a a a a b b b b c $\boxed{\text{a c b c a a}}$ a $\cdots$

# String Synchronizing Sets [Kempa & Kociumaka'19]

For a string $T[1\mathinner{.\,.} n]$ and a positive integer $1 \le \tau \le n/2$, we say $\mathsf{A} \subseteq [1\mathinner{.\,.} n-2\tau+1]$ is a *$\tau$-synchronizing set of $T$* if it satisfies the following properties:

- **Consistency:** If $T[i\mathinner{.\,.} i+2\tau) = T[j\mathinner{.\,.} j+2\tau)$, then $i \in \mathsf{A}$ if and only if $j \in \mathsf{A}$.
- **Density:** For $i \in [1\mathinner{.\,.} n-3\tau+2]$, $\mathsf{A} \cap [i\mathinner{.\,.} i+\tau) = \emptyset$ if and only if $\mathrm{per}(T[i\mathinner{.\,.} i+3\tau-2]) \le \tau/3$.

Example: $\tau = 3$
red: positions in A

consistency

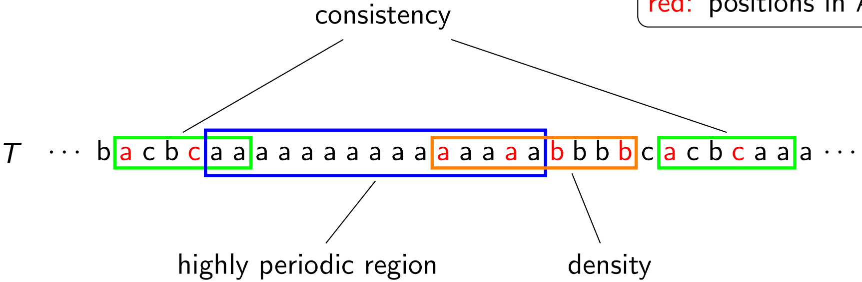
$T$ $\cdots$ b $\boxed{\text{a c b c a a}}$ a a a a a a a a a a a a a b b b b c $\boxed{\text{a c b c a a}}$ a $\cdots$

# String Synchronizing Sets [Kempa & Kociumaka'19]

For a string $T[1\mathinner{.\,.} n]$ and a positive integer $1 \le \tau \le n/2$, we say $\mathsf{A} \subseteq [1\mathinner{.\,.} n-2\tau+1]$ is a *$\tau$-synchronizing set of $T$* if it satisfies the following properties:

- **Consistency:** If $T[i\mathinner{.\,.} i+2\tau) = T[j\mathinner{.\,.} j+2\tau)$, then $i \in \mathsf{A}$ if and only if $j \in \mathsf{A}$.
- **Density:** For $i \in [1\mathinner{.\,.} n-3\tau+2]$, $\mathsf{A} \cap [i\mathinner{.\,.} i+\tau) = \emptyset$ if and only if $\mathrm{per}(T[i\mathinner{.\,.} i+3\tau-2]) \le \tau/3$.

Example: $\tau = 3$
red: positions in A



# Our Result for String Synchronizing Sets

We want (ideally):

- **Sparsity:** small size $\mathsf{A} = O(n/\tau)$.
- **Efficient Computability:** fast reporting time of an element of $\mathsf{A}$.

# Our Result for String Synchronizing Sets

We want (ideally):

- **Sparsity:** small size $\mathsf{A} = O(n/\tau)$.
- **Efficient Computability:** fast reporting time of an element of $\mathsf{A}$.

**Theorem [Jin & N'23]**

There exists a (randomized) $\tau$-synchronizing set $\mathsf{A}$ of size $n/\tau^{1-o(1)}$, such that given an $i$ we can reported each element of $\mathsf{A} \cap [i\,..\,i+\tau]$ in $\widetilde{O}(\tau^{\frac{1}{2}+o(1)})$ quantum query and time complexity.

# Our Result for String Synchronizing Sets

We want (ideally):

- **Sparsity:** small size $\mathsf{A} = O(n/\tau)$.
- **Efficient Computability:** fast reporting time of an element of A.

**Theorem [Jin & N'23]**

There exists a (randomized) $\tau$-synchronizing set A of size $n/\tau^{1-o(1)}$, such that given an $i$ we can reported each element of $\mathsf{A} \cap [i \mathinner{.\,.} i+\tau]$ in $\widetilde{O}(\tau^{\frac{1}{2}+o(1)})$ quantum query and time complexity.

Techniques: [KK'19] + Divide & Conquer + quantum minimum finding + [Vishkin'91]

# Our Result for String Synchronizing Sets

We want (ideally):

- **Sparsity:** small size $\mathrm{A} = O(n/\tau)$.
- **Efficient Computability:** fast reporting time of an element of A.

**Theorem [Jin & N'23]**

There exists a (randomized) $\tau$-synchronizing set A of size $n/\tau^{1-o(1)}$, such that given an $i$ we can reported each element of $\mathrm{A} \cap [i \mathinner{.\,.} i+\tau]$ in $\widetilde{O}(\tau^{\frac{1}{2}+o(1)})$ quantum query and time complexity.

Techniques: [KK'19] + Divide & Conquer + quantum minimum finding + [Vishkin'91]

Many recent applications in classical string algorithms:

# Our Result for String Synchronizing Sets

We want (ideally):

- **Sparsity:** small size $\mathsf{A} = O(n/\tau)$.
- **Efficient Computability:** fast reporting time of an element of A.

**Theorem [Jin & N'23]**

There exists a (randomized) $\tau$-synchronizing set A of size $n/\tau^{1-o(1)}$, such that given an $i$ we can reported each element of $\mathsf{A} \cap [i \mathinner{.\,.} i+\tau]$ in $\widetilde{O}(\tau^{\frac{1}{2}+o(1)})$ quantum query and time complexity.

Techniques: [KK'19] + Divide & Conquer + quantum minimum finding + [Vishkin'91]

Many recent applications in classical string algorithms:

- Sublinear-time Burrows-Wheeler Transform [KK'19]

# Our Result for String Synchronizing Sets

We want (ideally):

- **Sparsity:** small size $\mathsf{A} = O(n/\tau)$.
- **Efficient Computability:** fast reporting time of an element of $\mathsf{A}$.

**Theorem [Jin & N'23]**

There exists a (randomized) $\tau$-synchronizing set $\mathsf{A}$ of size $n/\tau^{1-o(1)}$, such that given an $i$ we can reported each element of $\mathsf{A} \cap [i\,..\,i+\tau]$ in $\widetilde{O}(\tau^{\frac{1}{2}+o(1)})$ quantum query and time complexity.

Techniques: [KK'19] + Divide & Conquer + quantum minimum finding + [Vishkin'91]

Many recent applications in classical string algorithms:

- Sublinear-time Burrows-Wheeler Transform [KK'19]
- Optimal LCE data structure [KK'19]

# Our Result for String Synchronizing Sets

We want (ideally):

- **Sparsity:** small size $\mathsf{A} = O(n/\tau)$.
- **Efficient Computability:** fast reporting time of an element of $\mathsf{A}$.

**Theorem [Jin & N'23]**

There exists a (randomized) $\tau$-synchronizing set $\mathsf{A}$ of size $n/\tau^{1-o(1)}$, such that given an $i$ we can reported each element of $\mathsf{A} \cap [i \,..\, i+\tau]$ in $\widetilde{O}(\tau^{\frac{1}{2}+o(1)})$ quantum query and time complexity.

Techniques: [KK'19] + Divide & Conquer + quantum minimum finding + [Vishkin'91]

Many recent applications in classical string algorithms:

- Sublinear-time Burrows-Wheeler Transform [KK'19]
- Optimal LCE data structure [KK'19]
- Longest Common Substring [CKPR'21]

# Our Result for String Synchronizing Sets

We want (ideally):

- **Sparsity:** small size $\mathsf{A} = O(n/\tau)$.
- **Efficient Computability:** fast reporting time of an element of $\mathsf{A}$.

**Theorem [Jin & N'23]**

There exists a (randomized) $\tau$-synchronizing set $\mathsf{A}$ of size $n/\tau^{1-o(1)}$, such that given an $i$ we can reported each element of $\mathsf{A} \cap [i \mathinner{.\,.} i+\tau]$ in $\widetilde{O}(\tau^{\frac{1}{2}+o(1)})$ quantum query and time complexity.

Techniques: [KK'19] + Divide & Conquer + quantum minimum finding + [Vishkin'91]

Many recent applications in classical string algorithms:

- Sublinear-time Burrows-Wheeler Transform [KK'19]
- Optimal LCE data structure [KK'19]
- Longest Common Substring [CKPR'21]
- Dynamic & Compressed suffix arrays [KK'22], [KK'23]

# Data Structure for LCE Queries

**Input:** $T \in \Sigma^n$ and $i, j \in [n]$
**Output:** length of the longest common prefix of $T[i\,..\,n]$ and $T[j\,..\,n]$

# Data Structure for LCE Queries

**Input:** $T \in \Sigma^n$ and $i, j \in [n]$
**Output:** length of the longest common prefix of $T[i \mathinner{.\,.} n]$ and $T[j \mathinner{.\,.} n]$

**Theorem [Jin & N'23], based on [KK'19]**

Given an integer $1 \le \tau \le n/2$ there is a quantum algorithm such that:

# Data Structure for LCE Queries

**Input:** $T \in \Sigma^n$ and $i, j \in [n]$
**Output:** length of the longest common prefix of $T[i \,..\, n]$ and $T[j \,..\, n]$

**Theorem [Jin & N'23], based on [KK'19]**

Given an integer $1 \leq \tau \leq n/2$ there is a quantum algorithm such that:

- it outputs in $\mathcal{T}_{\text{prep}} = \widetilde{O}(n/\tau^{\frac{1}{2}-o(1)})$ a data structure $D$ (with classical representation).

# Data Structure for LCE Queries

**Input:** $T \in \Sigma^n$ and $i, j \in [n]$
**Output:** length of the longest common prefix of $T[i\,..\,n]$ and $T[j\,..\,n]$

**Theorem [Jin & N'23], based on [KK'19]**

Given an integer $1 \le \tau \le n/2$ there is a quantum algorithm such that:

- it outputs in $\mathcal{T}_{\mathrm{prep}} = \widetilde{O}(n/\tau^{\frac{1}{2}-o(1)})$ a data structure $D$ (with classical representation).
- Given quantum random access to $D$, we can answer LCE queries in $\mathcal{T}_{\mathrm{ans}} = \widetilde{O}(\sqrt{\tau})$ quantum time.

# Data Structure for LCE Queries

**Input:** $T \in \Sigma^n$ and $i, j \in [n]$
**Output:** length of the longest common prefix of $T[i \mathinner{..} n]$ and $T[j \mathinner{..} n]$

**Theorem [Jin & N'23], based on [KK'19]**

Given an integer $1 \le \tau \le n/2$ there is a quantum algorithm such that:

- it outputs in $\mathcal{T}_{\text{prep}} = \widetilde{O}(n/\tau^{\frac{1}{2}-o(1)})$ a data structure $D$ (with classical representation).
- Given quantum random access to $D$, we can answer LCE queries in $\mathcal{T}_{\text{ans}} = \widetilde{O}(\sqrt{\tau})$ quantum time.

$\mathcal{T}_{\text{prep}} = 0$, $\mathcal{T}_{\text{ans}} = \widetilde{O}(\sqrt{n})$ ——— $\mathcal{T}_{\text{prep}} = O(n)$, $\mathcal{T}_{\text{ans}} = O(1)$

Tradeoff: $(\mathcal{T}_{\text{prep}} + \sqrt{n}) \cdot (\mathcal{T}_{\text{ans}} + 1) \ge \widetilde{\Omega}(n)$

# The (Hamming) $k$-mismatch Matching Problem

**Input:** a text $T \in \Sigma^n$, a pattern $P \in \Sigma^m$, and a threshold $k \in [m]$
**Output:** does $T$ contain a substring with hamming distance at most $k$ from $P$?

# The (Hamming) $k$-mismatch Matching Problem

**Input:** a text $T \in \Sigma^n$, a pattern $P \in \Sigma^m$, and a threshold $k \in [m]$
**Output:** does $T$ contain a substring with hamming distance at most $k$ from $P$?

**Quantum Query Lower Bound:** $\Omega(\sqrt{kn})$

# The (Hamming) $k$-mismatch Matching Problem

**Input:** a text $T \in \Sigma^{2m}$, a pattern $P \in \Sigma^m$, and a threshold $k \in [m]$
**Output:** does $T$ contain a substring with hamming distance at most $k$ from $P$?

**Quantum Query Lower Bound:** $\Omega(\sqrt{kn})$

# The (Hamming) $k$-mismatch Matching Problem

**Input:** a text $T \in \Sigma^{2m}$, a pattern $P \in \Sigma^m$, and a threshold $k \in [m]$
**Output:** does $T$ contain a substring with hamming distance at most $k$ from $P$?

**Quantum Query Lower Bound:** $\Omega(\sqrt{kn})$

[Bringmann, Künnemann & Wellnitz'19] [Charalampopoulus, Kociumaka & Wellnitz'20]
$\Rightarrow$ Structural insights: either there are few matches or $P$ is *almost* periodic.

# The (Hamming) $k$-mismatch Matching Problem

**Input:** a text $T \in \Sigma^{2m}$, a pattern $P \in \Sigma^m$, and a threshold $k \in [m]$
**Output:** does $T$ contain a substring with hamming distance at most $k$ from $P$?

**Quantum Query Lower Bound:** $\Omega(\sqrt{kn})$

[Bringmann, Künnemann & Wellnitz'19] [Charalampopoulus, Kociumaka & Wellnitz'20]
$\Rightarrow$ Structural insights: either there are few matches or $P$ is *almost* periodic.

**Theorem [CKW'20]**

At least one of the following holds:

# The (Hamming) $k$-mismatch Matching Problem

**Input:** a text $T \in \Sigma^{2m}$, a pattern $P \in \Sigma^m$, and a threshold $k \in [m]$
**Output:** does $T$ contain a substring with hamming distance at most $k$ from $P$?

**Quantum Query Lower Bound:** $\Omega(\sqrt{kn})$

[Bringmann, Künnemann & Wellnitz'19] [Charalampopoulus, Kociumaka & Wellnitz'20]
$\Rightarrow$ Structural insights: either there are few matches or $P$ is *almost* periodic.

**Theorem [CKW'20]**

At least one of the following holds:

(i) The number of $k$-mismatch occurrences of $P$ in $T$ is $O(k)$.

# The (Hamming) $k$-mismatch Matching Problem

**Input:** a text $T \in \Sigma^{2m}$, a pattern $P \in \Sigma^m$, and a threshold $k \in [m]$
**Output:** does $T$ contain a substring with hamming distance at most $k$ from $P$?

**Quantum Query Lower Bound:** $\Omega(\sqrt{kn})$

[Bringmann, Künnemann & Wellnitz'19] [Charalampopoulus, Kociumaka & Wellnitz'20]
$\Rightarrow$ Structural insights: either there are few matches or $P$ is *almost* periodic.

**Theorem [CKW'20]**

At least one of the following holds:

(i) The number of $k$-mismatch occurrences of $P$ in $T$ is $O(k)$.

(ii) There are $O(k)$ mismatches between $P$ and the periodic extension of a string $Q$.

# The (Hamming) $k$-mismatch Matching Problem

**Input:** a text $T \in \Sigma^{2m}$, a pattern $P \in \Sigma^m$, and a threshold $k \in [m]$
**Output:** does $T$ contain a substring with hamming distance at most $k$ from $P$?

**Quantum Query Lower Bound:** $\Omega(\sqrt{kn})$

[Bringmann, Künnemann & Wellnitz'19] [Charalampopoulus, Kociumaka & Wellnitz'20]
$\Rightarrow$ Structural insights: either there are few matches or $P$ is *almost* periodic.

**Theorem [CKW'20]**

At least one of the following holds:

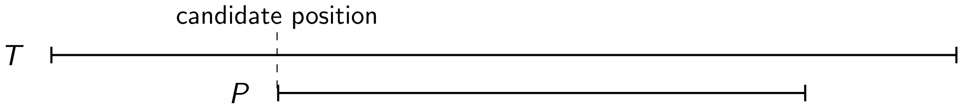
(i) The number of $k$-mismatch occurrences of $P$ in $T$ is $O(k)$.

(ii) There are $O(k)$ mismatches between $P$ and the periodic extension of a string $Q$.

Their constructive algorithm for the theorem can be adapted to a quantum algorithm requiring $\widetilde{O}(\sqrt{km})$ quantum time and query complexity.

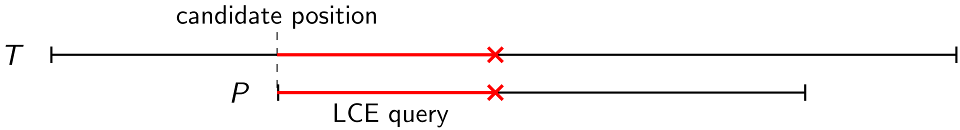
# Handling the two Cases

**Case (i):** $O(k)$ candidate positions for $k$-mismatch occurrences.



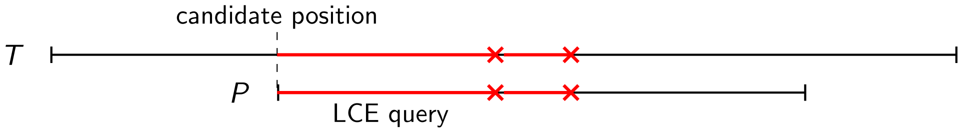
# Handling the two Cases

**Case (i):** $O(k)$ candidate positions for $k$-mismatch occurrences.



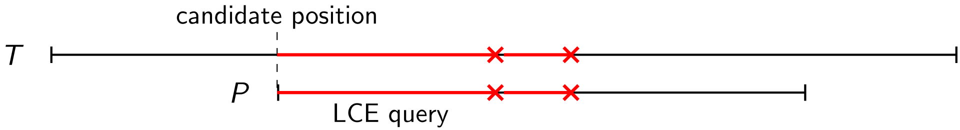
# Handling the two Cases

**Case (i):** $O(k)$ candidate positions for $k$-mismatch occurrences.



# Handling the two Cases

**Case (i):** $O(k)$ candidate positions for $k$-mismatch occurrences.

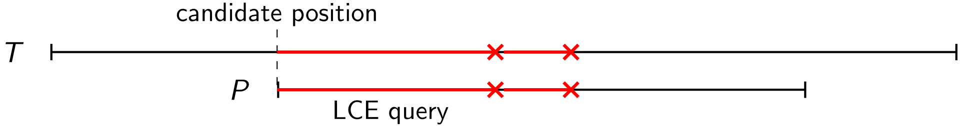



Speed up candidate position verification through LCE data structure:

$$\widetilde{O}\big(\underbrace{m/\tau^{\frac{1}{2}-o(1)}}_{\text{Preprocessing}} + \underbrace{\sqrt{k}\cdot k\sqrt{\tau}}_{\text{Grover's search } \times\ k \text{ LCE queries}}\big)$$

# Handling the two Cases

**Case (i):** $O(k)$ candidate positions for $k$-mismatch occurrences.

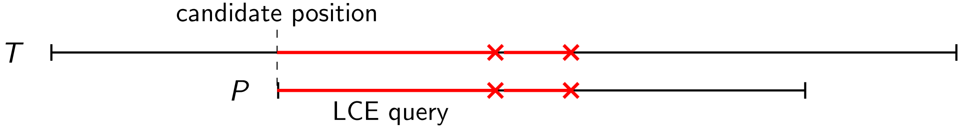



Speed up candidate position verification through LCE data structure:

$$\widetilde{O}\Big(\underbrace{m/\tau^{\frac{1}{2}-o(1)}}_{\text{Preprocessing}} + \underbrace{\sqrt{k}\cdot k\sqrt{\tau}}_{\text{Grover's search} \,\times\, k \text{ LCE queries}}\Big) \overset{(\tau=m/k^{3/2})}{=} \widetilde{O}(k^{3/4}m^{1/2+o(1)})$$

# Handling the two Cases

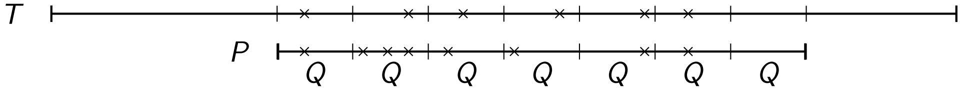
**Case (i):** $O(k)$ candidate positions for $k$-mismatch occurrences.




Speed up candidate position verification through LCE data structure:

$$\widetilde{O}\Big(\underbrace{m/\tau^{\frac{1}{2}-o(1)}}_{\text{Preprocessing}} + \underbrace{\sqrt{k}\cdot k\sqrt{\tau}}_{\text{Grover's search} \,\times\, k \text{ LCE queries}}\Big) \overset{(\tau=m/k^{3/2})}{=} \widetilde{O}(k^{3/4}m^{1/2+o(1)})$$

**Case (ii):** $P$ is *almost* periodic.



# Our Results for the $k$-mismatch Matching Problem

**Theorem [Jin & N'23]**

We can verify the existence of a $k$-mismatch occurrence of $P$ in $T$ (and report its starting position in case it exists) in $\widetilde{O}(k^{3/4}n^{1/2}m^{o(1)})$ query complexity and $\widetilde{O}(k\sqrt{n})$ time complexity.

# Our Results for the $k$-mismatch Matching Problem

**Theorem [Jin & N'23]**

We can verify the existence of a $k$-mismatch occurrence of $P$ in $T$ (and report its starting position in case it exists) in $\widetilde{O}(k^{3/4} n^{1/2} m^{o(1)})$ query complexity and $\widetilde{O}(k\sqrt{n})$ time complexity.

$\widetilde{O}(k\sqrt{n})$

# Open Problems

- Can we improve the extra $\tau^{o(1)}$ factors in the sparsity and the time complexity of our string synchronizing set to poly-logarithmic?

# Open Problems

- Can we improve the extra $\tau^{o(1)}$ factors in the sparsity and the time complexity of our string synchronizing set to poly-logarithmic?
- Can we improve the quantum query complexity of the $k$-mismatch matching algorithm to closer to the lower bound $\sqrt{kn}$?

# Open Problems

- Can we improve the extra $\tau^{o(1)}$ factors in the sparsity and the time complexity of our string synchronizing set to poly-logarithmic?
- Can we improve the quantum query complexity of the $k$-mismatch matching algorithm to closer to the lower bound $\sqrt{kn}$?
- Can our new result for string synchronizing sets find more applications in quantum string algorithms?

# Open Problems

- Can we improve the extra $\tau^{o(1)}$ factors in the sparsity and the time complexity of our string synchronizing set to poly-logarithmic?
- Can we improve the quantum query complexity of the $k$-mismatch matching algorithm to closer to the lower bound $\sqrt{kn}$?
- Can our new result for string synchronizing sets find more applications in quantum string algorithms?

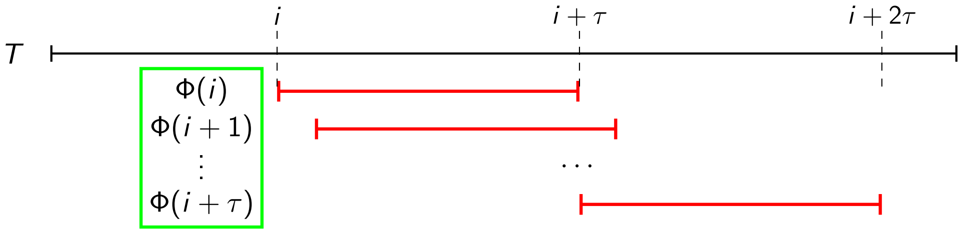
Thank you!

# Construction of $\tau$-synchronizing set A

We focus on 'non-periodic case': $\mathrm{per}(T[i \,..\, i+\tau]) > \tau/3$ for all $i$

Follow [KK'19]'s framework of 'picking local minimizers' of a hash function $\phi$:

- Choose $\phi : \Sigma^\tau \to \mathbb{Z}$
- Denote $\Phi(i) = \phi(T[i \,..\, i+\tau-1])$




$i \in$ A iff minimum is achieved at $\Phi(i)$ or $\Phi(i+\tau)$

Consistency and Density always hold.

# Construction of $\tau$-synchronizing set A

How to ensure **sparsity**?

- The hash function $\phi$ should guarantee probability $O(1/\tau)$ of $i$ being included in A.

How to make $\phi$ **efficiently computable** using few quantum queries?

- We need only ability to distinguish substrings with $\Omega(\tau)$ overlap: an adaptation of Deterministic Sampling [Vishkin'91] gives us this guarantee.
- To further speed up computability we structure $\phi$ such that one can find the minimal hash value in a tournament tree-like fashion.

By using quantum minimum finding for each level, we obtain the recursion

$$T(\tau) = \sqrt{b} \cdot (T(\tau/b) + \widetilde{O}(\sqrt{\tau})) \cdot O(\log \tau)$$
$$= \tau^{1/2+o(1)}$$