# Quantum divide and conquer

Daochen Wang (University of Maryland)

IQC Waterloo, full version: arXiv:2210.06419




Andrew M. Childs
University of Maryland

Robin Kothari
Microsoft
(→Google)

Matt Kovacs-Deak
University of Maryland

Aarthi Sundaram
Microsoft

# Examples of quantum speedups

## Some problems admit exponential quantum speedup

Factoring, discrete logarithm, solving Pell's equation, quantum simulation, EXIT-finding in glued-trees graph, graph connectivity using cut queries, Forrelation, Yamakawa-Zhandry problem…

## Others admit polynomial quantum speedup

Unstructured search, minimum finding, dynamic programming, graph properties using adjacency matrix queries, Monte Carlo mean estimation, element distinctness, formula evaluation…

## Can we find more problems with quantum speedup?

# Tools for designing quantum algorithms

- Fourier sampling
- Grover search/amplitude amplification
- Quantum walk
- Span programs
- Adiabatic optimization/QAOA
- Quantum signal processing/QSVT
- …

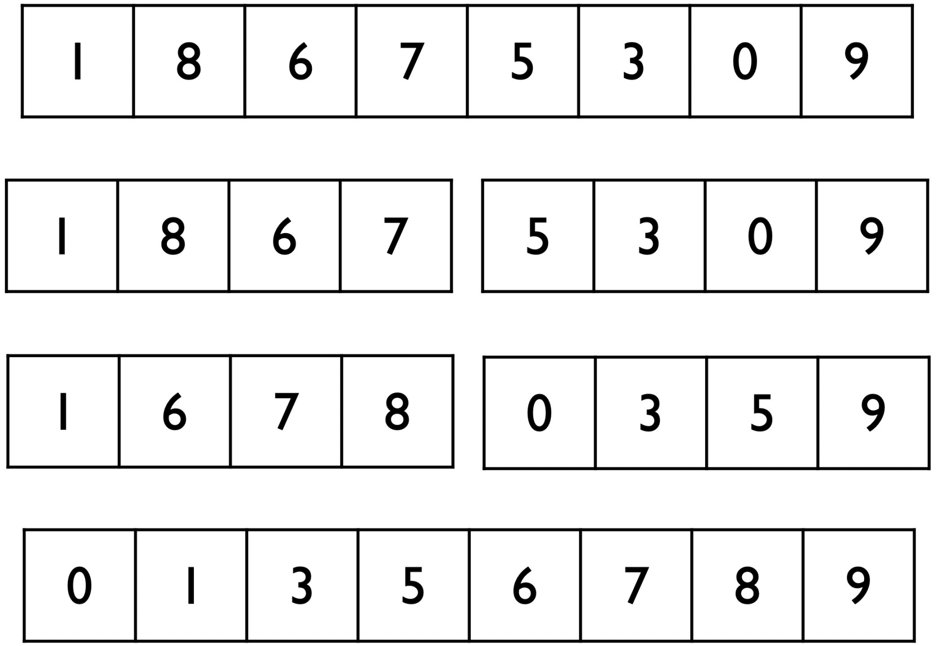
# Divide and conquer

1. Divide a problem into subproblems
2. Recursively solve each subproblem
3. Combine the solutions of the subproblems to solve full problem

**Example:** Mergesort

**Recurrence:**

$$C(n) = 2C(n/2) + O(n)$$
$$\downarrow$$
$$C(n) = O(n \log n)$$



# From classical to quantum divide and conquer

Simple example: $\mathrm{OR}(x) = x_1 \vee x_2 \vee \cdots \vee x_n$

Divide and conquer: $\mathrm{OR}(x) = \mathrm{OR}(\mathrm{OR}(x_{\mathrm{left}}), \mathrm{OR}(x_{\mathrm{right}}))$

Classical: $C(n) \leq 2C(n/2) \rightarrow C(n) \leq n$

Quantum: $C(n) \leq \sqrt{2}C(n/2) \rightarrow C(n) \leq \sqrt{n}$ ?

# From classical to quantum divide and conquer

Divide a problem of size $n$ into $a$ instances of size $n/b$ each

- Typical divide-and-conquer recurrence:

$$C(n) \leq a\,C(n/b) + C^{\mathrm{aux}}(n)$$

Classical cost of solving auxiliary problem

# Query complexity

The *query model* is a useful model in which we can provably compare the power of classical and quantum computers

Let $f: \Sigma^n \to \{0,1\}$ and suppose an algorithm computes $f(x)$ correctly with probability $\geq 2/3$ for all $x$

How many queries to the input $x$ does the algorithm need to make? Answer denoted $R(f)$ and $Q(f)$, when the algorithm is classical and quantum, respectively

Example: OR: $\{0,1\}^{\mathrm{n}} \to \{0,1\}$ $R(f) = \Theta(n)$ and $Q(f) = \Theta(\sqrt{n})$

Classical query

$i \mapsto x_i$

Quantum query

$|i\rangle|a\rangle \mapsto |i\rangle|a + x_i\rangle$

# The adversary quantity

Every function $f\colon \Sigma^n \to \{0,1\}$ can be associated with its adversary quantity $\mathrm{Adv}(f)$ which is a non-negative real number

**Theorem** [Høyer, Lee, Špalek 07; Lee, Mittal, Reichardt, Špalek 10]

$$Q(f) = \Theta(\mathrm{Adv}(f))$$

**Composition theorems**

- OR: let $g(x,y) = f_1(x) \vee f_2(y)$, then $\mathrm{Adv}(g)^2 \leq \mathrm{Adv}(f_1)^2 + \mathrm{Adv}(f_2)^2$
- SWITCH-CASE: let $h(x) = g_{f(x)}(x)$, then $\mathrm{Adv}(h) \leq O\big(\mathrm{Adv}(f)\big) + \max_s \mathrm{Adv}(g_s)$

# Quantum divide and conquer framework

Suppose $f$ is computed as an AND-OR formula of $f_1, \dots, f_a$ and $f^{\mathrm{aux}}$,

then $\mathrm{Adv}(f)^2 \leq \sum_{i=1}^{a} \mathrm{Adv}(f_i)^2 + O\big(Q(f^{\mathrm{aux}})\big)^2$

Suppose $f$ is computed by first computing $s = f^{\mathrm{aux}}(x)$ and then some function $g_s$,

then $\mathrm{Adv}(f) \leq O\big(Q(f^{\mathrm{aux}})\big) + \max_s \mathrm{Adv}(g_s)$

These strategies combine the adversary method (for the term where the constant matters) with the world of quantum algorithms (which are easier to design)

Other strategies are possible using other quantum adversary primitives

# Applications

Quantum query complexity

Simpler analysis with slightly improved upper bounds:

- **Regular languages**: Deciding whether a string over $\{0,1,2\}$ contains $20^*2$. This is a key algorithmic result in the query complexity trichotomy for regular languages [Aaronson, Grier, Schaeffer 19] — $O(\sqrt{n \log n})$

- **String minimality problems**: Decision versions of Minimal String Rotation and Minimal Suffix. Simpler, tighter analysis than [Akmal, Jin 22] — $O\left(\sqrt{n \log^5 n}\right)$

# Regular languages

Let $\Sigma = \{0,1,2\}$, $f_n : \Sigma^n \to \{0,1\}$ such that $f_n(x) = 1$ iff $x \in \Sigma^* 2 0^* 2 \Sigma^*$

02002110 ✅ 02102112 ⛔

Observation: let $g_n(x) = (x_{\text{left}} \in \Sigma^* 2 0^*) \wedge (x_{\text{right}} \in 0^* 2 \Sigma^*)$, then

$$f_n(x) = f_{n/2}(x_{\text{left}}) \vee f_{n/2}(x_{\text{right}}) \vee g_n(x)$$

Let $a(n) = \mathrm{Adv}(f_n)$, then $a(n)^2 \le 2a^2(n/2) + O\big(Q(g_n)\big)^2$

But $Q(g_n) = O(\sqrt{n})$, so $a(n) = O\left(\sqrt{n \log n}\right)$

# $k$-Increasing Subsequence

A **subsequence** of a string is obtained by taking a (not necessarily consecutive) subset of characters, without changing their order

**$k$-Increasing Subsequence ($k$-IS)**: given $x \in \Sigma^n$, $\Sigma$ ordered, does $x$ contain a strictly increasing subsequence of length $k$?

$$56122941$$

$k \leq 3$ ✅

$k > 3$ ⛔

$R(k\text{-IS}) = \Theta(n)$ for $k \geq 2$

$Q(2\text{-IS}) = \Theta(\sqrt{n})$ → equivalent to unstructured search ($x_1 \geq x_2 \geq \cdots \geq x_n$)

$Q(k\text{-IS}) = O\left(n^{k/(k+1)}\right)$ using [Ambainis 03]

Can we do better?

# $k$-Increasing Subsequence

**Theorem.** For any fixed $k$, $Q(k\text{-IS}) = O\left(\sqrt{n}\log^{3(k-1)/2} n\right)$

Let $\Sigma$ be an ordered set, $x \in \Sigma^n$ contains a $k$-IS iff

- $x_{\text{left}}$ contains a $k$-IS or
- $x_{\text{right}}$ contains a $k$-IS or
- $x$ contains a $k$-IS with $1 < j < k$ elements in $x_{\text{left}}$ and $k - j$ elements in $x_{\text{right}}$

**Composite** $k$-IS: Can be detected with $O(\log n)$ computations of $j$-IS, $(k - j)$-IS, and Grover search

Let $a_k(n)$ = adversary quantity for $k$-IS with input length $n$, then

$$a_k(n)^2 \leq 2a_k(n/2)^2 + O\left(\sum_{j=1}^{k-1}\left(\left(a_j(n) + \sqrt{n}\right)\log n\right)^2\right)$$

Result follows by induction on $k$

# Detecting composite $k$-IS

$x$ contains a $k$-IS with $1 < j < k$ elements in $x_{\text{left}}$ and $k - j$ elements in $x_{\text{right}}$
Can be detected with $O(\log n)$ computations of $j$-IS, $(k-j)$-IS, and Grover search

$x =$ _ _ _ _ _ _ _ _ $\overset{\alpha}{\_}$ _ _ _ _ | _ _ $\overset{\beta}{\_}$ _ _ _ _ _ _ _ _ _ _ _

Find the **smallest** element of $\Sigma$ that is at the **end** of a $j$-IS in $x_{\text{left}}$, $\alpha$

Find the **largest** element of $\Sigma$ that is at the **start** of a $(k-j)$-IS in $x_{\text{right}}$, $\beta$

If $\alpha < \beta$, then output 1

- Naively $O\left(\log(|\Sigma|)\left(a_j(n/2) + a_{k-j}(n/2)\right)\right)$
- Can do $O\left(\left(a_j(n/2) + a_{k-j}(n/2) + \sqrt{n}\right)\log n\right)$ using *randomized search*: pick a uniformly random position $a \in [n/2]$, use $j$-IS algorithm to compare $x_a$ *and* $\alpha$, if $\alpha \leq x_a$ (say), then pick a uniformly random position from $S = \{b \in [n/2] \mid x_b \leq x_a\}$ using Grover search, and repeat

# $k$-Common Subsequence

**$k$-Common Subsequence ($k$-CS)**: given $x, y \in \Sigma^n$ do $x$ and $y$ share a subsequence of length $k$?

| | | |
|---|---|---|
| E i n s t e i n | $k \leq 4$ | ✅ |
| e n t w i n e d | $k > 4$ | ⛔ |

$R(k\text{-CS}) = \Theta(n)$ for $k \geq 1$

$Q(1\text{-CS}) = \Theta(n^{2/3}) \rightarrow$ bipartite element distinctness [Aaronson, Shi 04; Ambainis 03]

$Q(k\text{-CS}) = O\left(n^{2k/(2k+1)}\right)$ using [Ambainis 03]

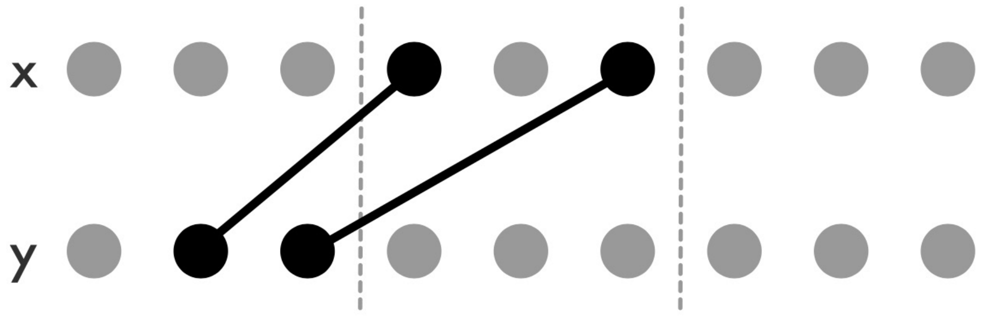
Can we do better?

# $k$-Common Subsequence

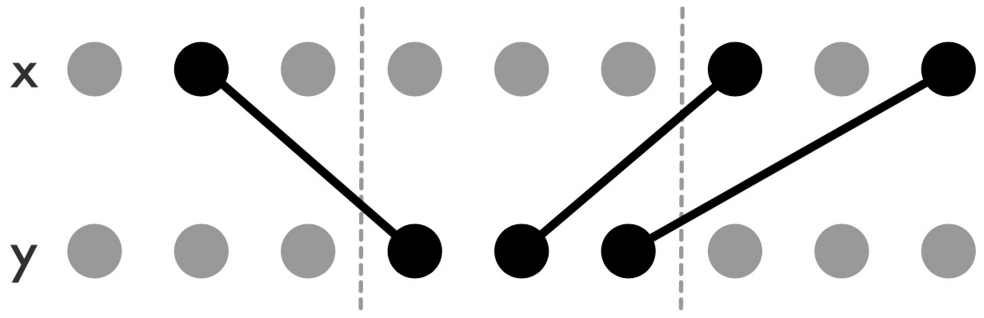
Divide the two input strings $x$ and $y$ into $m$ parts each. Then, a $k$-CS can either be **simple** or **composite**

- A simple $k$-CS is a $k$-CS formed by symbols within a *single* part of $x$ and a *single* part of $y$
- A composite $k$-CS is any $k$-CS that is not simple

Examples with $m = 3$




Simple

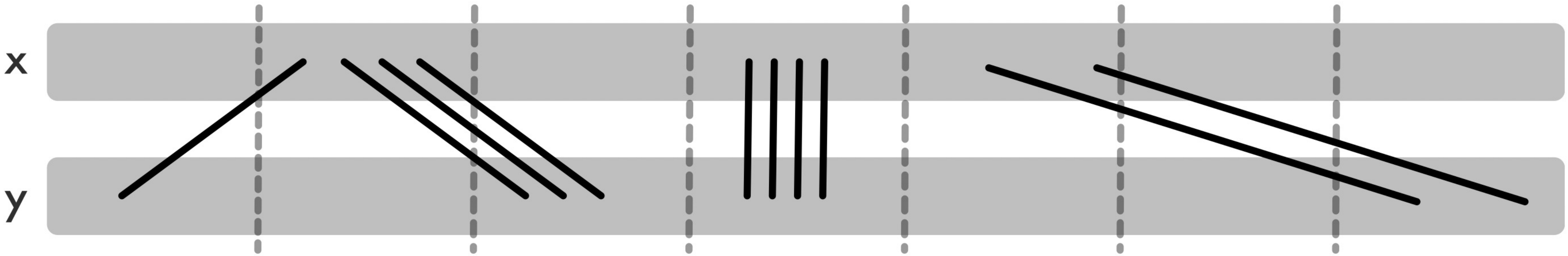



Composite

# Detecting composite $k$-CS

Let $a_k(n)$ = adversary quantity for $k$-CS with input length $n$

$m = 7, k = 10$

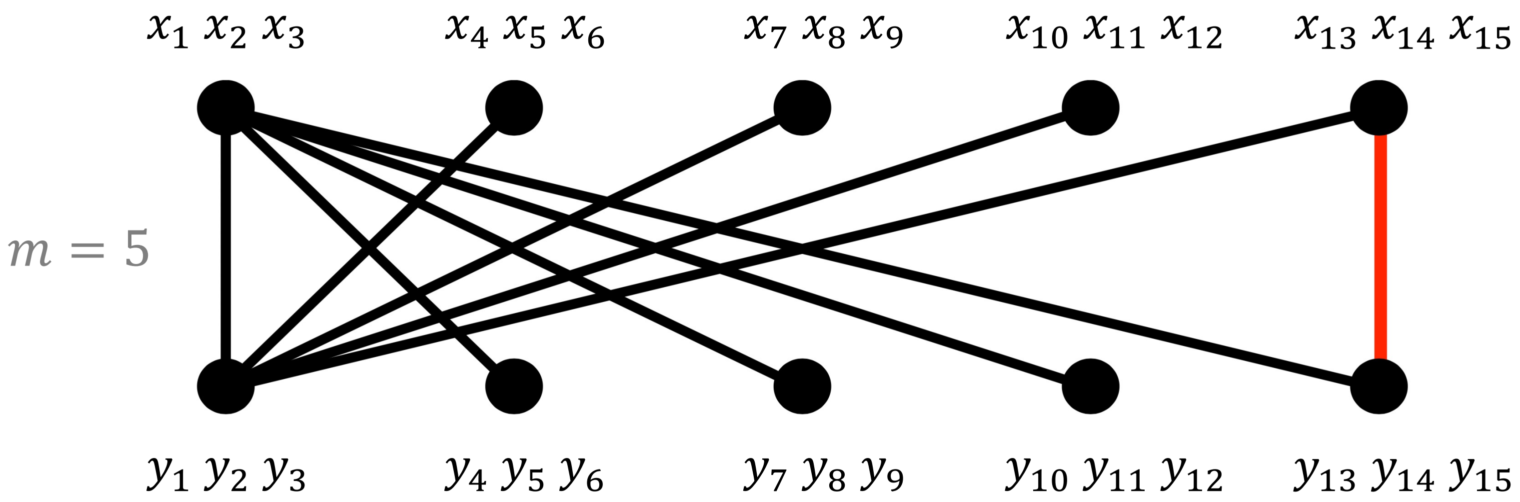



Only a constant number of possible configurations (depends on constants $m$ and $k$)

$$O\left(\sum_{j=1}^{k-1} a_j(n) \log n\right)$$

# Detecting simple $k$-CS

1. Compute $A \in \{0,1\}^{m\times m}$ such that $A_{ij} = 1$ iff the $i$th part of $x$ and the $j$th part of $y$ have a common symbol: $O\left(m^2 n^{2/3}\right) = O(n^{2/3})$




2. Need to compute OR of *at most* $2m-1$ copies of $k$-CS of size $n/m$: $\sqrt{2m-1}\, a_k(n/m)$

Overall: $O\left(n^{2/3}\right) + \sqrt{2m-1}\, a_k(n/m)$

Recall: suppose $f$ is computed by first computing $s = f^{\text{aux}}(x)$ and then some function $g_s$, then
$\text{Adv}(f) \le O\left(Q(f^{\text{aux}})\right) + \max_s \text{Adv}(g_s)$

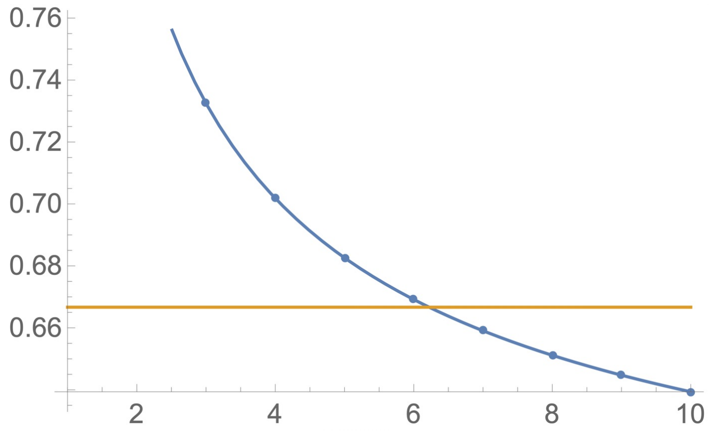
# Putting it together

**Claim.** $a_k(n) = O(n^{2/3} \log^{k-1} n)$

- Detecting composite $k$-CS: $O\left(\sum_{j=1}^{k-1} a_j(n) \log n\right)$
- Detecting simple $k$-CS: $O\left(n^{2/3}\right) + \sqrt{2m-1}\, a_k(n/m)$

$\Downarrow$ by induction on $k$

$$a_k(n) \le \sqrt{2m-1}\, a_k(n/m) + O\left(n^{2/3} \log^{k-1} n\right)$$




Master Theorem: $a_k(n) = O\left(n^{2/3} \log^{k-1} n\right)$
provided $\log_m\left(\sqrt{2m-1}\right) < 2/3$, which is satisfied with $m = 7$

# Summary

We have introduced a divide-and-conquer framework for developing quantum algorithms using classical reasoning about division into subproblems, with speedup from quantum combining operations and the use of quantum subroutines. Applications:

- Simpler analysis for regular languages and minimal substring problems with tighter bounds
- $\tilde{O}(\sqrt{n})$ algorithm for $k$-IS
- $\tilde{O}\left(n^{2/3}\right)$ algorithm for $k$-CS

# Open problems

- Can we apply quantum divide and conquer to search problems? For example, is there a quantum divide-and-conquer algorithm for minimum finding?
- Can we find applications of quantum divide and conquer using combining functions other than AND-OR formulas and SWITCH-CASE?
- Can we obtain super-quadratic speedups using quantum divide and conquer?

# Appendix: adversary quantity definition

Let $f\colon \Sigma^n \to \{0,1\}$. Then

$$\mathrm{Adv}(f) = \max_{\Gamma} \frac{\|\Gamma\|}{\max_{i\in\{1,\dots,n\}} \|\Gamma\|},$$

where the max is taken over $|\Sigma|^n \times |\Sigma|^n$ real symmetric matrices $\Gamma$ with $f(x) = f(y) \implies \Gamma_{xy} = 0$ and

$$(\Gamma_i)_{xy} = \begin{cases} \Gamma_{xy} & \text{if } x_i \neq y_i, \\ 0 & \text{if } x_i = y_i. \end{cases}$$