Optimal quantum adversary lower bounds for ordered search

Andrew Childs
Waterloo

Troy Lee
Rutgers

arXiv:0708.3396

Quantum speedup

Quantum speedup

		Classical	Quantum
Exponential:	Simon's problem	$\Omega(2^{n/2})$	$\Theta(n)$

Quantum speedup

		Classical	Quantum
Exponential:	Simon's problem	$\Omega(2^{n/2})$	$\Theta(n)$
Super-polynomial:	Factoring	$2^{O(n^{1/3}(\log n)^{2/3})}$	$O(n^3)$

Quantum speedup

		Classical	Quantum
Exponential:	Simon's problem	$\Omega(2^{n/2})$	$\Theta(n)$
Super-polynomial:	Factoring	$2^{O(n^{1/3}(\log n)^{2/3})}$	$O(n^3)$
	Dihedral HSP	$\Omega(2^{n/2})$	$2^{O(\sqrt{\log N})}$

Quantum speedup

		Classical	Quantum
Exponential:	Simon's problem	$\Omega(2^{n/2})$	$\Theta(n)$
Super-polynomial:	Factoring	$2^{O(n^{1/3}(\log n)^{2/3})}$	$O(n^3)$
	Dihedral HSP	$\Omega(2^{n/2})$	$2^{O(\sqrt{\log N})}$
Polynomial:	Unstructured search	$\Theta(n)$	$\Theta(\sqrt{n})$

Quantum speedup

		Classical	Quantum
Exponential:	Simon's problem	$\Omega(2^{n/2})$	$\Theta(n)$
Super-polynomial:	Factoring	$2^{O(n^{1/3}(\log n)^{2/3})}$	$O(n^3)$
	Dihedral HSP	$\Omega(2^{n/2})$	$2^{O(\sqrt{\log N})}$
Polynomial:	Unstructured search	$\Theta(n)$	$\Theta(\sqrt{n})$
	Element distinctness	$\Theta(n)$	$\Theta(n^{2/3})$

Quantum speedup

		Classical	Quantum
Exponential:	Simon's problem	$\Omega(2^{n/2})$	$\Theta(n)$
Super-polynomial:	Factoring	$2^{O(n^{1/3}(\log n)^{2/3})}$	$O(n^3)$
	Dihedral HSP	$\Omega(2^{n/2})$	$2^{O(\sqrt{\log N})}$
Polynomial:	Unstructured search	$\Theta(n)$	$\Theta(\sqrt{n})$
	Element distinctness	$\Theta(n)$	$\Theta(n^{2/3})$
Constant:	Parity	n	$n/2$

Query complexity

Problem: Compute a function $f : S \to \Sigma$

Input set: $S \subseteq \{0,1\}^n$ Output set: Σ

Fix some (unknown) input $x \in S$. Given a black box for the bits of x, how many queries are required to compute $f(x)$?

Query complexity

Problem: Compute a function $f : S \to \Sigma$

Input set: $S \subseteq \{0,1\}^n$ Output set: Σ

Fix some (unknown) input $x \in S$. Given a black box for the bits of x, how many queries are required to compute $f(x)$?

Example: Unstructured search (aka OR)

Query complexity

Problem: Compute a function $f : S \to \Sigma$

Input set: $S \subseteq \{0,1\}^n$ Output set: Σ

Fix some (unknown) input $x \in S$. Given a black box for the bits of x, how many queries are required to compute $f(x)$?

Example: Unstructured search (aka OR)

$$S = \{0,1\}^n \qquad \Sigma = \{0,1\}$$

$$f(x) = \begin{cases} 0 & x = 00\ldots0 \\ 1 & \text{otherwise} \end{cases}$$

Query complexity

Problem: Compute a function $f : S \to \Sigma$

Input set: $S \subseteq \{0,1\}^n$ Output set: Σ

Fix some (unknown) input $x \in S$. Given a black box for the bits of x, how many queries are required to compute $f(x)$?

Example: Unstructured search (aka OR)

$$S = \{0,1\}^n \qquad \Sigma = \{0,1\}$$

$$f(x) = \begin{cases} 0 & x = 00\ldots0 \\ 1 & \text{otherwise} \end{cases}$$

(Deterministic) query complexity: n

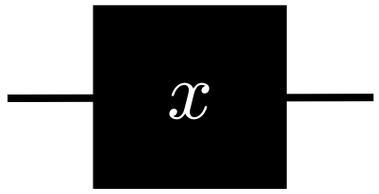
Quantum query complexity

Recall that the black box contains a string $x \in \{0,1\}^n$.

Quantum query complexity

Recall that the black box contains a string $x \in \{0,1\}^n$.

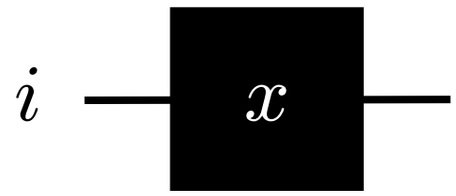
Classical query:



Quantum query complexity

Recall that the black box contains a string $x \in \{0,1\}^n$.

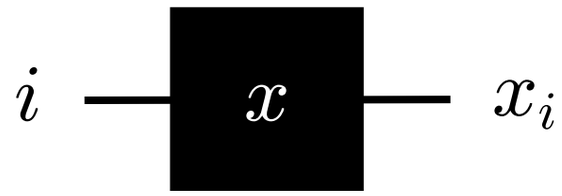
Classical query:



Quantum query complexity

Recall that the black box contains a string $x \in \{0,1\}^n$.

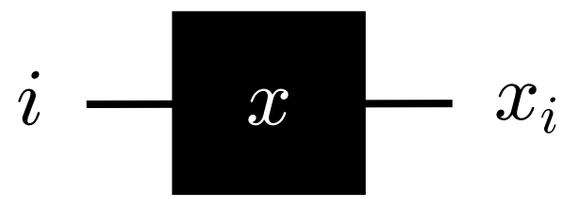
Classical query:



Quantum query complexity

Recall that the black box contains a string $x \in \{0,1\}^n$.

Classical query:

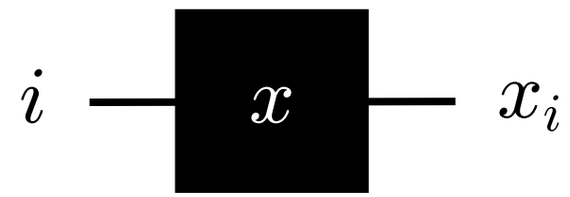


In a quantum computer, the state space is a *vector space* (instead of a finite set). For every possible state i of a classical computer, the corresponding quantum computer has a basis vector $\vec{e}_i$. Any linear combination of the basis vectors gives an allowed state.

Quantum query complexity

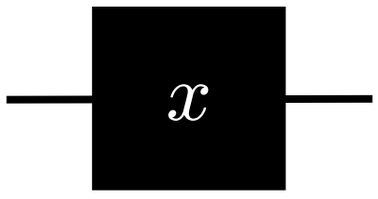
Recall that the black box contains a string $x \in \{0,1\}^n$.

Classical query:



In a quantum computer, the state space is a *vector space* (instead of a finite set). For every possible state i of a classical computer, the corresponding quantum computer has a basis vector $\vec{e}_i$. Any linear combination of the basis vectors gives an allowed state.

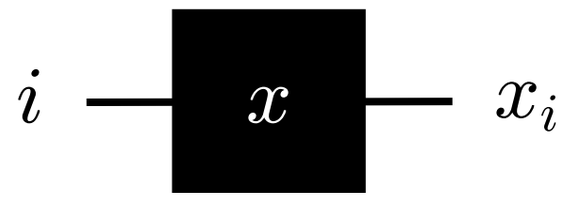
Quantum query:



Quantum query complexity

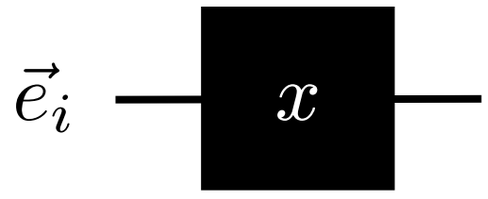
Recall that the black box contains a string $x \in \{0,1\}^n$.

Classical query:



In a quantum computer, the state space is a *vector space* (instead of a finite set). For every possible state i of a classical computer, the corresponding quantum computer has a basis vector $\vec{e}_i$. Any linear combination of the basis vectors gives an allowed state.

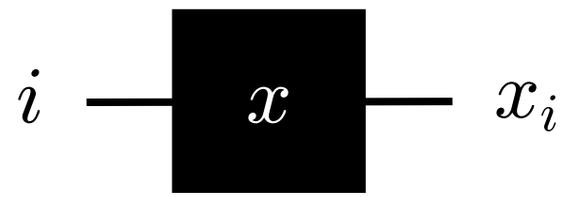
Quantum query:



Quantum query complexity

Recall that the black box contains a string $x \in \{0,1\}^n$.

Classical query:



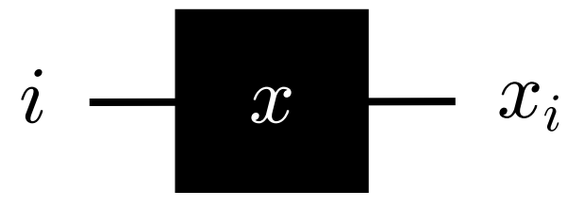
In a quantum computer, the state space is a *vector space* (instead of a finite set). For every possible state i of a classical computer, the corresponding quantum computer has a basis vector $\vec{e}_i$. Any linear combination of the basis vectors gives an allowed state.

Quantum query:

Quantum query complexity

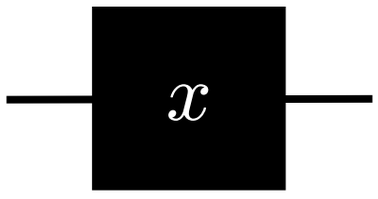
Recall that the black box contains a string $x \in \{0,1\}^n$.

Classical query:



In a quantum computer, the state space is a *vector space* (instead of a finite set). For every possible state i of a classical computer, the corresponding quantum computer has a basis vector $\vec{e}_i$. Any linear combination of the basis vectors gives an allowed state.

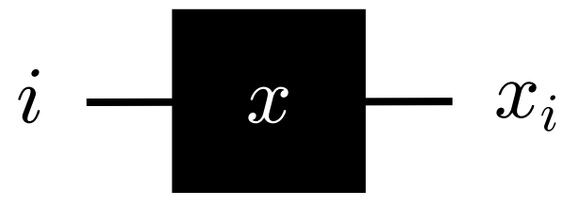
Quantum query:



Quantum query complexity

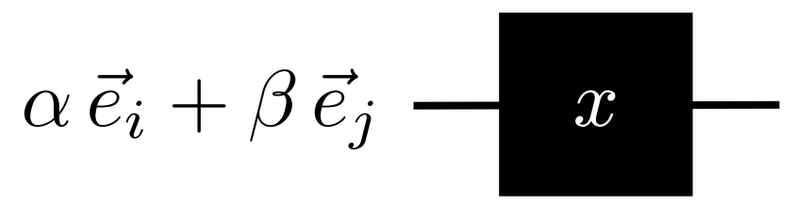
Recall that the black box contains a string $x \in \{0,1\}^n$.

Classical query:



In a quantum computer, the state space is a *vector space* (instead of a finite set). For every possible state i of a classical computer, the corresponding quantum computer has a basis vector $\vec{e}_i$. Any linear combination of the basis vectors gives an allowed state.

Quantum query:

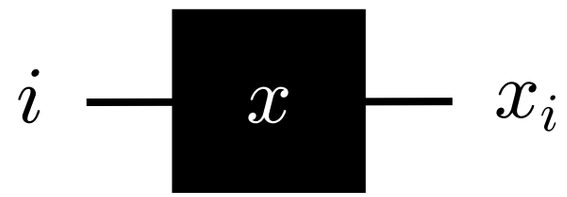


$\alpha, \beta \in \mathbb{C}$

Quantum query complexity

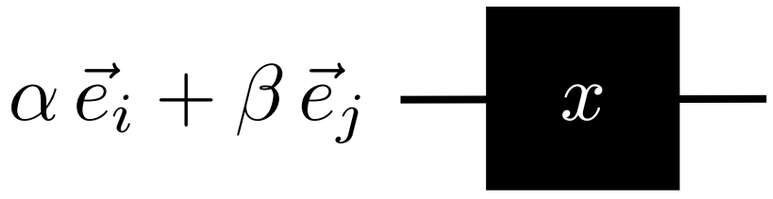
Recall that the black box contains a string $x \in \{0,1\}^n$.

Classical query:



In a quantum computer, the state space is a *vector space* (instead of a finite set). For every possible state i of a classical computer, the corresponding quantum computer has a basis vector $\vec{e}_i$. Any linear combination of the basis vectors gives an allowed state.

Quantum query:

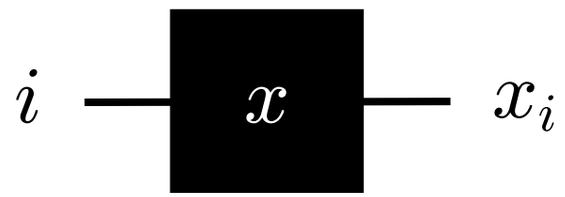


$\alpha, \beta \in \mathbb{C}$

Quantum query complexity

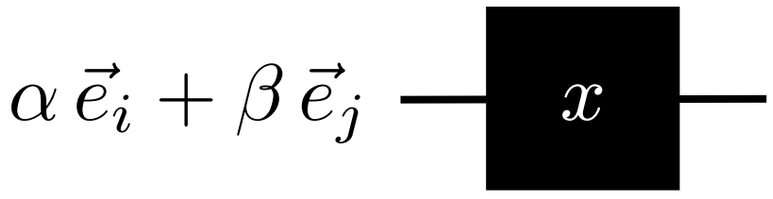
Recall that the black box contains a string $x \in \{0,1\}^n$.

Classical query:



In a quantum computer, the state space is a *vector space* (instead of a finite set). For every possible state i of a classical computer, the corresponding quantum computer has a basis vector $\vec{e}_i$. Any linear combination of the basis vectors gives an allowed state.

Quantum query:



$\alpha, \beta \in \mathbb{C}$

Quantum query complexity: Minimum number of quantum queries required to quantum compute $f(x)$, given a black box for the input x.

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

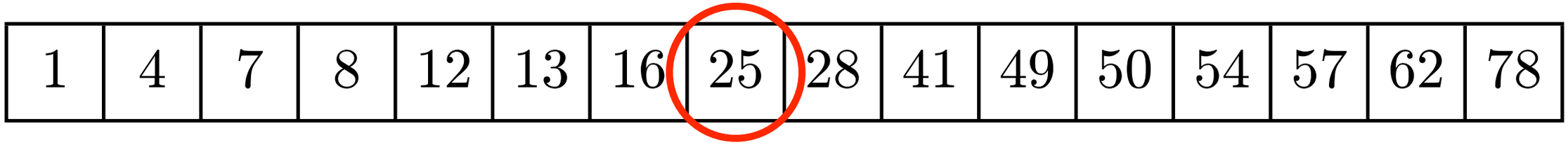
Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list



Ordered search

Given a *sorted* list of n items, find the position of a desired item.

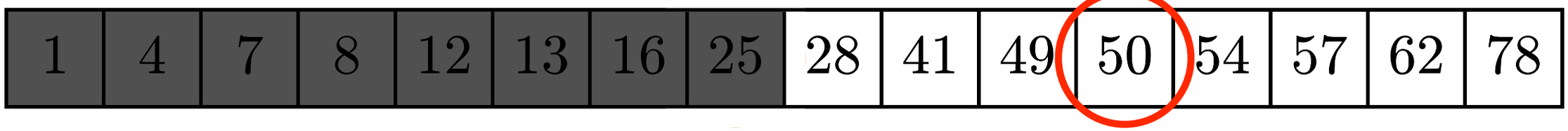
Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list



Ordered search

Given a *sorted* list of n items, find the position of a desired item.

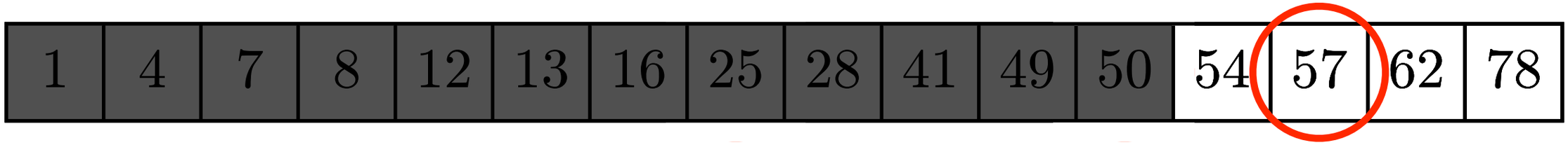
Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list



Ordered search

Given a *sorted* list of n items, find the position of a desired item.

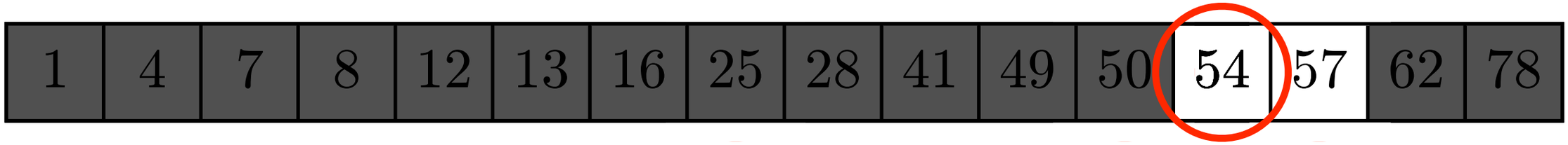
Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list



Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

This algorithm (binary search) uses about $\log_2 n$ queries.

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

This algorithm (binary search) uses about $\log_2 n$ queries.

This is optimal. (One bit per query.)

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

This algorithm (binary search) uses about $\log_2 n$ queries.

This is optimal. (One bit per query.)

Query complexity formulation: $f : S \to \Sigma$

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

This algorithm (binary search) uses about $\log_2 n$ queries.

This is optimal. (One bit per query.)

Query complexity formulation: $f : S \to \Sigma$

$S =$ strings of the form $\underbrace{0 \cdots 0}_{k}\underbrace{1 \cdots 1}_{n-k}$ with $k = 0, \ldots, n-1$

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

This algorithm (binary search) uses about $\log_2 n$ queries.

This is optimal. (One bit per query.)

Query complexity formulation: $f : S \to \Sigma$

$S =$ strings of the form $\underbrace{0 \cdots 0}_{k}\underbrace{1 \cdots 1}_{n-k}$ with $k = 0, \ldots, n-1$

$\Sigma = S$, and $f(x) = x$ (i.e., this is an *oracle identification problem*)

Ordered search

Given a *sorted* list of n items, find the position of a desired item.

Example: Search for 54 in the list

1	4	7	8	12	13	16	25	28	41	49	50	54	57	62	78

This algorithm (binary search) uses about $\log_2 n$ queries.

This is optimal. (One bit per query.)

Query complexity formulation: $f : S \to \Sigma$

$S =$ strings of the form $\underbrace{0 \cdots 0}_{k}\underbrace{1 \cdots 1}_{n-k}$ with $k = 0, \ldots, n-1$

$\Sigma = S$, and $f(x) = x$ (i.e., this is an *oracle identification problem*)

In the above example, we have $x =$

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Quantum query complexity of ordered search

Quantum query complexity of ordered search

Upper bounds

Quantum query complexity of ordered search

Upper bounds

$$\log_3 n \approx 0.631 \log_2 n$$

Høyer, Neerbek, Shi 01

$$3 \log_{52} n \approx 0.526 \log_2 n$$

Farhi, Goldstone, Gutmann, Sipser 99

$$4 \log_{550} n \approx 0.439 \log_2 n$$

Brookes, Jacokes, Landahl 04

$$4 \log_{605} n \approx 0.433 \log_2 n$$

Childs, Landahl, Parrilo 06

Quantum query complexity of ordered search

Upper bounds

$\log_3 n \approx 0.631 \log_2 n$

Høyer, Neerbek, Shi 01

$3 \log_{52} n \approx 0.526 \log_2 n$

Farhi, Goldstone, Gutmann, Sipser 99

$4 \log_{550} n \approx 0.439 \log_2 n$

Brookes, Jacokes, Landahl 04

$4 \log_{605} n \approx 0.433 \log_2 n$

Childs, Landahl, Parrilo 06

$\approx 0.32 \log_2 n$ (bounded-error)

Ben-Or, Hassidim 07

Quantum query complexity of ordered search

Upper bounds

$\log_3 n \approx 0.631 \log_2 n$

Høyer, Neerbek, Shi 01

$3 \log_{52} n \approx 0.526 \log_2 n$

Farhi, Goldstone, Gutmann, Sipser 99

$4 \log_{550} n \approx 0.439 \log_2 n$

Brookes, Jacokes, Landahl 04

$4 \log_{605} n \approx 0.433 \log_2 n$

Childs, Landahl, Parrilo 06

$\approx 0.32 \log_2 n$ (bounded-error)

Ben-Or, Hassidim 07

Lower bounds

Quantum query complexity of ordered search

Upper bounds

$\log_3 n \approx 0.631 \log_2 n$

Høyer, Neerbek, Shi 01

$3 \log_{52} n \approx 0.526 \log_2 n$

Farhi, Goldstone, Gutmann, Sipser 99

$4 \log_{550} n \approx 0.439 \log_2 n$

Brookes, Jacokes, Landahl 04

$4 \log_{605} n \approx 0.433 \log_2 n$

Childs, Landahl, Parrilo 06

$\approx 0.32 \log_2 n$ (bounded-error)

Ben-Or, Hassidim 07

Lower bounds

$\Omega(\frac{\sqrt{\log N}}{\log \log N})$

Buhrman, de Wolf 98

$\Omega(\frac{\log N}{\log \log N})$

Farhi, Goldstone, Gutmann, Sipser 98

Quantum query complexity of ordered search

Upper bounds

$\log_3 n \approx 0.631 \log_2 n$

Høyer, Neerbek, Shi 01

$3 \log_{52} n \approx 0.526 \log_2 n$

Farhi, Goldstone, Gutmann, Sipser 99

$4 \log_{550} n \approx 0.439 \log_2 n$

Brookes, Jacokes, Landahl 04

$4 \log_{605} n \approx 0.433 \log_2 n$

Childs, Landahl, Parrilo 06

$\approx 0.32 \log_2 n$ (bounded-error)

Ben-Or, Hassidim 07

Lower bounds

$\Omega(\frac{\sqrt{\log N}}{\log \log N})$

Buhrman, de Wolf 98

$\Omega(\frac{\log N}{\log \log N})$

Farhi, Goldstone, Gutmann, Sipser 98

$\frac{1}{12} \log_2 n \approx 0.0833 \log_2 n$

Ambainis 99

Quantum query complexity of ordered search

Upper bounds

$\log_3 n \approx 0.631 \log_2 n$

Høyer, Neerbek, Shi 01

$3 \log_{52} n \approx 0.526 \log_2 n$

Farhi, Goldstone, Gutmann, Sipser 99

$4 \log_{550} n \approx 0.439 \log_2 n$

Brookes, Jacokes, Landahl 04

$4 \log_{605} n \approx 0.433 \log_2 n$

Childs, Landahl, Parrilo 06

$\approx 0.32 \log_2 n$ (bounded-error)

Ben-Or, Hassidim 07

Lower bounds

$\Omega(\frac{\sqrt{\log N}}{\log \log N})$

Buhrman, de Wolf 98

$\Omega(\frac{\log N}{\log \log N})$

Farhi, Goldstone, Gutmann, Sipser 98

$\frac{1}{12} \log_2 n \approx 0.0833 \log_2 n$

Ambainis 99

$\frac{1}{\pi} \ln n \approx 0.221 \log_2 n$

Høyer, Neerbek, Shi 01

Quantum query complexity of ordered search

Upper bounds

$\log_3 n \approx 0.631 \log_2 n$

Høyer, Neerbek, Shi 01

$3 \log_{52} n \approx 0.526 \log_2 n$

Farhi, Goldstone, Gutmann, Sipser 99

$4 \log_{550} n \approx 0.439 \log_2 n$

Brookes, Jacokes, Landahl 04

$4 \log_{605} n \approx 0.433 \log_2 n$

Childs, Landahl, Parrilo 06

$\approx 0.32 \log_2 n$ (bounded-error)

Ben-Or, Hassidim 07

Lower bounds

$\Omega(\frac{\sqrt{\log N}}{\log \log N})$

Buhrman, de Wolf 98

$\Omega(\frac{\log N}{\log \log N})$

Farhi, Goldstone, Gutmann, Sipser 98

$\frac{1}{12} \log_2 n \approx 0.0833 \log_2 n$

Ambainis 99

$\frac{1}{\pi} \ln n \approx 0.221 \log_2 n$

Høyer, Neerbek, Shi 01

Quantum query complexity: $c \log_2 n$ for some c. What is c?

Quantum query complexity of ordered search

Upper bounds

$\log_3 n \approx 0.631 \log_2 n$

Høyer, Neerbek, Shi 01

$3 \log_{52} n \approx 0.526 \log_2 n$

Farhi, Gol 99

$4 \log_5$

$4 \log_{60}$

$\approx 0.32 \log_2 n$ (bounded-error)

Ben-Or, Hassidim 07

Lower bounds

$\Omega(\frac{\sqrt{\log N}}{\log \log N})$

Buhrman, de Wolf 98

$\Omega(\frac{\log N}{\log \log N})$

Farhi, Goldstone, Gutmann, Sipser 98

$\frac{1}{12} \log_2 n \approx 0.0833 \log_2 n$

Ambainis 99

$\frac{1}{\pi} \ln n \approx 0.221 \log_2 n$

Høyer, Neerbek, Shi 01

Theorem.
This is asymptotically optimal among all adversary lower bounds.

Quantum query complexity: $c \log_2 n$ for some c. What is c?

The quantum adversary method

$$\mathrm{ADV}(f) := \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$$

where Γ is an $|S| \times |S|$ matrix

entries $\Gamma[x, y]$ correspond to pairs of inputs $x, y \in S$

$\Gamma[x, y] = 0$ if $f(x) = f(y)$

$$\Gamma_i[x, y] := \begin{cases} 0 & x_i = y_i \\ \Gamma[x, y] & x_i \neq y_i \end{cases}$$

The quantum adversary method

$$\mathrm{ADV}(f) := \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$$

where Γ is an $|S| \times |S|$ matrix

entries $\Gamma[x, y]$ correspond to pairs of inputs $x, y \in S$

$\Gamma[x, y] = 0$ if $f(x) = f(y)$

$$\Gamma_i[x, y] := \begin{cases} 0 & x_i = y_i \\ \Gamma[x, y] & x_i \neq y_i \end{cases}$$

Theorem [Ambainis 00]: (Q. query complexity of f) $\geq \frac{1}{2}\mathrm{ADV}(f)$.

The quantum adversary method

$$\mathrm{ADV}(f) := \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$$

where Γ is an $|S| \times |S|$ matrix

entries $\Gamma[x, y]$ correspond to pairs of inputs $x, y \in S$

$\Gamma[x, y] = 0$ if $f(x) = f(y)$

$$\Gamma_i[x, y] := \begin{cases} 0 & x_i = y_i \\ \Gamma[x, y] & x_i \neq y_i \end{cases}$$

Theorem [Ambainis 00]: (Q. query complexity of f) $\geq \frac{1}{2}\mathrm{ADV}(f)$.

Proof idea: Define a progress measure for algorithms. It starts at 0 and must reach $\|\Gamma\|$ for the algorithm to succeed; the maximum change per query is $2 \max_i \|\Gamma_i\|$.

Semidefinite programming and $\mathrm{ADV}(f)$

In a *semidefinite program*, we optimize a linear objective function subject to matrix positivity constraints.

Semidefinite programming and $\mathrm{ADV}(f)$

In a *semidefinite program*, we optimize a linear objective function subject to matrix positivity constraints.

Two important features:

Semidefinite programming and $\mathrm{ADV}(f)$

In a *semidefinite program*, we optimize a linear objective function subject to matrix positivity constraints.

Two important features:

- There is good software to solve semidefinite programs numerically (using interior point methods).

Semidefinite programming and $\mathrm{ADV}(f)$

In a *semidefinite program*, we optimize a linear objective function subject to matrix positivity constraints.

Two important features:

- There is good software to solve semidefinite programs numerically (using interior point methods).
- From a primal SDP (say, a maximization problem), we can construct a dual SDP, which is a minimization problem.
  The maximum value of the primal SDP equals the minimum value of the dual SDP.
  A particular solution of the primal gives a lower bound; a particular solution of the dual gives an upper bound.

Semidefinite programming and $\mathrm{ADV}(f)$

In a *semidefinite program*, we optimize a linear objective function subject to matrix positivity constraints.

Two important features:

- There is good software to solve semidefinite programs numerically (using interior point methods).
- From a primal SDP (say, a maximization problem), we can construct a dual SDP, which is a minimization problem.
  The maximum value of the primal SDP equals the minimum value of the dual SDP.
  A particular solution of the primal gives a lower bound; a particular solution of the dual gives an upper bound.

Notice that computing $\mathrm{ADV}(f)$ is a semidefinite program!

Symmetry

Intuitively, symmetries of f should make it easier to deal with.

Symmetry

Intuitively, symmetries of f should make it easier to deal with.

Definition: An *automorphism* of $f : S \to \Sigma$ is a permutation $\pi \in S_n$ with

$$\pi(S) = S \quad \text{and} \quad f(x) = f(y) \Leftrightarrow f(\pi(x)) = f(\pi(y)) \; \forall \, x, y \in S.$$

Symmetry

Intuitively, symmetries of f should make it easier to deal with.

Definition: An *automorphism* of $f : S \to \Sigma$ is a permutation $\pi \in S_n$ with

$\pi(S) = S$ and $f(x) = f(y) \Leftrightarrow f(\pi(x)) = f(\pi(y)) \; \forall \, x, y \in S.$

Automorphism principle [Høyer, Lee, Špalek 07]: If π is an automorphism of f, then we can choose an optimal adversary matrix Γ satisfying $\Gamma[x, y] = \Gamma[\pi(x), \pi(y)]$ for all pairs of inputs x, y.

Furthermore, if the automorphism group is transitive, then the uniform vector is a principal eigenvector of Γ, and all $\|\Gamma_i\|$ are equal.

Symmetrizing ordered search

Recall ordered search function: e.g., for $n = 4$, the inputs are

$$S = \{1111, 0111, 0011, 0001\}$$

Symmetrizing ordered search

Recall ordered search function: e.g., for $n = 4$, the inputs are

$$S = \{1111, 0111, 0011, 0001\}$$

The automorphism group is trivial! No permutation but id fixes S.

Symmetrizing ordered search

Recall ordered search function: e.g., for $n = 4$, the inputs are

$$S = \{1111, 0111, 0011, 0001\}$$

The automorphism group is trivial! No permutation but id fixes S.

Extend to a circle of $2n$ bits: e.g., for $n = 4$,

$$S' = \{11110000, 01111000, 00111100, 00011110,$$
$$00001111, 10000111, 11000011, 11100001\}$$

Farhi, Goldstone, Gutmann, Sipser 99

Symmetrizing ordered search

Recall ordered search function: e.g., for $n = 4$, the inputs are

$$S = \{1111, 0111, 0011, 0001\}$$

The automorphism group is trivial! No permutation but id fixes S.

Extend to a circle of $2n$ bits: e.g., for $n = 4$,

$$S' = \{11110000, 01111000, 00111100, 00011110,$$
$$00001111, 10000111, 11000011, 11100001\}$$

Again, the problem is to identify the input ($f(x) = x$).
Now the automorphism group is cyclic ($2n$ elements).
We call this problem OSP_n.

Farhi, Goldstone, Gutmann, Sipser 99

Symmetric ordered search ≈ Ordered search

This problem looks similar, but maybe its query complexity is dramatically different!

Symmetric ordered search ≈ Ordered search

This problem looks similar, but maybe its query complexity is dramatically different!

In fact, the query complexity differs by at most 1.

Symmetric ordered search ≈ Ordered search

This problem looks similar, but maybe its query complexity is dramatically different!

In fact, the query complexity differs by at most 1.

Reduction, original → symmetric: $x' = x_1 x_2 \ldots x_n \bar{x}_1 \bar{x}_2 \ldots \bar{x}_n$

Symmetric ordered search ≈ Ordered search

This problem looks similar, but maybe its query complexity is dramatically different!

In fact, the query complexity differs by at most 1.

Reduction, original → symmetric: $x' = x_1 x_2 \dots x_n \bar{x}_1 \bar{x}_2 \dots \bar{x}_n$

Reduction, symmetric → original: $x' = \begin{cases} x_1 x_2 \dots x_n & x_n = 1 \\ x_{n+1} x_{n+2} \dots x_{2n} & x_n = 0 \end{cases}$

Symmetric ordered search ≈ Ordered search

This problem looks similar, but maybe its query complexity is dramatically different!

In fact, the query complexity differs by at most 1.

Reduction, original → symmetric: $x' = x_1 x_2 \dots x_n \bar{x}_1 \bar{x}_2 \dots \bar{x}_n$

Reduction, symmetric → original: $x' = \begin{cases} x_1 x_2 \dots x_n & x_n = 1 \\ x_{n+1} x_{n+2} \dots x_{2n} & x_n = 0 \end{cases}$

one extra query

Symmetric ordered search ≈ Ordered search

This problem looks similar, but maybe its query complexity is dramatically different!

In fact, the query complexity differs by at most 1.

Reduction, original → symmetric: $x' = x_1 x_2 \dots x_n \bar{x}_1 \bar{x}_2 \dots \bar{x}_n$

Reduction, symmetric → original: $x' = \begin{cases} x_1 x_2 \dots x_n & x_n = 1 \\ x_{n+1} x_{n+2} \dots x_{2n} & x_n = 0 \end{cases}$

one extra query

Asymptotically, this is negligible.

Adversary SDP for ordered search

$$\Gamma = \begin{array}{c} \begin{matrix} 11110000 & 01111000 & 00111100 & 00011110 & 00001111 & 10000111 & 11000011 & 11100001 \end{matrix} \\ \left[\begin{matrix} 0 & \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 & \gamma_5 & \gamma_6 & \gamma_7 \\ \gamma_1 & 0 & \gamma_8 & \gamma_9 & \gamma_{10} & \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_2 & \gamma_8 & 0 & \gamma_{14} & \gamma_{15} & \gamma_{16} & \gamma_{17} & \gamma_{18} \\ \gamma_3 & \gamma_9 & \gamma_{14} & 0 & \gamma_{19} & \gamma_{20} & \gamma_{21} & \gamma_{22} \\ \gamma_4 & \gamma_{10} & \gamma_{15} & \gamma_{19} & 0 & \gamma_{23} & \gamma_{24} & \gamma_{25} \\ \gamma_5 & \gamma_{11} & \gamma_{16} & \gamma_{20} & \gamma_{23} & 0 & \gamma_{26} & \gamma_{27} \\ \gamma_6 & \gamma_{12} & \gamma_{17} & \gamma_{21} & \gamma_{24} & \gamma_{26} & 0 & \gamma_{28} \\ \gamma_7 & \gamma_{13} & \gamma_{18} & \gamma_{22} & \gamma_{25} & \gamma_{27} & \gamma_{28} & 0 \end{matrix}\right] \end{array} \begin{matrix} 11110000 \\ 01111000 \\ 00111100 \\ 00011110 \\ 00001111 \\ 10000111 \\ 11000011 \\ 11100001 \end{matrix}$$

Adversary SDP for ordered search

By the automorphism principle, we can assume

$$\Gamma = \begin{array}{c} \begin{matrix} 11110000 & 01111000 & 00111100 & 00011110 & 00001111 & 10000111 & 11000011 & 11100001 \end{matrix} \\ \begin{bmatrix} 0 & \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 & \gamma_3 & \gamma_2 & \gamma_1 \\ \gamma_1 & 0 & \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 & \gamma_3 & \gamma_2 \\ \gamma_2 & \gamma_1 & 0 & \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 & \gamma_3 \\ \gamma_3 & \gamma_2 & \gamma_1 & 0 & \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 \\ \gamma_4 & \gamma_3 & \gamma_2 & \gamma_1 & 0 & \gamma_1 & \gamma_2 & \gamma_3 \\ \gamma_3 & \gamma_4 & \gamma_3 & \gamma_2 & \gamma_1 & 0 & \gamma_1 & \gamma_2 \\ \gamma_2 & \gamma_3 & \gamma_4 & \gamma_3 & \gamma_2 & \gamma_1 & 0 & \gamma_1 \\ \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 & \gamma_3 & \gamma_2 & \gamma_1 & 0 \end{bmatrix} \begin{matrix} 11110000 \\ 01111000 \\ 00111100 \\ 00011110 \\ 00001111 \\ 10000111 \\ 11000011 \\ 11100001 \end{matrix} \end{array}$$

Spectral norm achieved by uniform eigenvector: $\gamma_n + 2\sum_{i=1}^{n-1} \gamma_i$

Adversary SDP for ordered search

Also by the automorphism principle, it suffices to consider

$$\Gamma_8 = \begin{array}{c} \begin{array}{cccccccc} 11110000 & 01111000 & 00111100 & 00011110 & 00001111 & 10000111 & 11000011 & 11100001 \end{array} \\ \left[\begin{array}{cccccccc} 0 & 0 & 0 & 0 & \gamma_4 & \gamma_3 & \gamma_2 & \gamma_1 \\ 0 & 0 & 0 & 0 & \gamma_3 & \gamma_4 & \gamma_3 & \gamma_2 \\ 0 & 0 & 0 & 0 & \gamma_2 & \gamma_3 & \gamma_4 & \gamma_3 \\ 0 & 0 & 0 & 0 & \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 \\ \gamma_4 & \gamma_3 & \gamma_2 & \gamma_1 & 0 & 0 & 0 & 0 \\ \gamma_3 & \gamma_4 & \gamma_3 & \gamma_2 & 0 & 0 & 0 & 0 \\ \gamma_2 & \gamma_3 & \gamma_4 & \gamma_3 & 0 & 0 & 0 & 0 \\ \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 & 0 & 0 & 0 & 0 \end{array}\right] \begin{array}{c} 11110000 \\ 01111000 \\ 00111100 \\ 00011110 \\ 00001111 \\ 10000111 \\ 11000011 \\ 11100001 \end{array} \end{array}$$

In general, $\|\Gamma_{2n}\| = \|\mathrm{Toeplitz}(\gamma_n, \gamma_{n-1}, \ldots, \gamma_1)\|$.

Adversary SDP for ordered search

Primal:

$$\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1,\ \gamma_i \geq 0$$

Adversary SDP for ordered search

Primal:

$$\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1,\ \gamma_i \geq 0$$

Høyer, Neerbek, Shi: Let $\gamma_i = \begin{cases} \frac{1}{\pi i} & i = 1, 2, \ldots, \lfloor n/2 \rfloor \\ 0 & \text{otherwise} \end{cases}$

Adversary SDP for ordered search

Primal:

$$\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1,\ \gamma_i \geq 0$$

Høyer, Neerbek, Shi: Let $\gamma_i = \begin{cases} \frac{1}{\pi i} & i = 1, 2, \ldots, \lfloor n/2 \rfloor \\ 0 & \text{otherwise} \end{cases}$

Objective function: $2\sum_{i=1}^{\lfloor n/2 \rfloor} \frac{1}{\pi i} \approx \frac{2}{\pi} \ln n$

Adversary SDP for ordered search

Primal:

$$\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1,\ \gamma_i \geq 0$$

Høyer, Neerbek, Shi: Let $\gamma_i = \begin{cases} \frac{1}{\pi i} & i = 1, 2, \ldots, \lfloor n/2 \rfloor \\ 0 & \text{otherwise} \end{cases}$

Objective function: $2\sum_{i=1}^{\lfloor n/2 \rfloor} \frac{1}{\pi i} \approx \frac{2}{\pi} \ln n$

Hilbert matrix: $\left\| \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots \\ \frac{1}{2} & \frac{1}{3} & & \\ \frac{1}{3} & & & \\ \vdots & & & \end{bmatrix} \right\| = \pi$

Adversary SDP for ordered search

Primal:

$$\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1,\ \gamma_i \geq 0$$

Adversary SDP for ordered search

Primal:

$$\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1,\ \gamma_i \geq 0$$

Dual:

$$\min \text{tr}(P) \quad \text{subject to} \quad P \succeq 0,\ \ \text{tr}_i(P) \geq 1 \text{ for } i = 0, \ldots, n-1$$

Adversary SDP for ordered search

Primal:

$$\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1,\ \gamma_i \geq 0$$

Dual:

$$\min \text{tr}(P) \quad \text{subject to} \quad P \succeq 0,\ \ \text{tr}_i(P) \geq 1 \text{ for } i = 0, \ldots, n-1$$

Theorem.

$$\text{ADV}(\text{OSP}_{2m}) = 2\sum_{i=0}^{m-1} \left(\frac{\binom{2i}{i}}{4^i} \right)^2$$

$$\text{ADV}(\text{OSP}_{2m+1}) = 2\sum_{i=0}^{m-1} \left(\frac{\binom{2i}{i}}{4^i} \right)^2 + \left(\frac{\binom{2m}{m}}{4^m} \right)^2$$

Optimal ordered search adversary: dual

Dual:

$$\min \operatorname{tr}(P) \quad \text{subject to} \quad P \succeq 0, \;\; \operatorname{tr}_i(P) \geq 1 \text{ for } i = 0, \ldots, n-1$$

Optimal ordered search adversary: dual

Dual:

$$\min \operatorname{tr}(P) \quad \text{subject to} \quad P \succeq 0, \;\; \operatorname{tr}_i(P) \geq 1 \text{ for } i = 0, \ldots, n-1$$

Let $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

Optimal ordered search adversary: dual

Dual:

$$\min \operatorname{tr}(P) \quad \text{subject to} \quad P \succeq 0, \;\; \operatorname{tr}_i(P) \geq 1 \text{ for } i = 0, \ldots, n-1$$

Let $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

$$\vec{u} := (\xi_0, \xi_1, \ldots, \xi_{\frac{n}{2}-1}, \xi_{\frac{n}{2}-1}, \ldots, \xi_1, \xi_0)$$

Optimal ordered search adversary: dual

Dual:

$$\min \operatorname{tr}(P) \quad \text{subject to} \quad P \succeq 0, \;\; \operatorname{tr}_i(P) \geq 1 \text{ for } i = 0, \ldots, n-1$$

Let $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

$$\vec{u} := (\xi_0, \xi_1, \ldots, \xi_{\frac{n}{2}-1}, \xi_{\frac{n}{2}-1}, \ldots, \xi_1, \xi_0)$$

$$P := \vec{u}\vec{u}^T$$

Optimal ordered search adversary: dual

Dual:

$$\min \operatorname{tr}(P) \quad \text{subject to} \quad P \succeq 0, \;\; \operatorname{tr}_i(P) \geq 1 \text{ for } i = 0, \ldots, n-1$$

Let $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

$$\vec{u} := (\xi_0, \xi_1, \ldots, \xi_{\frac{n}{2}-1}, \xi_{\frac{n}{2}-1}, \ldots, \xi_1, \xi_0)$$

$$P := \vec{u}\vec{u}^T$$

Then $\operatorname{tr}(P) = 2 \sum_{i=0}^{\frac{n}{2}-1} \xi_i^2$ as claimed.

Optimal ordered search adversary: dual

Dual:

$$\min \operatorname{tr}(P) \quad \text{subject to} \quad P \succeq 0, \;\; \operatorname{tr}_i(P) \geq 1 \text{ for } i = 0, \ldots, n-1$$

Let $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

$$\vec{u} := (\xi_0, \xi_1, \ldots, \xi_{\frac{n}{2}-1}, \xi_{\frac{n}{2}-1}, \ldots, \xi_1, \xi_0)$$

$$P := \vec{u}\vec{u}^T$$

Then $\operatorname{tr}(P) = 2 \sum_{i=0}^{\frac{n}{2}-1} \xi_i^2$ as claimed.

$$\operatorname{tr}_i(P) = \sum_{j=1}^{n-i} u_j u_{i+j} = \sum_{j=1}^{n-i} u_j u_{n-i-j+1} \geq \sum_{j=0}^{n-i-1} \xi_j \xi_{n-i-j-1} = 1$$

Optimal ordered search adversary: dual

Dual:

$$\min \operatorname{tr}(P) \quad \text{subject to} \quad P \succeq 0, \;\; \operatorname{tr}_i(P) \geq 1 \text{ for } i = 0, \ldots, n-1$$

Let $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

$$\vec{u} := (\xi_0, \xi_1, \ldots, \xi_{\frac{n}{2}-1}, \xi_{\frac{n}{2}-1}, \ldots, \xi_1, \xi_0)$$

$$P := \vec{u}\vec{u}^T$$

Then $\operatorname{tr}(P) = 2 \sum_{i=0}^{\frac{n}{2}-1} \xi_i^2$ as claimed.

$$\operatorname{tr}_i(P) = \sum_{j=1}^{n-i} u_j u_{i+j} = \sum_{j=1}^{n-i} u_j u_{n-i-j+1} \geq \sum_{j=0}^{n-i-1} \xi_j \xi_{n-i-j-1} = 1$$

Primal is more technical but uses similar ideas.

A binomial identity

Recall $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

Proposition. For any j, $\displaystyle\sum_{i=0}^{j} \xi_i \, \xi_{j-i} = 1.$

A binomial identity

Recall $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

Proposition. For any j, $\displaystyle\sum_{i=0}^{j} \xi_i \, \xi_{j-i} = 1.$

i.e., $\displaystyle\sum_{i=0}^{j} \binom{2i}{i}\binom{2(j-i)}{j-i} = 4^i$

A binomial identity

Recall $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

Proposition. For any j, $\displaystyle\sum_{i=0}^{j} \xi_i \, \xi_{j-i} = 1.$

i.e., $\displaystyle\sum_{i=0}^{j} \binom{2i}{i}\binom{2(j-i)}{j-i} = 4^i$

Proof.

A binomial identity

Recall $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

Proposition. For any j, $\displaystyle\sum_{i=0}^{j} \xi_i \, \xi_{j-i} = 1.$

i.e., $\displaystyle\sum_{i=0}^{j} \binom{2i}{i}\binom{2(j-i)}{j-i} = 4^i$

Proof.

GF for $\{\xi_i\}$: $\quad g(z) := \displaystyle\sum_{i=0}^{\infty} \xi_i z^i = \frac{1}{\sqrt{1-z}}$

A binomial identity

Recall $\xi_i := \dfrac{\binom{2i}{i}}{4^i}$

Proposition. For any j, $\displaystyle\sum_{i=0}^{j} \xi_i \, \xi_{j-i} = 1.$

$$\text{i.e., } \sum_{i=0}^{j} \binom{2i}{i}\binom{2(j-i)}{j-i} = 4^i$$

Proof.

GF for $\{\xi_i\}$: $\quad g(z) := \displaystyle\sum_{i=0}^{\infty} \xi_i z^i = \frac{1}{\sqrt{1-z}}$

GF for LHS: $\quad \dfrac{1}{1-z} = \displaystyle\sum_{i=0}^{\infty} z^i$

Asymptotic analysis

$$\mathrm{ADV}(\mathrm{OSP}_n) = 2 \sum_{i=0}^{\frac{n}{2}-1} \left(\frac{\binom{2i}{i}}{4^i} \right)^2$$

Asymptotically, we have $\mathrm{ADV}(\mathrm{OSP}_n) = \frac{2}{\pi}(\ln n + \gamma + \ln 8) + O(1/n)$

Asymptotic analysis

$$\mathrm{ADV}(\mathrm{OSP}_n) = 2 \sum_{i=0}^{\frac{n}{2}-1} \left(\frac{\binom{2i}{i}}{4^i} \right)^2$$

Asymptotically, we have $\mathrm{ADV}(\mathrm{OSP}_n) = \dfrac{2}{\pi}(\ln n + \gamma + \ln 8) + O(1/n)$

Proof. GF of $\{\mathrm{ADV}(\mathrm{OSP}_{2m})\}$ is $\dfrac{2 \cdot {}_2F_1(\frac{1}{2}, \frac{1}{2}; 1; z)}{1 - z}$

Result follows by analyzing the logarithmic singularity at $z = 1$ using Darboux's method.

Asymptotic analysis

$$\mathrm{ADV}(\mathrm{OSP}_n) = 2 \sum_{i=0}^{\frac{n}{2}-1} \left(\frac{\binom{2i}{i}}{4^i} \right)^2$$

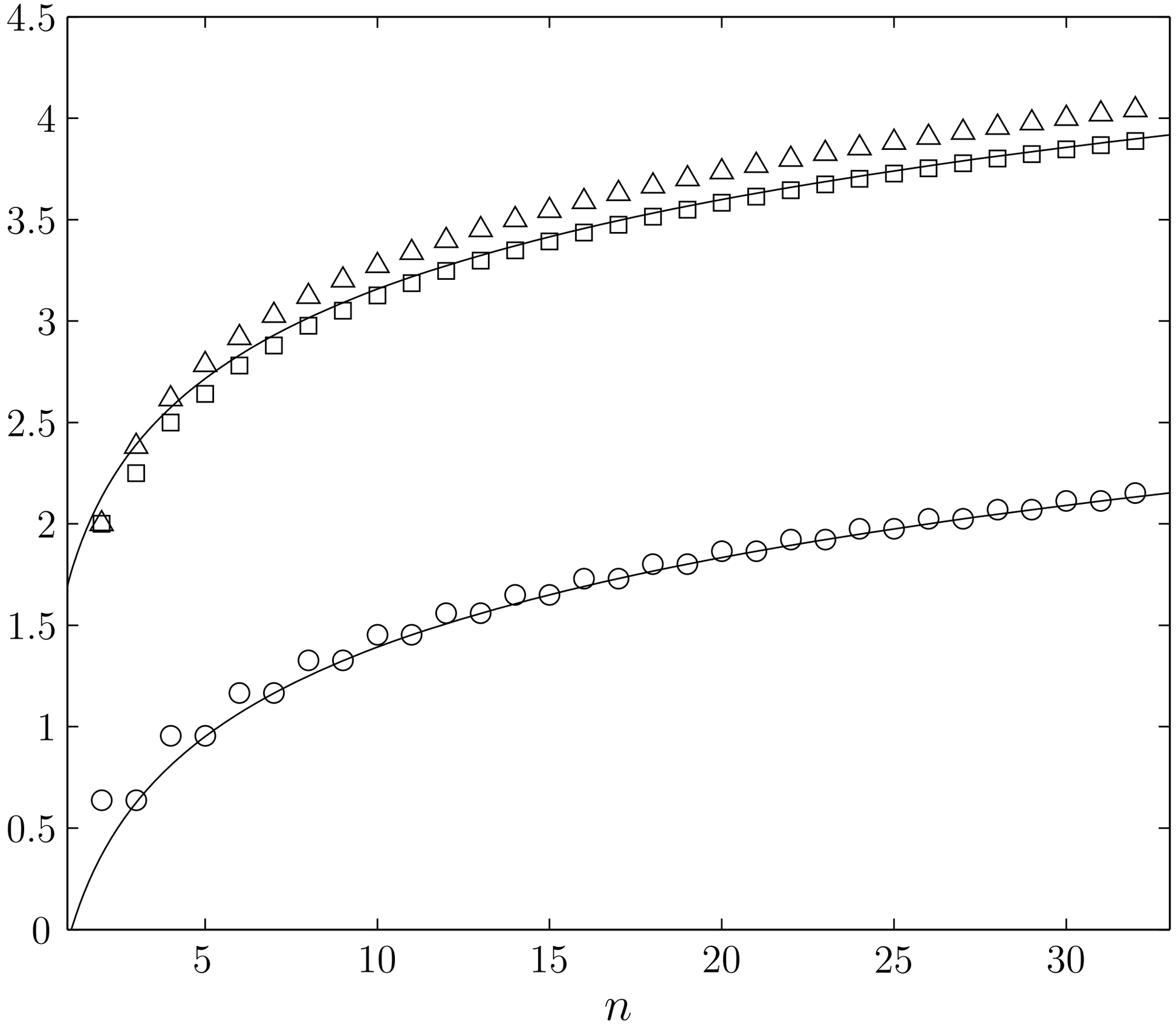
Asymptotically, we have $\mathrm{ADV}(\mathrm{OSP}_n) = \dfrac{2}{\pi}(\ln n + \gamma + \ln 8) + O(1/n)$

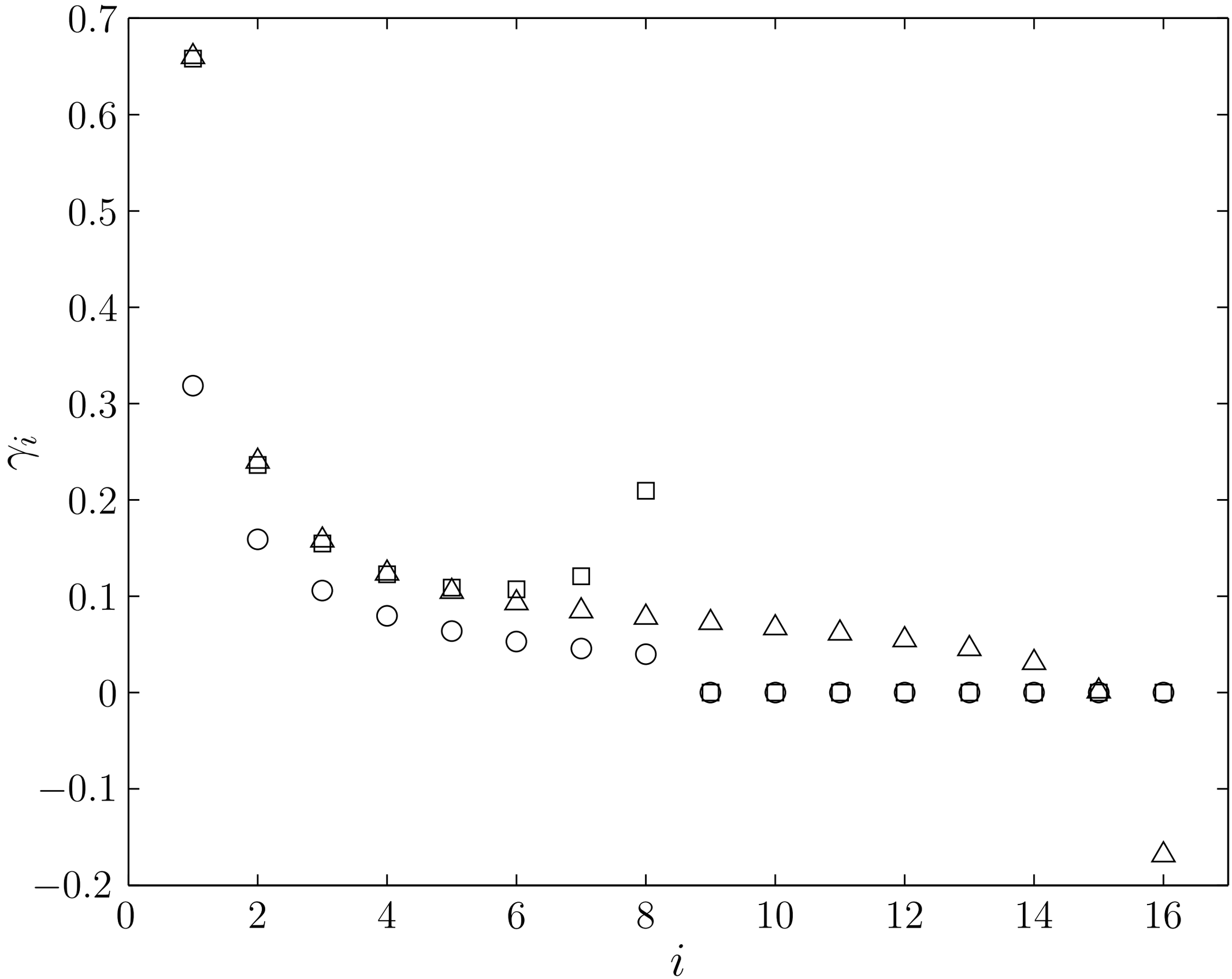
Proof. GF of $\{\mathrm{ADV}(\mathrm{OSP}_{2m})\}$ is $\dfrac{2 \cdot {}_2F_1(\frac{1}{2}, \frac{1}{2}; 1; z)}{1 - z}$

Result follows by analyzing the logarithmic singularity at $z = 1$ using Darboux's method.

For comparison, the HNS bound says

$$\mathrm{ADV}(\mathrm{OSP}_n) \geq \frac{2}{\pi}(\ln n + \gamma - \ln 2) + O(1/n)$$


4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
5
10
15
20
25
30
n


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
γ_i
0
2
4
6
8
10
12
14
16
i

The negative adversary

Recall definition of adversary: $\mathrm{ADV}(f) := \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$

The negative adversary

Recall definition of adversary: $\mathrm{ADV}(f) := \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$

Negative adversary: $\mathrm{ADV}^{\pm}(f) := \max_{\Gamma \neq 0} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$

The negative adversary

Recall definition of adversary: $\mathrm{ADV}(f) := \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$

Negative adversary: $\mathrm{ADV}^{\pm}(f) := \max_{\Gamma \neq 0} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$

Theorem [Høyer, Lee, Špalek 07]:

(Quantum query complexity of f) $\geq \frac{1}{2}\mathrm{ADV}^{\pm}(f) \geq \frac{1}{2}\mathrm{ADV}(f)$.

The negative adversary

Recall definition of adversary: $\mathrm{ADV}(f) := \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$

Negative adversary: $\mathrm{ADV}^{\pm}(f) := \max_{\Gamma \neq 0} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$

Theorem [Høyer, Lee, Špalek 07]:

(Quantum query complexity of f) $\geq \frac{1}{2}\mathrm{ADV}^{\pm}(f) \geq \frac{1}{2}\mathrm{ADV}(f)$.

Furthermore, there are functions for which the negative adversary gives a significantly better lower bound.

Negative adversary for ordered search

Negative adversary for ordered search

Primal: $\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1$

Negative adversary for ordered search

Primal: $\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1$

Dual: $\min \text{tr}(P+Q) \quad \text{subject to} \quad P, Q \succeq 0, \;\; \text{tr}_i(P-Q) = 1$

Negative adversary for ordered search

Primal: $\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1$

Dual: $\min \text{tr}(P+Q) \quad \text{subject to} \quad P, Q \succeq 0, \;\; \text{tr}_i(P-Q) = 1$

Theorem. $\text{ADV}^{\pm}(\text{OSP}_n) \leq \text{ADV}(\text{OSP}_{2n}) + 1$

Negative adversary for ordered search

Primal: $\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1$

Dual: $\min \operatorname{tr}(P+Q) \quad \text{subject to} \quad P, Q \succeq 0, \;\; \operatorname{tr}_i(P-Q) = 1$

Theorem. $\mathrm{ADV}^{\pm}(\mathrm{OSP}_n) \leq \mathrm{ADV}(\mathrm{OSP}_{2n}) + 1$

Idea: Given $R = P - Q$ satisfying $\operatorname{tr}_i R = 1$, objective is $\operatorname{tr}|R|$.

Negative adversary for ordered search

Primal: $\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \leq 1$

Dual: $\min \operatorname{tr}(P+Q) \quad \text{subject to} \quad P, Q \succeq 0, \;\; \operatorname{tr}_i(P-Q) = 1$

Theorem. $\text{ADV}^{\pm}(\text{OSP}_n) \leq \text{ADV}(\text{OSP}_{2n}) + 1$

Idea: Given $R = P - Q$ satisfying $\operatorname{tr}_i R = 1$, objective is $\operatorname{tr}|R|$.

With $\vec{v} := (\xi_0, \xi_1, \ldots, \xi_{n-1}), \;\; \vec{w} := (\xi_{n-1}, \ldots, \xi_1, \xi_0)$,
the matrix $\vec{v}\vec{w}^T$ has correct above-diagonal traces.

Negative adversary for ordered search

Primal: $\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \le 1$

Dual: $\min \operatorname{tr}(P+Q) \quad \text{subject to} \quad P, Q \succeq 0, \;\; \operatorname{tr}_i(P-Q) = 1$

Theorem. $\text{ADV}^{\pm}(\text{OSP}_n) \le \text{ADV}(\text{OSP}_{2n}) + 1$

Idea: Given $R = P - Q$ satisfying $\operatorname{tr}_i R = 1$, objective is $\operatorname{tr}|R|$.

With $\vec{v} := (\xi_0, \xi_1, \ldots, \xi_{n-1}), \;\; \vec{w} := (\xi_{n-1}, \ldots, \xi_1, \xi_0)$,
the matrix $\vec{v}\vec{w}^T$ has correct above-diagonal traces.

Replace below-diagonal entries with the above-diagonal ones.

Negative adversary for ordered search

Primal: $\max \gamma_n + 2\sum_{i=1}^{n-1} \gamma_i \quad \text{subject to} \quad \|\text{Toeplitz}(\gamma_n, \ldots, \gamma_1)\| \le 1$

Dual: $\min \operatorname{tr}(P+Q) \quad \text{subject to} \quad P, Q \succeq 0, \;\; \operatorname{tr}_i(P-Q) = 1$

Theorem. $\text{ADV}^{\pm}(\text{OSP}_n) \le \text{ADV}(\text{OSP}_{2n}) + 1$

Idea: Given $R = P - Q$ satisfying $\operatorname{tr}_i R = 1$, objective is $\operatorname{tr}|R|$.

With $\vec{v} := (\xi_0, \xi_1, \ldots, \xi_{n-1}), \;\; \vec{w} := (\xi_{n-1}, \ldots, \xi_1, \xi_0)$,
the matrix $\vec{v}\vec{w}^T$ has correct above-diagonal traces.

Replace below-diagonal entries with the above-diagonal ones.

We give a general analysis of the spectra of such matrices.

Summary

Summary

Quantum computers can search ordered lists faster than classical computers, by a constant factor between 2.3 and 4.6.

Summary

Quantum computers can search ordered lists faster than classical computers, by a constant factor between 2.3 and 4.6.

To find that constant, we will have to

- Find a better algorithm, and/or
- Prove a better lower bound by a non-adversary technique

Summary

Quantum computers can search ordered lists faster than classical computers, by a constant factor between 2.3 and 4.6.

To find that constant, we will have to

- Find a better algorithm, and/or
- Prove a better lower bound by a non-adversary technique

Open problems

Summary

Quantum computers can search ordered lists faster than classical computers, by a constant factor between 2.3 and 4.6.

To find that constant, we will have to

- Find a better algorithm, and/or
- Prove a better lower bound by a non-adversary technique

Open problems

- What is the constant?

Summary

Quantum computers can search ordered lists faster than classical computers, by a constant factor between 2.3 and 4.6.

To find that constant, we will have to

- Find a better algorithm, and/or
- Prove a better lower bound by a non-adversary technique

Open problems

- What is the constant?
- Can we use insights from the optimal adversary to find a better algorithm? (Note: Quantum query complexity is an SDP.)

Summary

Quantum computers can search ordered lists faster than classical computers, by a constant factor between 2.3 and 4.6.

To find that constant, we will have to

- Find a better algorithm, and/or
- Prove a better lower bound by a non-adversary technique

Open problems

- What is the constant?
- Can we use insights from the optimal adversary to find a better algorithm? (Note: Quantum query complexity is an SDP.)
- Can we find optimal adversary lower bounds for other problems? (Element distinctness?)