Quantum walks
Daniel J. Bernstein
University of Illinois at Chicago

Focusing on quantum walks as an algorithm-design tool:
e.g. Grover’s algorithm.
e.g. Ambainis’s algorithm.

Can also study quantum walks on much more general graphs.
2008 Childs, 2009 Lovett–Cooper–Everitt–Trevers–Kendon:
Can view, e.g., Shor’s algorithm as quantum walk on Shor graph.

Examples of applications to crypto
Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>cipher</td>
</tr>
<tr>
<td>0.791</td>
<td>0.462</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.290</td>
<td>0.241</td>
<td>MQ</td>
</tr>
</tbody>
</table>

“Pre-q” $e$: as $n \to \infty$, $2^{(e+o(1))n}$ simple non-quantum ops.

“Post-q” $e$: as $n \to \infty$, $2^{(e+o(1))n}$ simple quantum ops.

“Cipher”: find $n$-bit cipher key.
0.5: 1996 Grover.
Quantum walks

Daniel J. Bernstein
University of Illinois at Chicago

Focusing on quantum walks as an algorithm-design tool:
e.g. Grover’s algorithm.
e.g. Ambainis’s algorithm.

Can also study quantum walks on much more general graphs.

Childs, 2009 Lovett–Cooper–Everitt–Trevers–Kendon:
Can view, e.g., Shor’s algorithm as quantum walk on Shor graph.

Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>cipher</td>
</tr>
<tr>
<td>ρ</td>
<td>ρ/2</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.791…</td>
<td>0.462…</td>
<td>MQ</td>
</tr>
<tr>
<td>0.290…</td>
<td>0.241…</td>
<td>subset sum</td>
</tr>
</tbody>
</table>

“Pre-q” e: as $n \to \infty$, $2^{(e+o(1))n}$ simple non-quantum ops.

“Post-q” e: as $n \to \infty$, $2^{(e+o(1))n}$ simple quantum ops.

“Cipher”: find $n$-bit cipher key.

0.5: 1996 Grover.

“McEliece”: in linear code of length $(1+o(1))n \log_2 n$ and dimension $(R+o(1))n \log_2 n$, decode $(1-R+o(1))n$ errors.
Quantum walks

Daniel J. Bernstein
University of Illinois at Chicago

Focusing on quantum walks as an algorithm-design tool:
  e.g. Grover’s algorithm.
  e.g. Ambainis’s algorithm.
Can also study quantum walks on much more general graphs.

2008 Childs, 2009 Lovett–Cooper–Everitt–Trevers–Kendon:
Can view, e.g., Shor’s algorithm as quantum walk on Shor graph.

Examples of applications to crypto
Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>cipher</td>
</tr>
<tr>
<td>\rho</td>
<td>\rho/2</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.791</td>
<td>0.462</td>
<td>MQ</td>
</tr>
<tr>
<td>0.290</td>
<td>0.241</td>
<td>subset sum</td>
</tr>
</tbody>
</table>

“Pre-q” e: as \( n \to \infty \), \( 2^{(e+o(1))n} \)
  simple non-quantum ops.

“Post-q” e: as \( n \to \infty \), \( 2^{(e+o(1))n} \)
  simple quantum ops.

“Cipher”: find \( n \)-bit cipher key.
0.5: 1996 Grover.

“McEliece”: in linear code of length \( (1 + o(1))n \log_2 n \) and dimension \( (R + o(1))n \log_2 n \),
  decode \( (1 - R + o(1))n \) errors.
Examples of applications to crypto
Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>cipher</td>
</tr>
<tr>
<td>ρ</td>
<td>ρ/2</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.791...</td>
<td>0.462...</td>
<td>MQ</td>
</tr>
<tr>
<td>0.290...</td>
<td>0.241...</td>
<td>subset sum</td>
</tr>
</tbody>
</table>

"Pre-q" e: as $n \to \infty$, $2^{(e+o(1))n}$ simple non-quantum ops.

"Post-q" e: as $n \to \infty$, $2^{(e+o(1))n}$ simple quantum ops.

"Cipher": find $n$-bit cipher key.
0.5: 1996 Grover.

"McEliece": in linear code of length $(1 + o(1))n \log_2 n$ and dimension $(R + o(1))n \log_2 n$, decode $(1 - R + o(1))n$ errors.

"Cipher": find $n$-bit cipher key.
0.5: 1996 Grover.
Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>cipher</td>
</tr>
<tr>
<td>$\rho$</td>
<td>$\rho/2$</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.791...</td>
<td>0.462...</td>
<td>MQ</td>
</tr>
<tr>
<td>0.290...</td>
<td>0.241...</td>
<td>subset sum</td>
</tr>
</tbody>
</table>

“Pre-q” $e$: as $n \to \infty$, $2^{(e+o(1))n}$ simple non-quantum ops.

“Post-q” $e$: as $n \to \infty$, $2^{(e+o(1))n}$ simple quantum ops.

“Cipher”: find $n$-bit cipher key.
0.5: 1996 Grover.

“McEliece”: in linear code of length $(1 + o(1))n \log_2 n$ and dimension $(R + o(1))n \log_2 n$, decode $(1 - R + o(1))n$ errors.
Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>cipher</td>
</tr>
<tr>
<td>ρ</td>
<td>ρ/2</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.791</td>
<td>0.462</td>
<td>MQ</td>
</tr>
<tr>
<td>0.290</td>
<td>0.241</td>
<td>subset sum</td>
</tr>
</tbody>
</table>

“Pre-q” e: as \( n \to \infty \), \( 2^{(e+o(1))n} \) simple non-quantum ops.

“Post-q” e: as \( n \to \infty \), \( 2^{(e+o(1))n} \) simple quantum ops.

“Cipher”: find \( n \)-bit cipher key.

0.5: 1996 Grover.

“McEliece”: in linear code of length \((1 + o(1))n \log_2 n\) and dimension \((R + o(1))n \log_2 n\), decode \((1 − R + o(1))n\) errors.

\( ρ = (1 − R) \log_2(1/(1 − R))\): 1962 Prange.
Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>cipher</td>
</tr>
<tr>
<td>$\rho$</td>
<td>$\rho/2$</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.791...</td>
<td>0.462...</td>
<td>MQ</td>
</tr>
<tr>
<td>0.290...</td>
<td>0.241...</td>
<td>subset sum</td>
</tr>
</tbody>
</table>

“Pre-q” $e$: as $n \to \infty$, $2^{(e+o(1))n}$ simple non-quantum ops.

“Post-q” $e$: as $n \to \infty$, $2^{(e+o(1))n}$ simple quantum ops.

“Cipher”: find $n$-bit cipher key.
0.5: 1996 Grover.

“McEliece”: in linear code of length $(1 + o(1))n \log_2 n$ and dimension $(R + o(1))n \log_2 n$, decode $(1 - R + o(1))n$ errors.

$\rho = (1 - R) \log_2 (1/(1 - R))$: 1962 Prange.

$\rho/2$: 2009 Bernstein (via Grover).
Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>cipher</td>
</tr>
<tr>
<td>$\rho$</td>
<td>$\rho/2$</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.791...</td>
<td>0.462...</td>
<td>MQ</td>
</tr>
<tr>
<td>0.290...</td>
<td>0.241...</td>
<td>subset sum</td>
</tr>
</tbody>
</table>

“Pre-q” $e$: as $n \to \infty$, $2^{(e + o(1))n}$ simple non-quantum ops.

“Post-q” $e$: as $n \to \infty$, $2^{(e + o(1))n}$ simple quantum ops.

“Cipher”: find $n$-bit cipher key.

0.5: 1996 Grover.

“McEliece”: in linear code of length $(1 + o(1))n \log_2 n$ and dimension $(R + o(1))n \log_2 n$, decode $(1 - R + o(1))n$ errors.

$\rho = (1 - R)\log_2(1/(1 - R))$: 1962 Prange.

$\rho/2$: 2009 Bernstein (via Grover).

“MQ”: solve system of $n$ deg-2 equations in $n$ variables over $\mathbb{F}_2$. 
Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>cipher</td>
</tr>
<tr>
<td>ρ</td>
<td>ρ/2</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.791</td>
<td>0.462</td>
<td>MQ</td>
</tr>
<tr>
<td>0.290</td>
<td>0.241</td>
<td>subset sum</td>
</tr>
</tbody>
</table>

“Pre-q” e: as \( n \to \infty \), \( 2^{(e+o(1))n} \) simple non-quantum ops.

“Post-q” e: as \( n \to \infty \), \( 2^{(e+o(1))n} \) simple quantum ops.

“Cipher”: find \( n \)-bit cipher key.
0.5: 1996 Grover.

“McEliece”: in linear code of length \((1 + o(1))n \log_2 n\) and dimension \((R + o(1))n \log_2 n\), decode \((1 - R + o(1))n\) errors.
\[ \rho = (1 - R) \log_2(1/(1 - R)) \]
1962 Prange.
\[ \rho/2 : 2009 \text{ Bernstein (via Grover).} \]

“MQ”: solve system of \( n \) deg-2 equations in \( n \) variables over \( \mathbb{F}_2 \).
0.791 (modulo calculation errors): 2004 Yang–Chen–Courtois.
Examples of applications to crypto
Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>cipher</td>
</tr>
<tr>
<td>ρ</td>
<td>ρ/2</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.791...</td>
<td>0.462...</td>
<td>MQ</td>
</tr>
<tr>
<td>0.290...</td>
<td>0.241...</td>
<td>subset sum</td>
</tr>
</tbody>
</table>

“Pre-q” e: as $n \to \infty$, $2^{(e+o(1))n}$ simple non-quantum ops.

“Post-q” e: as $n \to \infty$, $2^{(e+o(1))n}$ simple quantum ops.

“Cipher”: find $n$-bit cipher key.
0.5: 1996 Grover.

“McEliece”: in linear code of length $(1 + o(1))n \log_2 n$ and dimension $(R + o(1))n \log_2 n$, decode $(1 - R + o(1))n$ errors. $\rho = (1 - R) \log_2(1/(1 - R))$: 1962 Prange.
ρ/2: 2009 Bernstein (via Grover).

“MQ”: solve system of $n$ degree-2 equations in $n$ variables over $F_2$. 0.791 (modulo calculation errors): 2004 Yang–Chen–Courtois.
Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>pre-q</th>
<th>post-q</th>
<th>problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>cipher</td>
<td>cipher</td>
</tr>
<tr>
<td>ρ/2</td>
<td>McEliece</td>
<td>McEliece</td>
</tr>
<tr>
<td>0.462...</td>
<td>MQ</td>
<td>MQ</td>
</tr>
<tr>
<td>0.241...</td>
<td>subset sum</td>
<td>subset sum</td>
</tr>
</tbody>
</table>

“pre-q” e: as n → ∞, 2^{(e+o(1))n} non-quantum ops.

“post-q” e: as n → ∞, 2^{(e+o(1))n} quantum ops.

“Cipher”: find n-bit cipher key.

0: 5: 1996 Grover.

“McEliece”: in linear code of length (1 + o(1))n log₂ n and dimension (R + o(1))n log₂ n, decode (1 − R + o(1))n errors.

ρ = (1 − R) log₂(1/(1 − R)):

1962 Prange.

ρ/2: 2009 Bernstein (via Grover).

“MQ”: solve system of n deg-2 equations in n variables over F₂.

0.791 (modulo calculation errors):

2004 Yang–Chen–Courtois.


“Subset sum” (‘hard’ case): find S ⊆ {1, 2, ..., n} given x₁, x₂, ..., xₙ ∈ {0, 1, ..., 2ⁿ−1} and ∑ᵢ∈S xᵢ.
Applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

<table>
<thead>
<tr>
<th>problem</th>
<th>pre-q</th>
<th>post-q</th>
</tr>
</thead>
<tbody>
<tr>
<td>cipher</td>
<td></td>
<td></td>
</tr>
<tr>
<td>McEliece</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subset sum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$e : \text{as } n \to \infty, 2^{(e+o(1))n}$

simple non-quantum ops.

$\rho = (1 - R) \log_2(1/(1 - R))$: 1962 Prange.

$\rho/2$: 2009 Bernstein (via Grover).

$\rho/2$: 2009 Bernstein (via Grover).

“McEliece”: in linear code of length \((1 + o(1)) n \log_2 n\) and
dimension \((R + o(1)) n \log_2 n\),
decode \((1 - R + o(1)) n\) errors.

\(\rho = (1 - R) \log_2(1/(1 - R))\): 1962 Prange.

$\rho/2$: 2009 Bernstein (via Grover).

“MQ”: solve system of \(n\) \(\deg-2\) equations in \(n\) variables over \(\mathbb{F}_2\).

0.791 (modulo calculation errors): 2004 Yang–Chen–Courtois.

0.462: 2017 Bernstein–Yang (via Grover), independently 2017

“Subset sum” (“hard” case):
find \(S \subseteq \{1, 2, \ldots, n\}\) given
\(x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}\)
and \(\sum_{i \in S} x_i\).
Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

pre-q | post-q | problem
--- | --- | ---
| | | 0.5 |
| cipher | | |
| | | |
| | | |

McEliece: in linear code of length \((1 + o(1))n\log_2 n\) and dimension \((R + o(1))n\log_2 n\), decode \(\rho = (1 - R)\log_2(1/(1-o(1))n)\) errors.

\[ \rho/2 \]

"Cipher": find \(n\)-bit cipher key.

0: 5: 1996 Grover.

\[ \rho \]

MQ: solve system of \(n\) deg-2 equations in \(n\) variables over \(F_2\).

0.462: 2017 Bernstein–Yang (via Grover).

0.791 (modulo calculation errors):

1962 Prange.

\[ \frac{1}{2} \]

Subset sum ("hard" case):

find \(S \subseteq \{1, 2, \ldots, n\}\) given \(x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}\) and \(\sum_{i \in S} x_i = \rho\).

2004 Yang–Chen–Courtois.

2009 Bernstein (via Grover).

“McEliece”: in linear code of length $(1 + o(1))n \log_2 n$ and dimension $(R + o(1))n \log_2 n$, decode $(1 - R + o(1))n$ errors. 
$$\rho = (1 - R) \log_2(1/(1 - R)):$$ 1962 Prange. 
$$\rho/2:$$ 2009 Bernstein (via Grover). 

“ MQ”: solve system of $n$ deg-2 equations in $n$ variables over $F_2$. 
0.791 (modulo calculation errors): 2004 Yang–Chen–Courtois. 

“Subset sum” (“hard” case): find $S \subseteq \{1, 2, \ldots, n\}$ given $x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}$ and $\sum_{i \in S} x_i$. 
“McEliece”: in linear code of length $(1 + o(1)) n \log_2 n$ and dimension $(R + o(1)) n \log_2 n$, decode $(1 - R + o(1)) n$ errors. 

$\rho = (1 - R) \log_2(1/(1 - R))$: 1962 Prange.

$\rho/2$: 2009 Bernstein (via Grover).

“MQ”: solve system of $n$ deg-2 equations in $n$ variables over $\mathbb{F}_2$.

$0.791$ (modulo calculation errors): 2004 Yang–Chen–Courtois.


“Subset sum” (“hard” case): find $S \subseteq \{1, 2, \ldots, n\}$ given $x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}$ and $\sum_{i \in S} x_i$.

$0.5$: easy.
“McEliece”: in linear code of length \((1 + o(1))n \log_2 n\) and dimension \((R + o(1))n \log_2 n\), decode \((1 - R + o(1))n\) errors. 
\[
\rho = (1 - R) \log_2 (1/(1 - R)):
\]
1962 Prange.
\[
\rho/2: 2009 \text{ Bernstein (via Grover).}
\]
“MQ”: solve system of \(n\) deg-2 equations in \(n\) variables over \(\mathbb{F}_2\).
0.791 (modulo calculation errors): 2004 Yang–Chen–Courtois.

“Subset sum” ("hard" case): find \(S \subseteq \{1, 2, \ldots, n\}\) given \(x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}\) and \(\sum_{i \in S} x_i\).
0.5: easy.
0.337: 2010 Howgrave-Graham–Joux. Claimed 0.311; error discovered by May–Meurer.
“McEliece”: in linear code of length $(1 + o(1))n \log_2 n$ and dimension $(R + o(1))n \log_2 n$, decode $(1 - R + o(1))n$ errors. $\rho = (1 - R) \log_2(1/(1 - R))$: 1962 Prange. $\rho/2$: 2009 Bernstein (via Grover).


“Subset sum” (“hard” case): find $S \subseteq \{1, 2, \ldots, n\}$ given $x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}$ and $\sum_{i \in S} x_i$.

0.5: easy.

0.337: 2010 Howgrave-Graham–Joux. Claimed 0.311; error discovered by May–Meurer.

“McEliece”: in linear code of length \((1 + o(1))n \log_2 n\) and dimension \((R + o(1))n \log_2 n\), decode \((1 - R + o(1))n\) errors.

\[ \rho = (1 - R) \log_2 (1/(1 - R)): \]
1962 Prange.

\[ \rho/2: \] 2009 Bernstein (via Grover).

“MQ”: solve system of \(n\) deg-2 equations in \(n\) variables over \(F_2\).

\[ 0.791 \text{ (modulo calculation errors)}: \] 2004 Yang–Chen–Courtois.


“Subset sum” (“hard” case): find \(S \subseteq \{1, 2, \ldots, n\}\) given \(x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}\) and \(\sum_{i \in S} x_i\).

0.5: easy.

0.337: 2010 Howgrave-Graham–Joux. Claimed 0.311; error discovered by May–Meurer.


McEliece: in linear code of length $(1 + o(1)) n \log_2 n$ and dimension $(R + o(1)) n \log_2 n$, decode $(1 - R + o(1)) n$ errors.

\[ \bar{c} = (1 - R) \log_2 (1/(1 - R)) \]

2009 Bernstein (via Grover).

“Subset sum” (“hard” case):
find $S \subseteq \{1, 2, \ldots, n\}$ given $x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}$ and $\sum_{i \in S} x_i$.

0.5: easy.

0.337: 2010 Howgrave-Graham–Joux. Claimed 0.311; error discovered by May–Meurer.


Grover’s algorithm
Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find $s$:
compute $f$ for many inputs, hope to find output 0.
Success probability is very low until $\#\text{inputs} \approx 2^n$. 

Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find $s$:
compute $f$ for many inputs, hope to find output 0.
Success probability is very low until $\#\text{inputs} \approx 2^n$. 

solve system of $n$ deg-2 equations in $n$ variables over $\mathbb{F}_2$.
(modulo calculation errors): Yang–Chen–Courtois.
“McEliece”: in linear code of length $(1 + o(1))n \log_2 n$ and dimension $(R + o(1))n \log_2 n$, decode $(1 - R + o(1))n$ errors. 

\[ \text{1/}(1 - R) \] 

Grover (via Grover).

"MQ": solve system of $n$ deg-2 equations in $n$ variables over $F_2$. (Calculation errors): Courtois.

Bernstein–Yang independently 2017

Fa\` uge–Horan–Kahrobaei–Kaplan–Kashefi–Perret.

“Subset sum” (“hard” case): find $S \subseteq \{1, 2, \ldots, n\}$ given $x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}$ and $\sum_{i \in S} x_i$.

0.5: easy.

0.337: 2010 Howgrave-Graham–Joux. Claimed 0.311; error discovered by May–Meurer.


“Subset sum” ("hard" case):
find \( S \subseteq \{1, 2, \ldots, n\} \) given
\( x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\} \)
and \( \sum_{i \in S} x_i \).

0.5: easy.

0.337: 2010 Howgrave-Graham-Joux. Claimed 0.311; error
discovered by May–Meurer.


0.241: 2013 Bernstein–Jeffery–Lange–Meurer, using HGJ and
quantum walks (not just Grover).

Grover’s algorithm
Assume: unique \( s \in \{0, 1\}^n \) has \( f(s) = 0 \).

Traditional algorithm to find \( s \):
compute \( f \) for many inputs,
hope to find output 0.
Success probability is very low
until \#inputs approaches \( 2^n \).
“Subset sum” ("hard" case):
find $S \subseteq \{1, 2, \ldots, n\}$ given
$x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}$
and $\sum_{i \in S} x_i$.

0.5: easy.

0.337: 2010 Howgrave-Graham–Joux. Claimed 0.311; error
discovered by May–Meurer.


0.241: 2013 Bernstein–Jeffery–Lange–Meurer, using HGJ and
quantum walks (not just Grover).

Grover’s algorithm
Assume: unique $s \in \{0, 1\}^n$
has $f(s) = 0$.

Traditional algorithm to find $s$:
compute $f$ for many inputs,
hope to find output 0.
Success probability is very low
until #inputs approaches $2^n$. 
“Subset sum” ("hard" case):
find \( S \subseteq \{1, 2, \ldots, n\} \) given
\( x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\} \)
and \( \sum_{i \in S} x_i \).

0.5: easy.

0.337: 2010 Howgrave-Graham–Joux. Claimed 0.311; error
discovered by May–Meurer.


0.241: 2013 Bernstein–Jeffery–Lange–Meurer, using HGJ and
quantum walks (not just Grover).

Grover’s algorithm
Assume: unique \( s \in \{0, 1\}^n \) has \( f(s) = 0 \).

Traditional algorithm to find \( s \):
compute \( f \) for many inputs,
hope to find output 0.
Success probability is very low
until \#inputs approaches \( 2^n \).

Grover’s algorithm takes only \( 2^{n/2} \)
reversible computations of \( f \).
Typically: reversibility overhead
is small enough that this easily
wins for all sufficiently large \( n \).
Subset sum ("hard" case):

\[ S \subseteq \{1, 2, \ldots, n\} \]

given \( x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\} \),
and

\[ \sum_{i \in S} x_i. \]

Grover’s algorithm

Assume: unique \( s \in \{0, 1\}^n \) has \( f(s) = 0. \)

Traditional algorithm to find \( s \):
compute \( f \) for many inputs,
hope to find output 0.
Success probability is very low
until \#inputs approaches \( 2^n \).

Grover’s algorithm takes only \( 2^{n/2} \)
reversible computations of \( f \).
Typically: reversibility overhead
is small enough that this easily
wins for all sufficiently large \( n \).

Start from uniform superposition \( a \) over \( q \in \{0, 1\}^n \): \( a_q = 2^{-n} \).
"Subset sum" ("hard" case):

find $S \subseteq \{1, 2, \ldots, n\}$ given

$x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^n - 1\}$

and

$\sum_{i \in S} x_i$.


**Grover’s algorithm**

Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find $s$:
compute $f$ for many inputs,
hope to find output 0.
Success probability is very low until #inputs approaches $2^n$.

Grover’s algorithm takes only $2^{n/2}$ reversible computations of $f$.

Typically: reversibility overhead is small enough that this easily wins for all sufficiently large $n$.

Start from uniform superposition $a$ over $q \in \{0, 1\}^n$.
**Subset sum** ("hard" case):
find $S \subseteq \{1, 2, \ldots, n\}$ given $x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 2^{n-1}\}$ and $\sum_{i \in S} x_i$.

0: 5: easy.


---

**Grover’s algorithm**

Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find $s$:
compute $f$ for many inputs,
hope to find output 0.
Success probability is very low until \#inputs approaches $2^n$.

Grover’s algorithm takes only $2^{n/2}$ reversible computations of $f$.
Typically: reversibility overhead is small enough that this easily wins for all sufficiently large $n$.

---

Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n}$. 
Grover’s algorithm

Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find $s$: compute $f$ for many inputs, hope to find output 0. Success probability is very low until #inputs approaches $2^n$.

Grover’s algorithm takes only $2^{n/2}$ reversible computations of $f$. Typically: reversibility overhead is small enough that this easily wins for all sufficiently large $n$.

Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$. 
Grover’s algorithm
Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find $s$: compute $f$ for many inputs, hope to find output 0. Success probability is very low until \#inputs approaches $2^n$.

Grover’s algorithm takes only $2^{n/2}$ reversible computations of $f$. Typically: reversibility overhead is small enough that this easily wins for all sufficiently large $n$.

Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where $b_q = -a_q$ if $f(q) = 0$, $b_q = a_q$ otherwise. This is fast.
Grover’s algorithm

Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find $s$: compute $f$ for many inputs, hope to find output 0.
Success probability is very low until #inputs approaches $2^n$.

Grover’s algorithm takes only $2^{n/2}$ reversible computations of $f$.
Typically: reversibility overhead is small enough that this easily wins for all sufficiently large $n$.

Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where
- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.
This is fast.

Step 2: “Grover diffusion”. Negate $a$ around its average.
This is also fast.
Grover’s algorithm

Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find $s$: compute $f$ for many inputs, hope to find output 0. Success probability is very low until $\#\text{inputs}$ approaches $2^n$.

Grover’s algorithm takes only $2^{n/2}$ reversible computations of $f$. Typically: reversibility overhead is small enough that this easily wins for all sufficiently large $n$.

Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where $b_q = -a_q$ if $f(q) = 0$, $b_q = a_q$ otherwise. This is fast.

Step 2: “Grover diffusion”. Negate $a$ around its average. This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.
Grover’s algorithm
Assume: unique \( s \in \{0, 1\}^n \)
has \( f(s) = 0 \).

Traditional algorithm to find \( s \):
compute \( f \) for many inputs,
hope to find output 0.
Success probability is very low until \#inputs approaches \( 2^n \).

Grover’s algorithm takes only \( 2^{n/2} \) reversible computations of \( f \).
Typically: reversibility overhead is small enough that this easily wins for all sufficiently large \( n \).

Start from uniform superposition \( a \) over \( q \in \{0, 1\}^n \):
\( a_q = 2^{-n/2} \).

Step 1: Set \( a \leftarrow b \) where
\( b_q = -a_q \) if \( f(q) = 0 \),
\( b_q = a_q \) otherwise.
This is fast.

Step 2: “Grover diffusion”.
Negate \( a \) around its average.
This is also fast.

Repeat Step 1 + Step 2
about \( 0.58 \cdot 2^{n/2} \) times.

Measure the \( n \) qubits.
With high probability this finds \( s \).
Grover’s algorithm

unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find $s$: compute $f$ for many inputs, hope to find output 0.

Success probability is very low until #inputs approaches $2^n$.

Grover’s algorithm takes only $2^{n/2}$ reversible computations of $f$.

Typically: reversibility overhead is small enough that this easily wins for all sufficiently large $n$.

Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where $b_q = -a_q$ if $f(q) = 0$,

$b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.

With high probability this finds $s$. 

Typically: reversibility overhead is small enough that this easily wins for all sufficiently large $n$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after 0 steps: 

-1.0  -0.5  0.0  0.5  1.0
Grover's algorithm
Assume: unique \( s \in \{0, 1\}^n \) has \( f(s) = 0 \).

Traditional algorithm to find \( s \):
compute \( f \) for many inputs,
hope to find output 0.
Success probability is very low until \( \#\text{inputs} \) approaches \( 2^n \).

Grover's algorithm takes only \( 2^{n/2} \) evaluations of \( f \).
This is fast.

Typically: reversibility overhead is small enough that this easily wins for all sufficiently large \( n \).

Start from uniform superposition \( a \) over \( q \in \{0, 1\}^n \): \( a_q = 2^{-n/2} \).

Step 1: Set \( a \leftarrow b \) where
\( b_q = -a_q \) if \( f(q) = 0 \),
\( b_q = a_q \) otherwise.
This is fast.

Step 2: “Grover diffusion”.
Negate \( a \) around its average.
This is also fast.

Repeat Step 1 + Step 2 about \( 0.58 \cdot 2^{n/2} \) times.

Measure the \( n \) qubits.
With high probability this finds \( s \).
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.
Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after 0 steps:
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

\[
\begin{align*}
    b_q &= -a_q \text{ if } f(q) = 0, \\
    b_q &= a_q \text{ otherwise.}
\end{align*}
\]

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after 0 steps:
Start from uniform superposition \( a \) over \( q \in \{0, 1\}^n \): \( a_q = 2^{-n/2} \).

Step 1: Set \( a \leftarrow b \) where
\[
b_q = -a_q \text{ if } f(q) = 0,
\]
\[
b_q = a_q \text{ otherwise}.
\]
This is fast.

Step 2: “Grover diffusion”.
Negate \( a \) around its average.
This is also fast.

Repeat Step 1 + Step 2 about \( 0.58 \cdot 2^{n/2} \) times.

Measure the \( n \) qubits.
With high probability this finds \( s \).
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

\[
 b_q = -a_q \text{ if } f(q) = 0,
 b_q = a_q \text{ otherwise.}
\]

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.

With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after Step 1 + Step 2:
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

$$b_q = -a_q \text{ if } f(q) = 0,$$

$$b_q = a_q \text{ otherwise.}$$

This is fast.

Step 2: “Grover diffusion”. Negate $a$ around its average. This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits. With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after Step 1 + Step 2 + Step 1:
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

\[ b_q = -a_q \text{ if } f(q) = 0, \]
\[ b_q = a_q \text{ otherwise.} \]

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $2 \times (\text{Step 1} + \text{Step 2})$:
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.

With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $3 \times (\text{Step 1} + \text{Step 2})$: 

![Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $3 \times (\text{Step 1} + \text{Step 2})$]
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where
$$b_q = -a_q \text{ if } f(q) = 0,$$
$$b_q = a_q \text{ otherwise.}$$
This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $4 \times (\text{Step 1 + Step 2})$:
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

\[
b_q = -a_q \text{ if } f(q) = 0, \quad b_q = a_q \text{ otherwise}.
\]

This is fast.

Step 2: “Grover diffusion”. Negate $a$ around its average. This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits. With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $5 \times (\text{Step 1 + Step 2})$: 

\[
\begin{array}{c}
-1.0 \\
-0.5 \\
0.0 \\
0.5 \\
1.0
\end{array}
\]
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

\[ b_q = -a_q \text{ if } f(q) = 0, \]
\[ b_q = a_q \text{ otherwise.} \]

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $6 \times (\text{Step 1} + \text{Step 2}):$
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

\[
b_q = -a_q \text{ if } f(q) = 0, \\
b_q = a_q \text{ otherwise.}
\]

This is fast.

Step 2: “Grover diffusion”. Negate $a$ around its average. This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits. With high probability this finds $s$.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $7 \times (\text{Step 1} + \text{Step 2})$: 

\[
\begin{array}{c}
-1.0 \\
-0.5 \\
0.0 \\
0.5 \\
1.0 \\
\end{array}
\]
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

\[ b_q = -a_q \text{ if } f(q) = 0, \]
\[ b_q = a_q \text{ otherwise.} \]

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $8 \times (\text{Step 1} + \text{Step 2})$:
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where
\[ b_q = -a_q \text{ if } f(q) = 0, \]
\[ b_q = a_q \text{ otherwise.} \]
This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $9 \times (\text{Step 1 + Step 2})$: 

![Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $9 \times (\text{Step 1 + Step 2})$.]
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

$$b_q = -a_q \text{ if } f(q) = 0,$$
$$b_q = a_q \text{ otherwise.}$$

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.

With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $10 \times (\text{Step 1} + \text{Step 2})$: 

![Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $10 \times (\text{Step 1} + \text{Step 2})$]
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

$$b_q = -a_q$$ if $f(q) = 0$, $$b_q = a_q$$ otherwise.
This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $11 \times (\text{Step 1} + \text{Step 2})$: 
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $12 \times (\text{Step 1} + \text{Step 2})$: 
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $13 \times (\text{Step 1} + \text{Step 2})$: 

![Normalized graph of $q \mapsto a_q$](image)
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $14 \times (\text{Step 1} + \text{Step 2})$: 
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where
$$b_q = -a_q \text{ if } f(q) = 0,$$
$$b_q = a_q \text{ otherwise}.$$  
This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$.  

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $15 \times (\text{Step 1} + \text{Step 2}):$
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

$b_q = -a_q$ if $f(q) = 0$,

$b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.

With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $16 \times (\text{Step 1} + \text{Step 2})$: 

\[ 
\begin{array}{c|c|c|c|c}
 & -1.0 & -0.5 & 0.0 & 0.5 & 1.0 \\
\hline
-1.0 & & & & & \\
-0.5 & & & & & \\
0.0 & & & & & \\
0.5 & & & & & \\
1.0 & & & & & \\
\end{array} 
\]
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

$$b_q = -a_q \text{ if } f(q) = 0,$$
$$b_q = a_q \text{ otherwise.}$$

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.

With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $17 \times (\text{Step 1 + Step 2})$: 

![Graph](image-url)
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where
- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.
This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $18 \times (\text{Step 1} + \text{Step 2})$: 

\[\begin{array}{c}
-1.0 \\
-0.5 \\
0.0 \\
0.5 \\
1.0 \\
\end{array}\]
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $19 \times (\text{Step 1 } + \text{Step 2})$: 

![Normalized graph](image)
Start from uniform superposition \(a\) over \(q \in \{0, 1\}^n\): \(a_q = 2^{-n/2}\).

Step 1: Set \(a \leftarrow b\) where
\[b_q = -a_q \text{ if } f(q) = 0,\]
\[b_q = a_q \text{ otherwise}.\]
This is fast.

Step 2: “Grover diffusion”. Negate \(a\) around its average. This is also fast.

Repeat Step 1 + Step 2 about \(0.58 \cdot 2^{n/2}\) times.

Measure the \(n\) qubits. With high probability this finds \(s\).
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $25 \times (\text{Step 1} + \text{Step 2})$: 

![Normalized graph of $q \mapsto a_q$](image)
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where $b_q = -a_q$ if $f(q) = 0$, $b_q = a_q$ otherwise.
This is fast.

Step 2: “Grover diffusion”. Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$
after $30 \times (\text{Step 1} + \text{Step 2}):$
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $35 \times (\text{Step 1} + \text{Step 2})$:

Good moment to stop, measure.
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $40 \times (\text{Step 1} + \text{Step 2})$: 

![Normalized graph](image-url)
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$.
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

\[ b_q = -a_q \text{ if } f(q) = 0, \]
\[ b_q = a_q \text{ otherwise.} \]

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $50 \times (\text{Step 1 + Step 2})$: 

Traditiona stopping point.
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.
Negate $a$ around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $60 \times (\text{Step 1} + \text{Step 2})$: 

![Normalized graph of q ↦→ a_q](image)
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

$$b_q = -a_q \text{ if } f(q) = 0, \quad b_q = a_q \text{ otherwise.}$$

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.

With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $70 \times (\text{Step 1 + Step 2})$: 

![Normalized graph](image)
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

$b_q = -a_q$ if $f(q) = 0$,

$b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.

With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $80 \times (\text{Step 1} + \text{Step 2}):$
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

$b_q = -a_q$ if $f(q) = 0$,

$b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.

With high probability this finds $s$. 

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $90 \times (\text{Step 1} + \text{Step 2})$: 

\begin{center}
\begin{tikzpicture}
\end{tikzpicture}
\end{center}
Start from uniform superposition $a$ over $q \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where

- $b_q = -a_q$ if $f(q) = 0$,
- $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.

With high probability this finds $s$.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $100 \times (\text{Step 1} + \text{Step 2})$:

Very bad stopping point.
Start from uniform superposition $a \in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where
\[ b_q = -a_q \text{ if } f(q) = 0, \]
\[ b_q = a_q \text{ otherwise.} \]
This is fast.

“Grover diffusion”.

Negate $a$ around its average.
This is also fast.

Step 1 + Step 2 about $0: 58 \cdot 2^{n/2}$ times.

Measure the $n$ qubits.
With high probability this finds $s$.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $100 \times (\text{Step 1 + Step 2})$:

$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):
(1) $a_q$ for roots $q$;
(2) $a_q$ for non-roots $q$. Very bad stopping point.
Start from uniform superposition $a$ over $q \in \{0, 1\}$: $a_q = 2^{-n/2}$.

Step 1: Set $a \leftarrow b$ where $b_q = -a_q$ if $f(q) = 0$, $b_q = a_q$ otherwise.

This is fast.

Step 2: “Grover diffusion”.

Negate $a$ around its average.

This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^n = 2$ times.

Measure the $n$ qubits.

With high probability this finds $s$.
Start from uniform superposition $a$ over $q \in \{0, 1\}$:

$$a_q = 2^{-n} = 2^{-\frac{n}{2}}.$$ 

Step 1: Set $a \leftarrow b$ where

$$b_q = -a_q$$

if $f(q) = 0$,

$$b_q = a_q$$

otherwise.

This is fast.

Step 2: "Grover diffusion". Negate $a$ around its average. This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^n$ times.

Measure the $n$ qubits. With high probability this finds $s$.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $100 \times (\text{Step 1 + Step 2})$:

$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):

1. $a_q$ for roots $q$;
2. $a_q$ for non-roots $q$.

Very bad stopping point.
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $100 \times (\text{Step 1} + \text{Step 2})$:

Very bad stopping point.

$q \mapsto a_q$ is completely described by a vector of two numbers
(with fixed multiplicities):
(1) $a_q$ for roots $q$;
(2) $a_q$ for non-roots $q$. 
Normalized graph of \( q \mapsto a_q \) for an example with \( n = 12 \) after \( 100 \times (\text{Step 1 + Step 2}) \):

Very bad stopping point.

\( q \mapsto a_q \) is completely described by a vector of two numbers (with fixed multiplicities):

1. \( a_q \) for roots \( q \);
2. \( a_q \) for non-roots \( q \).

Step 1 + Step 2 act linearly on this vector.
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $100 \times (\text{Step 1} + \text{Step 2})$:

Very bad stopping point.

$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):
(1) $a_q$ for roots $q$;
(2) $a_q$ for non-roots $q$.

Step 1 + Step 2 act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover’s algorithm.

$\Rightarrow$ Probability is $\approx 1$ after $\approx (\pi/4)2^{n/2}$ iterations.
Normalized graph of $q \mapsto a_q$

for an example with $n = 12$

after $100 \times (\text{Step 1} + \text{Step 2})$:

Very bad stopping point.

$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):

1. $a_q$ for roots $q$;
2. $a_q$ for non-roots $q$.

Step 1 + Step 2 act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover's algorithm.

$\Rightarrow$ Probability is $\approx 1$
after $\approx (\pi/4)2^{n/2}$ iterations.

Ambainis's algorithm

Unique-collision-finding problem:

Say $f$ has $n$-bit inputs,

exactly one collision ${p; q}$:

i.e., $p \neq q$, $f(p) = f(q)$.

Problem: find this collision.
Normalized graph of \( q \mapsto a_q \) for an example with \( n = 12 \) after \( 100 \times (\text{Step 1} + \text{Step 2}) \):

Very bad stopping point.

\( q \mapsto a_q \) is completely described by a vector of two numbers (with fixed multiplicities):

1. \( a_q \) for roots \( q \);
2. \( a_q \) for non-roots \( q \).

Step 1 + Step 2 act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover’s algorithm.

\( \Rightarrow \) Probability is \( \approx 1 \) after \( \approx (\pi/4)2^{n/2} \) iterations.

Ambainis’s algorithm

Unique-collision-finding problem:

Say \( f \) has \( n \)-bit inputs, exactly one collision \( \{ p; q \} \):

i.e., \( p \neq q \), \( f(p) = f(q) \).

Problem: find this collision.
$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):
(1) $a_q$ for roots $q$;
(2) $a_q$ for non-roots $q$.

Step 1 + Step 2
act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover’s algorithm.
$\Rightarrow$ Probability is $\approx 1$ after $\approx (\pi/4)2^{n/2}$ iterations.

Ambainis’s algorithm

Unique-collision-finding problem:
Say $f$ has $n$-bit inputs, exactly one collision $\{p, q\}$: i.e., $p \neq q$, $f(p) = f(q)$.
Problem: find this collision.
$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):

1. $a_q$ for roots $q$;
2. $a_q$ for non-roots $q$.

Step 1 + Step 2 act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover’s algorithm.

$\Rightarrow$ Probability is $\approx 1$ after $\approx (\pi/4)2^{n/2}$ iterations.

Ambainis’s algorithm

Unique-collision-finding problem:
Say $f$ has $n$-bit inputs, exactly one collision $\{p, q\}$:
i.e., $p \neq q$, $f(p) = f(q)$.
Problem: find this collision.
\( q \mapsto a_q \) is completely described by a vector of two numbers (with fixed multiplicities):

1. \( a_q \) for roots \( q \);
2. \( a_q \) for non-roots \( q \).

Step 1 + Step 2

act linearly on this vector.

Easily compute eigenvalues
and powers of this linear map
to understand evolution
of state of Grover’s algorithm.

\[ \Rightarrow \text{Probability is } \approx 1 \]

after \( \approx (\pi/4)2^{n/2} \) iterations.

**Ambainis’s algorithm**

Unique-collision-finding problem:

Say \( f \) has \( n \)-bit inputs, exactly one collision \( \{ p, q \} \):

i.e., \( p \neq q, f(p) = f(q) \).

Problem: find this collision.

Cost \( 2^n \): Define \( S \) as the set of \( n \)-bit strings.
Compute \( f(S) \), sort.
$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):

1. $a_q$ for roots $q$;
2. $a_q$ for non-roots $q$.

Step 1 + Step 2 act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover’s algorithm.

⇒ Probability is $\approx 1$ after $\approx (\frac{\pi}{4})2^{n/2}$ iterations.

Ambainis’s algorithm

Unique-collision-finding problem:

Say $f$ has $n$-bit inputs, exactly one collision $\{p, q\}$:

i.e., $p \neq q$, $f(p) = f(q)$.

Problem: find this collision.

Cost $2^n$: Define $S$ as the set of $n$-bit strings.

Compute $f(S)$, sort.

Generalize to cost $r$,

success probability $\approx (r/2^n)^2$:

Choose a set $S$ of size $r$.

Compute $f(S)$, sort.
Ambainis’s algorithm

Unique-collision-finding problem:
Say \( f \) has \( n \)-bit inputs, exactly one collision \( \{p, q\} \):
i.e., \( p \neq q, f(p) = f(q) \).
Problem: find this collision.

Cost \( 2^n \): Define \( S \) as the set of \( n \)-bit strings.
Compute \( f(S) \), sort.

Generalize to cost \( r \), success probability \( \approx \left(\frac{r}{2^n}\right)^2 \):
Choose a set \( S \) of size \( r \).
Compute \( f(S) \), sort.

Data structure \( D(S) \) capturing the generalized computation:
the set \( S \); the multiset \( f(S) \);
the number of collisions in \( S \).
8

\[ q \mapsto a_q \] is completely described by a vector of two numbers (with fixed multiplicities):

1. \( a_q \) for roots \( q \);
2. \( a_q \) for non-roots \( q \).

Step 1 + Step 2 act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover's algorithm.

\[ \Rightarrow \text{Probability is } \approx 1 \text{ after } \approx (\frac{1}{4})^2 \text{ iterations.} \]

9

Ambainis's algorithm

Unique-collision-finding problem:

Say \( f \) has \( n \)-bit inputs, exactly one collision \( \{ p, q \} \):
i.e., \( p \neq q, f(p) = f(q) \).

Problem: find this collision.

Cost \( 2^n \): Define \( S \) as the set of \( n \)-bit strings.
Compute \( f(S) \), sort.

Generalize to cost \( r \), success probability \( \approx \left(\frac{r}{2^n}\right)^2 \):
Choose a set \( S \) of size \( r \).
Compute \( f(S) \), sort.

Data structure \( D(S) \) capturing the generalized computation:
the set \( S \); the multiset \( f(S) \); the number of collisions in \( S \).
Ambainis’s algorithm

Unique-collision-finding problem:
Say $f$ has $n$-bit inputs,
exactly one collision \{p, q\}:
i.e., \( p \neq q, f(p) = f(q) \).
Problem: find this collision.

Cost $2^n$: Define $S$ as
the set of $n$-bit strings.
Compute $f(S)$, sort.

Generalize to cost $r$,
success probability $\approx (r/2^n)^2$:
Choose a set $S$ of size $r$.
Compute $f(S)$, sort.

Data structure $D(S)$ capturing
the generalized computation:
the set $S$; the multiset $f(S)$;
the number of collisions in $S$. 
Ambainis’s algorithm

Unique-collision-finding problem:
Say \( f \) has \( n \)-bit inputs, exactly one collision \( \{ p, q \} \):
i.e., \( p \neq q, f(p) = f(q) \).
Problem: find this collision.

Cost \( 2^n \): Define \( S \) as the set of \( n \)-bit strings.
Compute \( f(S) \), sort.

Generalize to cost \( r \),
success probability \( \approx (r/2^n)^2 \):
Choose a set \( S \) of size \( r \).
Compute \( f(S) \), sort.

Data structure \( D(S) \) capturing the generalized computation:
the set \( S \); the multiset \( f(S) \);
the number of collisions in \( S \).
Ambainis’s algorithm

Unique-collision-finding problem:
Say $f$ has $n$-bit inputs,
exactly one collision $\{p,q\}$:
i.e., $p \neq q$, $f(p) = f(q)$.
Problem: find this collision.

Cost $2^n$: Define $S$ as
the set of $n$-bit strings.
Compute $f(S)$, sort.

Generalize to cost $r$,
success probability $\approx (r/2^n)^2$:
Choose a set $S$ of size $r$.
Compute $f(S)$, sort.

Data structure $D(S)$ capturing
the generalized computation:
the set $S$; the multiset $f(S)$;
the number of collisions in $S$.

Very efficient to move from $D(S)$
to $D(T)$ if $T$ is an adjacent set:
$\#S = \#T = r$, $\#(S \cap T) = r - 1$. 
Ambainis’s algorithm

Unique-collision-finding problem:
Say $f$ has $n$-bit inputs,
exactly one collision \{p, q\}:
i.e., $p \neq q$, $f(p) = f(q)$.
Problem: find this collision.

Cost $2^n$: Define $S$ as
the set of $n$-bit strings.
Compute $f(S)$, sort.

Generalize to cost $r$,
success probability $\approx (r/2^n)^2$:
Choose a set $S$ of size $r$.
Compute $f(S)$, sort.

Data structure $D(S)$ capturing
the generalized computation:
the set $S$; the multiset $f(S)$;
the number of collisions in $S$.

Very efficient to move from $D(S)$
to $D(T)$ if $T$ is an adjacent set:
$\#S = \#T = r$, $\#(S \cap T) = r - 1$.

2003 Ambainis, simplified 2007
Magniez–Nayak–Roland–Santha:
Create superposition of states
($D(S), D(T)$) with adjacent $S, T$.
By a quantum walk
find $S$ containing a collision.
Ambainis’s algorithm

Unique-collision-finding problem:
Say \( f \) has \( n \)-bit inputs,
exactly one collision \( \{ p, q \} \):
\[ p \neq q, \quad f(p) = f(q). \]
Problem: find this collision.

Cost \( 2^n \):
Define \( S \) as the set of \( n \)-bit strings.
Compute \( f(S) \), sort.

Generalize to cost \( r \),
success probability \( \approx (r=2^n)^2 \):
Choose a set \( S \) of size \( r \).
Compute \( f(S) \), sort.

Data structure \( D(S) \) capturing
the generalized computation:
the set \( S \); the multiset \( f(S) \);
the number of collisions in \( S \).

Very efficient to move from \( D(S) \)
to \( D(T) \) if \( T \) is an adjacent set:
\[ \#S = \#T = r, \quad (S \cap T) = r - 1. \]

2003 Ambainis, simplified 2007
Magniez–Nayak–Roland–Santha:
Create superposition of states
\( (D(S), D(T)) \) with adjacent \( S, T \).
By a quantum walk
find \( S \) containing a collision.

How the quantum walk works:
Start from uniform superposition.
Repeat \( \approx 6 \cdot 2^n \) times:
Negate a \( S; T \) if \( S \) contains collision.
Repeat \( \approx 7 \cdot \sqrt{r} \) times:
For each \( T \):
Diffuse a \( S; T \) across all \( S \).
For each \( S \):
Diffuse a \( S; T \) across all \( T \).

Now high probability
that \( T \) contains collision.
Cost \( r + 2^n = \sqrt{r} \). Optimize: \( 2^{2n} = 3 \).
Ambainis's algorithm

Unique-collision-finding problem:
Say \( f \) has \( n \)-bit inputs, exactly one collision \( \{ p, q \} \):
\( p \neq q \), \( f(p) = f(q) \).
Problem: find this collision.

Cost \( 2^n \): Define \( S \) as the set of \( n \)-bit strings.
Compute \( f(S) \), sort.

Generalize to cost \( r \), success probability \( \approx \left( \frac{r}{2^n} \right)^2 \): Choose a set \( S \) of size \( r \).
Compute \( f(S) \), sort.

Data structure \( D(S) \) capturing the generalized computation:
the set \( S \); the multiset \( f(S) \);
the number of collisions in \( S \).

Very efficient to move from \( D(S) \) to \( D(T) \) if \( T \) is an adjacent set:
\( \#S = \#T = r \), \( (S \cap T) = r - 1 \).

2003 Ambainis, simplified 2007 Magniez–Nayak–Roland–Santha:
Create superposition of states \( (D(S), D(T)) \) with adjacent \( S, T \).
By a quantum walk
find \( S \) containing a collision.

How the quantum walk works:
Start from uniform superposition.
Repeat \( \approx 0.6 \cdot 2^n / \sqrt{r} \) times:
Negate \( a_{S, T} \) if \( S \) contains collision.
Repeat \( \approx 0.7 \cdot \sqrt{r} \) times:
For each \( T \):
Diffuse \( a_{S, T} \) across all \( S \).
For each \( S \):
Diffuse \( a_{S, T} \) across all \( T \).

Now high probability
that \( T \) contains collision.
Cost \( r + 2^n / \sqrt{r} \).
9

Problem:

Data structure $D(S)$ capturing the generalized computation:
the set $S$; the multiset $f(S)$; the number of collisions in $S$.

Very efficient to move from $D(S)$ to $D(T)$ if $T$ is an adjacent set:
$\#S = \#T = r$, $\#(S \cap T) = r - 1$.

2003 Ambainis, simplified 2007
Magniez–Nayak–Roland–Santha:
Create superposition of states $(D(S), D(T))$ with adjacent $S, T$.
By a quantum walk find $S$ containing a collision.

10

How the quantum walk works:

Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n / r$ times:

Negate $a_{S,T}$ if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:

For each $T$:

Diffuse $a_{S,T}$ across all $S$.

For each $S$:

Diffuse $a_{S,T}$ across all $T$.

Now high probability that $T$ contains collision.
Cost $r + 2^n / \sqrt{r}$. Optimize:
Data structure $D(S)$ capturing the generalized computation: the set $S$; the multiset $f(S)$; the number of collisions in $S$.

Very efficient to move from $D(S)$ to $D(T)$ if $T$ is an adjacent set: $\#S = \#T = r$, $\#(S \cap T) = r - 1$.


By a quantum walk find $S$ containing a collision.

How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n/r$ times:
   Negate $a_{S,T}$
      if $S$ contains collision.
   Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
      For each $T$:
         Diffuse $a_{S,T}$ across all $S$.
      For each $S$:
         Diffuse $a_{S,T}$ across all $T$.

Now high probability that $T$ contains collision.
Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$. 
Data structure \( D(S) \) capturing generalized computation: the set \( S \); the multiset \( f(S) \); the number of collisions in \( S \).

Very efficient to move from \( D(S) \) to \( D(T) \) if \( T \) is an adjacent set: 

\[ \#T = r, \#(S \cap T) = r - 1. \]

2003 Ambainis, simplified 2007 Magniez–Nayak–Roland–Santha: Create superposition of states \( (D(S); D(T)) \) with adjacent \( S; T \). By a quantum walk find \( S \) containing a collision.

How the quantum walk works:

Start from uniform superposition.

Repeat \( \approx 0.6 \cdot 2^{n/r} \) times:

- Negate \( a_{S,T} \) if \( S \) contains collision.

Repeat \( \approx 0.7 \cdot \sqrt{r} \) times:

- For each \( T \):
  - Diffuse \( a_{S,T} \) across all \( S \).
- For each \( S \):
  - Diffuse \( a_{S,T} \) across all \( T \).

Now high probability that \( T \) contains collision.

Cost \( r + 2^n / \sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S; T)\) according to \((\#(S \cap \{p;q\}); \#(T \cap \{p;q\}))\); reduce \( a \) to low-dim vector.

Analyze evolution of this vector.

e.g. \( n = 15, r = 1024, \) after 0 negations and 0 diffusions:

\[
\begin{align*}
\Pr[\text{class } (0; 0)] & \approx 0.938; + \\
\Pr[\text{class } (0; 1)] & \approx 0.000; + \\
\Pr[\text{class } (1; 0)] & \approx 0.000; + \\
\Pr[\text{class } (1; 1)] & \approx 0.060; + \\
\Pr[\text{class } (1; 2)] & \approx 0.000; + \\
\Pr[\text{class } (2; 1)] & \approx 0.000; + \\
\Pr[\text{class } (2; 2)] & \approx 0.001; + \\
\end{align*}
\]

Right column is sign of \( a_{S,T} \).
Data structure D(\(S\)) capturing the generalized computation: the set \(S\); the multiset \(f(S)\); the number of collisions in \(S\).

How the quantum walk works:
Start from uniform superposition. Repeat \(\approx 0.6 \cdot 2^n/r\) times:
- Negate \(a_{S,T}\) if \(S\) contains collision.
Repeat \(\approx 0.7 \cdot \sqrt{r}\) times:
  - For each \(T\):
    - Diffuse \(a_{S,T}\) across all \(S\).
  - For each \(S\):
    - Diffuse \(a_{S,T}\) across all \(T\).

Now high probability that \(T\) contains collision.
Cost \(r + 2^n/\sqrt{r}\). Optimize: \(2^{2n/3}\).

Classify \((S, T)\) according to \((\#(S \cap \{p, q\}); \#(T \cap \{p, q\}))\); reduce \(a\) to low-dim vector. Analyze evolution of this vector.

e.g. \(n = 15, r = 1024\), after 0 negations and 0 diffusions:
\[
\begin{align*}
Pr[\text{class (0, 0)}] & \approx 0.938; + \\
Pr[\text{class (0, 1)}] & \approx 0.000; + \\
Pr[\text{class (1, 0)}] & \approx 0.000; + \\
Pr[\text{class (1, 1)}] & \approx 0.060; + \\
Pr[\text{class (1, 2)}] & \approx 0.000; + \\
Pr[\text{class (2, 1)}] & \approx 0.000; + \\
Pr[\text{class (2, 2)}] & \approx 0.001; + \\
\end{align*}
\]

Right column is sign of \(a_{S,T}\).
How the quantum walk works:

Start from uniform superposition.

Repeat $\approx 0.6 \cdot 2^n/r$ times:

- Negate $a_{S,T}$ if $S$ contains collision.

Repeat $\approx 0.7 \cdot \sqrt{r}$ times:

For each $T$:

- Diffuse $a_{S,T}$ across all $S$.

For each $S$:

- Diffuse $a_{S,T}$ across all $T$.

Now high probability that $T$ contains collision.

Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.

Analyze evolution of this vector.

E.g. $n = 15$, $r = 1024$, after $0$ negations and $0$ diffusions:

- $\Pr[\text{class } (0, 0)] \approx 0.938$;
- $\Pr[\text{class } (0, 1)] \approx 0.000$;
- $\Pr[\text{class } (1, 0)] \approx 0.000$;
- $\Pr[\text{class } (1, 1)] \approx 0.060$;
- $\Pr[\text{class } (1, 2)] \approx 0.000$;
- $\Pr[\text{class } (2, 1)] \approx 0.000$;
- $\Pr[\text{class } (2, 2)] \approx 0.001$;

Right column is sign of $a_{S,T}$. 

Classify $(S; T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.

Analyze evolution of this vector.

E.g. $n = 15$, $r = 1024$, after $0$ negations and $0$ diffusions:

- $\Pr[\text{class } (0, 0)] \approx 0.938$;
- $\Pr[\text{class } (0, 1)] \approx 0.000$;
- $\Pr[\text{class } (1, 0)] \approx 0.000$;
- $\Pr[\text{class } (1, 1)] \approx 0.060$;
- $\Pr[\text{class } (1, 2)] \approx 0.000$;
- $\Pr[\text{class } (2, 1)] \approx 0.000$;
- $\Pr[\text{class } (2, 2)] \approx 0.001$;

Right column is sign of $a_{S,T}$. 

Classify $(S; T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.

Analyze evolution of this vector.

E.g. $n = 15$, $r = 1024$, after $0$ negations and $0$ diffusions:

- $\Pr[\text{class } (0, 0)] \approx 0.938$;
- $\Pr[\text{class } (0, 1)] \approx 0.000$;
- $\Pr[\text{class } (1, 0)] \approx 0.000$;
- $\Pr[\text{class } (1, 1)] \approx 0.060$;
- $\Pr[\text{class } (1, 2)] \approx 0.000$;
- $\Pr[\text{class } (2, 1)] \approx 0.000$;
- $\Pr[\text{class } (2, 2)] \approx 0.001$;

Right column is sign of $a_{S,T}$. 

Classify $(S; T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.

Analyze evolution of this vector.

E.g. $n = 15$, $r = 1024$, after $0$ negations and $0$ diffusions:

- $\Pr[\text{class } (0, 0)] \approx 0.938$;
- $\Pr[\text{class } (0, 1)] \approx 0.000$;
- $\Pr[\text{class } (1, 0)] \approx 0.000$;
- $\Pr[\text{class } (1, 1)] \approx 0.060$;
- $\Pr[\text{class } (1, 2)] \approx 0.000$;
- $\Pr[\text{class } (2, 1)] \approx 0.000$;
- $\Pr[\text{class } (2, 2)] \approx 0.001$;

Right column is sign of $a_{S,T}$. 

Classify $(S; T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.

Analyze evolution of this vector.

E.g. $n = 15$, $r = 1024$, after $0$ negations and $0$ diffusions:

- $\Pr[\text{class } (0, 0)] \approx 0.938$;
- $\Pr[\text{class } (0, 1)] \approx 0.000$;
- $\Pr[\text{class } (1, 0)] \approx 0.000$;
- $\Pr[\text{class } (1, 1)] \approx 0.060$;
- $\Pr[\text{class } (1, 2)] \approx 0.000$;
- $\Pr[\text{class } (2, 1)] \approx 0.000$;
- $\Pr[\text{class } (2, 2)] \approx 0.001$;

Right column is sign of $a_{S,T}$. 

Classify $(S; T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.

Analyze evolution of this vector.

E.g. $n = 15$, $r = 1024$, after $0$ negations and $0$ diffusions:

- $\Pr[\text{class } (0, 0)] \approx 0.938$;
- $\Pr[\text{class } (0, 1)] \approx 0.000$;
- $\Pr[\text{class } (1, 0)] \approx 0.000$;
- $\Pr[\text{class } (1, 1)] \approx 0.060$;
- $\Pr[\text{class } (1, 2)] \approx 0.000$;
- $\Pr[\text{class } (2, 1)] \approx 0.000$;
- $\Pr[\text{class } (2, 2)] \approx 0.001$;

Right column is sign of $a_{S,T}$.
How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n / r$ times:
  Negate $a_{S,T}$ if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
  For each $T$:
    Diffuse $a_{S,T}$ across all $S$.
  For each $S$:
    Diffuse $a_{S,T}$ across all $T$.
Now high probability that $T$ contains collision.
Cost $r + 2^n / \sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after 0 negations and 0 diffusions:

\[
\begin{array}{l}
\Pr[\text{class (0, 0)}] \approx 0.938; + \\
\Pr[\text{class (0, 1)}] \approx 0.000; + \\
\Pr[\text{class (1, 0)}] \approx 0.000; + \\
\Pr[\text{class (1, 1)}] \approx 0.060; + \\
\Pr[\text{class (1, 2)}] \approx 0.000; + \\
\Pr[\text{class (2, 1)}] \approx 0.000; + \\
\Pr[\text{class (2, 2)}] \approx 0.001; + \\
\end{array}
\]
Right column is sign of $a_{S,T}$.
How the quantum walk works:
Start from uniform superposition. Repeat $\approx 0.6 \cdot 2^n / r$ times:
- Negate $a_{S,T}$ if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
  - For each $T$: Diffuse $a_{S,T}$ across all $S$.
  - For each $S$: Diffuse $a_{S,T}$ across all $T$.
Now high probability that $T$ contains collision.
Cost $r + 2^n / \sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.
Analyze evolution of this vector.
e.g. $n = 15, r = 1024$, after 1 negation and 46 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.935; +$
$\Pr[\text{class } (0, 1)] \approx 0.000; +$
$\Pr[\text{class } (1, 0)] \approx 0.000; −$
$\Pr[\text{class } (1, 1)] \approx 0.057; +$
$\Pr[\text{class } (1, 2)] \approx 0.000; +$
$\Pr[\text{class } (2, 1)] \approx 0.000; −$
$\Pr[\text{class } (2, 2)] \approx 0.008; +$

Right column is sign of $a_{S,T}$. 

Classify $\ldots$
How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n/r$ times:
- Negate $a_{S,T}$ if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
- For each $T$:
  - Diffuse $a_{S,T}$ across all $S$.
- For each $S$:
  - Diffuse $a_{S,T}$ across all $T$.

Now high probability
that $T$ contains collision.
Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to
$(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.

E.g. $n = 15$, $r = 1024$, after
2 negations and 92 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.918; +$
$\Pr[\text{class } (0, 1)] \approx 0.001; +$
$\Pr[\text{class } (1, 0)] \approx 0.000; −$
$\Pr[\text{class } (1, 1)] \approx 0.059; +$
$\Pr[\text{class } (1, 2)] \approx 0.001; +$
$\Pr[\text{class } (2, 1)] \approx 0.000; −$
$\Pr[\text{class } (2, 2)] \approx 0.022; +$

Right column is sign of $a_{S,T}$. 
How the quantum walk works:

Start from uniform superposition.

Repeat \( \approx 0.6 \cdot 2^n / r \) times:
- Negate \( a_{S,T} \) if \( S \) contains collision.

Repeat \( \approx 0.7 \cdot \sqrt{r} \) times:
- For each \( T \):
  - Diffuse \( a_{S,T} \) across all \( S \).
- For each \( S \):
  - Diffuse \( a_{S,T} \) across all \( T \).

Now high probability that \( T \) contains collision.

Cost \( r + 2^n / \sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to \((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \( a \) to low-dim vector.

Analyze evolution of this vector.

E.g. \( n = 15, r = 1024, \) after 3 negations and 138 diffusions:

\[
\begin{align*}
\text{Pr}[\text{class } (0, 0)] & \approx 0.897; + \\
\text{Pr}[\text{class } (0, 1)] & \approx 0.001; + \\
\text{Pr}[\text{class } (1, 0)] & \approx 0.000; - \\
\text{Pr}[\text{class } (1, 1)] & \approx 0.058; + \\
\text{Pr}[\text{class } (1, 2)] & \approx 0.002; + \\
\text{Pr}[\text{class } (2, 1)] & \approx 0.000; + \\
\text{Pr}[\text{class } (2, 2)] & \approx 0.042; + 
\end{align*}
\]

Right column is sign of \( a_{S,T} \).
How the quantum walk works:

Start from uniform superposition.

Repeat $\approx 0.6 \cdot 2^n/r$ times:
- Negate $a_{S,T}$ if $S$ contains collision.

Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
- For each $T$:
  - Diffuse $a_{S,T}$ across all $S$.
- For each $S$:
  - Diffuse $a_{S,T}$ across all $T$.

Now high probability that $T$ contains collision.

Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.

Analyze evolution of this vector.

E.g. $n = 15$, $r = 1024$, after 4 negations and 184 diffusions:

- $\text{Pr}[\text{class (0, 0)}] \approx 0.873; +$
- $\text{Pr}[\text{class (0, 1)}] \approx 0.001; +$
- $\text{Pr}[\text{class (1, 0)}] \approx 0.000; -$  
- $\text{Pr}[\text{class (1, 1)}] \approx 0.054; +$
- $\text{Pr}[\text{class (1, 2)}] \approx 0.002; +$
- $\text{Pr}[\text{class (2, 1)}] \approx 0.000; +$
- $\text{Pr}[\text{class (2, 2)}] \approx 0.070; +$

Right column is sign of $a_{S,T}$.
How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n / r$ times:
  Negate $a_{S,T}$
    if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
  For each $T$:
    Diffuse $a_{S,T}$ across all $S$.
  For each $S$:
    Diffuse $a_{S,T}$ across all $T$.

Now high probability that $T$ contains collision.
Cost $r + 2^n / \sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after 5 negations and 230 diffusions:
\[
\begin{align*}
\Pr[\text{class } (0, 0)] & \approx 0.838; + \\
\Pr[\text{class } (0, 1)] & \approx 0.001; + \\
\Pr[\text{class } (1, 0)] & \approx 0.001; - \\
\Pr[\text{class } (1, 1)] & \approx 0.054; + \\
\Pr[\text{class } (1, 2)] & \approx 0.003; + \\
\Pr[\text{class } (2, 1)] & \approx 0.000; + \\
\Pr[\text{class } (2, 2)] & \approx 0.104; + 
\end{align*}
\]

Right column is sign of $a_{S,T}$. 
How the quantum walk works:

Start from uniform superposition.

Repeat $\approx 0.6 \cdot 2^n / r$ times:
- Negate $a_{S,T}$ if $S$ contains collision.

Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
- For each $T$:
  - Diffuse $a_{S,T}$ across all $S$.
- For each $S$:
  - Diffuse $a_{S,T}$ across all $T$.

Now high probability that $T$ contains collision.

Cost $r + 2^n / \sqrt{r}$. Optimize: $2^{2n/3}$.

---

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;
reduce $a$ to low-dim vector.

Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after 6 negations and 276 diffusions:

Pr[\text{class (0, 0)}] \approx 0.800; +
Pr[\text{class (0, 1)}] \approx 0.001; +
Pr[\text{class (1, 0)}] \approx 0.001; −
Pr[\text{class (1, 1)}] \approx 0.051; +
Pr[\text{class (1, 2)}] \approx 0.006; +
Pr[\text{class (2, 1)}] \approx 0.000; +
Pr[\text{class (2, 2)}] \approx 0.141; +

Right column is sign of $a_{S,T}$. 
How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n/r$ times:
   Negate $a_{S,T}$
      if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
   For each $T$:
      Diffuse $a_{S,T}$ across all $S$.
   For each $S$:
      Diffuse $a_{S,T}$ across all $T$.
Now high probability
that $T$ contains collision.
Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to
$(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after
7 negations and 322 diffusions:

\[
\begin{align*}
\Pr[\text{class } (0, 0)] & \approx 0.758; + \\
\Pr[\text{class } (0, 1)] & \approx 0.002; + \\
\Pr[\text{class } (1, 0)] & \approx 0.001; - \\
\Pr[\text{class } (1, 1)] & \approx 0.047; + \\
\Pr[\text{class } (1, 2)] & \approx 0.007; + \\
\Pr[\text{class } (2, 1)] & \approx 0.000; + \\
\Pr[\text{class } (2, 2)] & \approx 0.184; + 
\end{align*}
\]
Right column is sign of $a_{S,T}$. 

How the quantum walk works:

Start from uniform superposition.

Repeat $\approx 0.6 \cdot 2^n/r$ times:

- Negate $a_{S,T}$ if $S$ contains collision.

Repeat $\approx 0.7 \cdot \sqrt{r}$ times:

- For each $T$:
  - Diffuse $a_{S,T}$ across all $S$.
- For each $S$:
  - Diffuse $a_{S,T}$ across all $T$.

Now high probability that $T$ contains collision.

Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.

Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after 8 negations and 368 diffusions:

$$
\begin{align*}
\Pr[\text{class } (0, 0)] &\approx 0.708; + \\
\Pr[\text{class } (0, 1)] &\approx 0.003; + \\
\Pr[\text{class } (1, 0)] &\approx 0.001; - \\
\Pr[\text{class } (1, 1)] &\approx 0.046; + \\
\Pr[\text{class } (1, 2)] &\approx 0.007; + \\
\Pr[\text{class } (2, 1)] &\approx 0.000; + \\
\Pr[\text{class } (2, 2)] &\approx 0.234; +
\end{align*}
$$

Right column is sign of $a_{S,T}$. 
How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n / r$ times:
- Negate $a_{S,T}$
  - if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
- For each $T$:
  - Diffuse $a_{S,T}$ across all $S$.
- For each $S$:
  - Diffuse $a_{S,T}$ across all $T$.
Now high probability
that $T$ contains collision.
Cost $r + 2^n / \sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to
$\left( \#(S \cap \{p, q\}), \#(T \cap \{p, q\}) \right)$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.
e.g. $n = 15, r = 1024$, after
9 negations and 414 diffusions:

\[
\begin{align*}
\Pr[\text{class (0, 0)}] & \approx 0.658; + \\
\Pr[\text{class (0, 1)}] & \approx 0.003; + \\
\Pr[\text{class (1, 0)}] & \approx 0.001; - \\
\Pr[\text{class (1, 1)}] & \approx 0.042; + \\
\Pr[\text{class (1, 2)}] & \approx 0.009; + \\
\Pr[\text{class (2, 1)}] & \approx 0.000; + \\
\Pr[\text{class (2, 2)}] & \approx 0.287; +
\end{align*}
\]

Right column is sign of $a_{S,T}$. 

How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n / r$ times:
   Negate $a_{S,T}$ if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
   For each $T$:
      Diffuse $a_{S,T}$ across all $S$.
   For each $S$:
      Diffuse $a_{S,T}$ across all $T$.
Now high probability that $T$ contains collision.
Cost $r + 2^n / \sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.
Analyze evolution of this vector.

E.g. $n = 15, r = 1024$, after 10 negations and 460 diffusions:
$$\Pr[\text{class } (0,0)] \approx 0.606; +$$
$$\Pr[\text{class } (0,1)] \approx 0.003; +$$
$$\Pr[\text{class } (1,0)] \approx 0.002; -$$
$$\Pr[\text{class } (1,1)] \approx 0.037; +$$
$$\Pr[\text{class } (1,2)] \approx 0.013; +$$
$$\Pr[\text{class } (2,1)] \approx 0.000; +$$
$$\Pr[\text{class } (2,2)] \approx 0.338; +$$
Right column is sign of $a_{S,T}$.
How the quantum walk works:
Start from uniform superposition.
Repeat \( \approx 0.6 \cdot 2^n / r \) times:
  Negate \( a_{S,T} \) if \( S \) contains collision.
Repeat \( \approx 0.7 \cdot \sqrt{r} \) times:
  For each \( T \):
    Diffuse \( a_{S,T} \) across all \( S \).
  For each \( S \):
    Diffuse \( a_{S,T} \) across all \( T \).
Now high probability
that \( T \) contains collision.
Cost \( r + 2^n / \sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to \((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \( a \) to low-dim vector.
Analyze evolution of this vector.
e.g. \( n = 15, r = 1024 \), after 11 negations and 506 diffusions:
\[
\begin{align*}
\Pr[\text{class } (0, 0)] & \approx 0.547; + \\
\Pr[\text{class } (0, 1)] & \approx 0.004; + \\
\Pr[\text{class } (1, 0)] & \approx 0.003; - \\
\Pr[\text{class } (1, 1)] & \approx 0.036; + \\
\Pr[\text{class } (1, 2)] & \approx 0.015; + \\
\Pr[\text{class } (2, 1)] & \approx 0.001; + \\
\Pr[\text{class } (2, 2)] & \approx 0.394; + 
\end{align*}
\]
Right column is sign of \( a_{S,T} \).
How the quantum walk works:

Start from uniform superposition.

Repeat $\approx 0.6 \cdot 2^n / r$ times:

Negate $a_{S,T}$

if $S$ contains collision.

Repeat $\approx 0.7 \cdot \sqrt{r}$ times:

For each $T$:

Diffuse $a_{S,T}$ across all $S$.

For each $S$:

Diffuse $a_{S,T}$ across all $T$.

Now high probability

that $T$ contains collision.

Cost $r + 2^n / \sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to

$(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;

reduce $a$ to low-dim vector.

Analyze evolution of this vector.

E.g. $n = 15$, $r = 1024$, after

12 negations and 552 diffusions:

Pr[Class (0, 0)] $\approx 0.491$; +
Pr[Class (0, 1)] $\approx 0.004$; +
Pr[Class (1, 0)] $\approx 0.003$; −
Pr[Class (1, 1)] $\approx 0.032$; +
Pr[Class (1, 2)] $\approx 0.014$; +
Pr[Class (2, 1)] $\approx 0.001$; +
Pr[Class (2, 2)] $\approx 0.455$; +

Right column is sign of $a_{S,T}$. 
How the quantum walk works:
Start from uniform superposition.
Repeat \( \approx 0.6 \cdot 2^n/r \) times:
\[ \text{Negate } a_{S,T} \]
\[ \text{if } S \text{ contains collision.} \]
Repeat \( \approx 0.7 \cdot \sqrt{r} \) times:
\[ \text{For each } T: \]
\[ \text{Diffuse } a_{S,T} \text{ across all } S. \]
\[ \text{For each } S: \]
\[ \text{Diffuse } a_{S,T} \text{ across all } T. \]
Now high probability that \( T \) contains collision.
Cost \( r + 2^n/\sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to \((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \( a \) to low-dim vector.
Analyze evolution of this vector.
e.g. \( n = 15, r = 1024, \) after 13 negations and 598 diffusions:
\[ \Pr[\text{class } (0, 0)] \approx 0.436; + \]
\[ \Pr[\text{class } (0, 1)] \approx 0.005; + \]
\[ \Pr[\text{class } (1, 0)] \approx 0.003; − \]
\[ \Pr[\text{class } (1, 1)] \approx 0.026; + \]
\[ \Pr[\text{class } (1, 2)] \approx 0.017; + \]
\[ \Pr[\text{class } (2, 1)] \approx 0.000; + \]
\[ \Pr[\text{class } (2, 2)] \approx 0.513; + \]
Right column is sign of \( a_{S,T} \).
How the quantum walk works:
Start from uniform superposition.
Repeat \( \approx 0.6 \cdot 2^n / r \) times:
    Negate \( a_{S,T} \)
        if \( S \) contains collision.
Repeat \( \approx 0.7 \cdot \sqrt{r} \) times:
    For each \( T \):
        Diffuse \( a_{S,T} \) across all \( S \).
    For each \( S \):
        Diffuse \( a_{S,T} \) across all \( T \).

Now high probability
that \( T \) contains collision.

Cost \( r + 2^n / \sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to \((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \( a \) to low-dim vector.
Analyze evolution of this vector.

e.g. \( n = 15 \), \( r = 1024 \), after
14 negations and 644 diffusions:

\[
\begin{align*}
\Pr[\text{class } (0, 0)] &\approx 0.377; + \\
\Pr[\text{class } (0, 1)] &\approx 0.006; + \\
\Pr[\text{class } (1, 0)] &\approx 0.004; - \\
\Pr[\text{class } (1, 1)] &\approx 0.025; + \\
\Pr[\text{class } (1, 2)] &\approx 0.022; + \\
\Pr[\text{class } (2, 1)] &\approx 0.001; + \\
\Pr[\text{class } (2, 2)] &\approx 0.566; + \\
\end{align*}
\]

Right column is sign of \( a_{S,T} \).
How the quantum walk works:

Start from uniform superposition.

Repeat $\approx 0.6 \cdot 2^n / r$ times:

Negate $a_{S,T}$

if $S$ contains collision.

Repeat $\approx 0.7 \cdot \sqrt{r}$ times:

For each $T$:

Diffuse $a_{S,T}$ across all $S$.

For each $S$:

Diffuse $a_{S,T}$ across all $T$.

Now high probability

that $T$ contains collision.

Cost $r + 2^n / \sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to

$\left( \#(S \cap \{p, q\}), \#(T \cap \{p, q\}) \right)$;

reduce $a$ to low-dim vector.

Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after

15 negations and 690 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.322$; +
$\Pr[\text{class } (0, 1)] \approx 0.005$; +
$\Pr[\text{class } (1, 0)] \approx 0.004$; −
$\Pr[\text{class } (1, 1)] \approx 0.021$; +
$\Pr[\text{class } (1, 2)] \approx 0.023$; +
$\Pr[\text{class } (2, 1)] \approx 0.001$; +
$\Pr[\text{class } (2, 2)] \approx 0.623$; +

Right column is sign of $a_{S,T}$. 
How the quantum walk works:
Start from uniform superposition.
Repeat ≈ 0.6 \cdot 2^n / r times:
  Negate \( a_{S,T} \)
    if \( S \) contains collision.
Repeat ≈ 0.7 \cdot \sqrt{r} times:
  For each \( T \):
    Diffuse \( a_{S,T} \) across all \( S \).
  For each \( S \):
    Diffuse \( a_{S,T} \) across all \( T \).
Now high probability
that \( T \) contains collision.
Cost \( r + 2^n / \sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to
\((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \( a \) to low-dim vector.
Analyze evolution of this vector.
e.g. \( n = 15 \), \( r = 1024 \), after
16 negations and 736 diffusions:
\[
\begin{align*}
\Pr[\text{class } (0, 0)] &\approx 0.270; + \\
\Pr[\text{class } (0, 1)] &\approx 0.006; + \\
\Pr[\text{class } (1, 0)] &\approx 0.005; − \\
\Pr[\text{class } (1, 1)] &\approx 0.017; + \\
\Pr[\text{class } (1, 2)] &\approx 0.022; + \\
\Pr[\text{class } (2, 1)] &\approx 0.001; + \\
\Pr[\text{class } (2, 2)] &\approx 0.680; + 
\end{align*}
\]
Right column is sign of \( a_{S,T} \).
How the quantum walk works:
Start from uniform superposition.
Repeat \( \approx 0.6 \cdot 2^n / r \) times:
   Negate \( a_{S,T} \) if \( S \) contains collision.
Repeat \( \approx 0.7 \cdot \sqrt{r} \) times:
   For each \( T \):
      Diffuse \( a_{S,T} \) across all \( S \).
   For each \( S \):
      Diffuse \( a_{S,T} \) across all \( T \).
Now high probability that \( T \) contains collision.
Cost \( r + 2^n / \sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to \((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \( a \) to low-dim vector.
Analyze evolution of this vector.
e.g. \( n = 15, r = 1024 \), after
17 negations and 782 diffusions:
\[
\begin{align*}
\Pr[\text{class (0, 0)}] & \approx 0.218; + \\
\Pr[\text{class (0, 1)}] & \approx 0.007; + \\
\Pr[\text{class (1, 0)}] & \approx 0.005; - \\
\Pr[\text{class (1, 1)}] & \approx 0.015; + \\
\Pr[\text{class (1, 2)}] & \approx 0.024; + \\
\Pr[\text{class (2, 1)}] & \approx 0.001; + \\
\Pr[\text{class (2, 2)}] & \approx 0.730; + 
\end{align*}
\]
Right column is sign of \( a_{S,T} \).
How the quantum walk works:

Start from uniform superposition.

Repeat $\approx 0.6 \cdot 2^n/r$ times:

Negate $a_{S,T}$ if $S$ contains collision.

Repeat $\approx 0.7 \cdot \sqrt{r}$ times:

For each $T$:

Diffuse $a_{S,T}$ across all $S$.

For each $S$:

Diffuse $a_{S,T}$ across all $T$.

Now high probability that $T$ contains collision.

Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.

Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after 18 negations and 828 diffusions:

$$
\begin{align*}
Pr[\text{class } (0, 0)] &\approx 0.172; + \\
Pr[\text{class } (0, 1)] &\approx 0.006; + \\
Pr[\text{class } (1, 0)] &\approx 0.005; - \\
Pr[\text{class } (1, 1)] &\approx 0.011; + \\
Pr[\text{class } (1, 2)] &\approx 0.029; + \\
Pr[\text{class } (2, 1)] &\approx 0.001; + \\
Pr[\text{class } (2, 2)] &\approx 0.775; + 
\end{align*}
$$

Right column is sign of $a_{S,T}$. 

How the quantum walk works:

Start from uniform superposition. Repeat \( \approx 0.6 \cdot 2^n/r \) times:

Negate \( a_{S,T} \)

if \( S \) contains collision.

Repeat \( \approx 0.7 \cdot \sqrt{r} \) times:

For each \( T \):

Diffuse \( a_{S,T} \) across all \( S \).

For each \( S \):

Diffuse \( a_{S,T} \) across all \( T \).

Now high probability

that \( T \) contains collision.

Cost \( r + 2^n/\sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to

\((#(S \cap \{p, q\}), #(T \cap \{p, q\}))\);

reduce \( a \) to low-dim vector.

Analyze evolution of this vector.

e.g. \( n = 15, r = 1024 \), after

19 negations and 874 diffusions:

\[
\begin{align*}
\Pr[\text{class } (0, 0)] &\approx 0.131; + \\
\Pr[\text{class } (0, 1)] &\approx 0.007; + \\
\Pr[\text{class } (1, 0)] &\approx 0.006; - \\
\Pr[\text{class } (1, 1)] &\approx 0.008; + \\
\Pr[\text{class } (1, 2)] &\approx 0.030; + \\
\Pr[\text{class } (2, 1)] &\approx 0.002; + \\
\Pr[\text{class } (2, 2)] &\approx 0.816; +
\end{align*}
\]

Right column is sign of \( a_{S,T} \).
How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n/r$ times:
  Negate $a_{S,T}$
    if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
  For each $T$:
    Diffuse $a_{S,T}$ across all $S$.
  For each $S$:
    Diffuse $a_{S,T}$ across all $T$.
Now high probability
that $T$ contains collision.
Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to
$(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.
e.g. $n = 15$, $r = 1024$, after
20 negations and 920 diffusions:
$\Pr[\text{class (0, 0)}] \approx 0.093$; +
$\Pr[\text{class (0, 1)}] \approx 0.007$; +
$\Pr[\text{class (1, 0)}] \approx 0.007$; −
$\Pr[\text{class (1, 1)}] \approx 0.007$; +
$\Pr[\text{class (1, 2)}] \approx 0.027$; +
$\Pr[\text{class (2, 1)}] \approx 0.002$; +
$\Pr[\text{class (2, 2)}] \approx 0.857$; +
Right column is sign of $a_{S,T}$. 
How the quantum walk works:

Start from uniform superposition. Repeat $\approx 0.6 \cdot 2^n / r$ times:

- Negate $a_{S,T}$ if $S$ contains collision.

Repeat $\approx 0.7 \cdot \sqrt{r}$ times:

- For each $T$:
  - Diffuse $a_{S,T}$ across all $S$.

- For each $S$:
  - Diffuse $a_{S,T}$ across all $T$.

Now high probability that $T$ contains collision.

Cost $r + 2^n / \sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector. Analyze evolution of this vector.

E.g. $n = 15$, $r = 1024$, after 21 negations and 966 diffusions:

- $\Pr[\text{class } (0, 0)] \approx 0.062$; +
- $\Pr[\text{class } (0, 1)] \approx 0.007$; +
- $\Pr[\text{class } (1, 0)] \approx 0.006$; −
- $\Pr[\text{class } (1, 1)] \approx 0.004$; +
- $\Pr[\text{class } (1, 2)] \approx 0.030$; +
- $\Pr[\text{class } (2, 1)] \approx 0.001$; +
- $\Pr[\text{class } (2, 2)] \approx 0.890$; +

Right column is sign of $a_{S,T}$.
How the quantum walk works:

Start from uniform superposition.

Repeat \( \approx 0.6 \cdot 2^n / r \) times:
- Negate \( a_S, T \) if \( S \) contains collision.

Repeat \( \approx 0.7 \cdot \sqrt{r} \) times:
- For each \( T \):
  - Diffuse \( a_S, T \) across all \( S \).
- For each \( S \):
  - Diffuse \( a_S, T \) across all \( T \).

Now high probability that \( T \) contains collision.

Cost \( r + 2^n / \sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to \((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \( a \) to low-dim vector.

Analyze evolution of this vector.

e.g. \( n = 15, r = 1024 \), after 22 negations and 1012 diffusions:

\[
\begin{align*}
\Pr[\text{class } (0, 0)] &\approx 0.037; + \\
\Pr[\text{class } (0, 1)] &\approx 0.008; + \\
\Pr[\text{class } (1, 0)] &\approx 0.007; - \\
\Pr[\text{class } (1, 1)] &\approx 0.002; + \\
\Pr[\text{class } (1, 2)] &\approx 0.034; + \\
\Pr[\text{class } (2, 1)] &\approx 0.001; + \\
\Pr[\text{class } (2, 2)] &\approx 0.910; + 
\end{align*}
\]

Right column is sign of \( a_S, T \).
How the quantum walk works:
Start from uniform superposition.
Repeat \( \approx 0.6 \cdot 2^n / r \) times:
    Negate \( a_{S,T} \)
    if \( S \) contains collision.
Repeat \( \approx 0.7 \cdot \sqrt{r} \) times:
    For each \( T \):
        Diffuse \( a_{S,T} \) across all \( S \).
    For each \( S \):
        Diffuse \( a_{S,T} \) across all \( T \).
Now high probability
that \( T \) contains collision.
Cost \( r + 2^n / \sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to
\((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \( a \) to low-dim vector.
Analyze evolution of this vector.
e.g. \( n = 15, r = 1024 \), after
23 negations and 1058 diffusions:

\[
\begin{align*}
\Pr[\text{class } (0, 0)] & \approx 0.017; + \\
\Pr[\text{class } (0, 1)] & \approx 0.008; + \\
\Pr[\text{class } (1, 0)] & \approx 0.007; - \\
\Pr[\text{class } (1, 1)] & \approx 0.002; + \\
\Pr[\text{class } (1, 2)] & \approx 0.034; + \\
\Pr[\text{class } (2, 1)] & \approx 0.002; + \\
\Pr[\text{class } (2, 2)] & \approx 0.930; + 
\end{align*}
\]
Right column is sign of \( a_{S,T} \).
How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n/r$ times:
Negate $a_{S,T}$
if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
For each $T$:
Diffuse $a_{S,T}$ across all $S$.
For each $S$:
Diffuse $a_{S,T}$ across all $T$.
Now high probability that $T$ contains collision.
Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.
E.g. $n = 15$, $r = 1024$, after 24 negations and 1104 diffusions:

\[
\begin{align*}
\Pr[\text{class } (0, 0)] &\approx 0.005; + \\
\Pr[\text{class } (0, 1)] &\approx 0.007; + \\
\Pr[\text{class } (1, 0)] &\approx 0.007; - \\
\Pr[\text{class } (1, 1)] &\approx 0.000; + \\
\Pr[\text{class } (1, 2)] &\approx 0.030; + \\
\Pr[\text{class } (2, 1)] &\approx 0.002; + \\
\Pr[\text{class } (2, 2)] &\approx 0.948; + 
\end{align*}
\]
Right column is sign of $a_{S,T}$. 

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.
E.g. $n = 15$, $r = 1024$, after 24 negations and 1104 diffusions:

\[
\begin{align*}
\Pr[\text{class } (0, 0)] &\approx 0.005; + \\
\Pr[\text{class } (0, 1)] &\approx 0.007; + \\
\Pr[\text{class } (1, 0)] &\approx 0.007; - \\
\Pr[\text{class } (1, 1)] &\approx 0.000; + \\
\Pr[\text{class } (1, 2)] &\approx 0.030; + \\
\Pr[\text{class } (2, 1)] &\approx 0.002; + \\
\Pr[\text{class } (2, 2)] &\approx 0.948; + 
\end{align*}
\]
Right column is sign of $a_{S,T}$.
How the quantum walk works:

Start from uniform superposition.

Repeat $\approx 0.6 \cdot 2^n/r$ times:
- Negate $a_{S,T}$ if $S$ contains collision.

Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
- For each $T$:
  - Diffuse $a_{S,T}$ across all $S$.
- For each $S$:
  - Diffuse $a_{S,T}$ across all $T$.

Now high probability that $T$ contains collision.

Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to

$(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;

reduce $a$ to low-dim vector.

Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after 25 negations and 1150 diffusions:

$Pr[class (0, 0)] \approx 0.000; +$
$Pr[class (0, 1)] \approx 0.008; +$
$Pr[class (1, 0)] \approx 0.008; −$
$Pr[class (1, 1)] \approx 0.000; +$
$Pr[class (1, 2)] \approx 0.031; +$
$Pr[class (2, 1)] \approx 0.001; +$
$Pr[class (2, 2)] \approx 0.952; +$

Right column is sign of $a_{S,T}$. 
How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n/r$ times:
  Negate $a_{S,T}$
    if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
  For each $T$:
    Diffuse $a_{S,T}$ across all $S$.
  For each $S$:
    Diffuse $a_{S,T}$ across all $T$.
Now high probability
that $T$ contains collision.
Cost $r + 2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to
$(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.
e.g. $n = 15, r = 1024$, after
26 negations and 1196 diffusions:

\[
\begin{align*}
\Pr[\text{class } (0, 0)] &\approx 0.002; - \\
\Pr[\text{class } (0, 1)] &\approx 0.008; + \\
\Pr[\text{class } (1, 0)] &\approx 0.008; - \\
\Pr[\text{class } (1, 1)] &\approx 0.000; - \\
\Pr[\text{class } (1, 2)] &\approx 0.035; + \\
\Pr[\text{class } (2, 1)] &\approx 0.002; + \\
\Pr[\text{class } (2, 2)] &\approx 0.945; + 
\end{align*}
\]
Right column is sign of $a_{S,T}$. 
How the quantum walk works:
Start from uniform superposition.
Repeat $\approx 0.6 \cdot 2^n / r$ times:
  Negate $a_{S, T}$ if $S$ contains collision.
Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
  For each $T$:
    Diffuse $a_{S, T}$ across all $S$.
  For each $S$:
    Diffuse $a_{S, T}$ across all $T$.
Now high probability that $T$ contains collision.
Cost $r + 2^n / \sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$; reduce $a$ to low-dim vector.
Analyze evolution of this vector.
e.g. $n = 15$, $r = 1024$, after 27 negations and 1242 diffusions:

- $Pr[\text{class } (0, 0)] \approx 0.011$;
- $Pr[\text{class } (0, 1)] \approx 0.007$;
- $Pr[\text{class } (1, 0)] \approx 0.007$;
- $Pr[\text{class } (1, 1)] \approx 0.001$;
- $Pr[\text{class } (1, 2)] \approx 0.034$;
- $Pr[\text{class } (2, 1)] \approx 0.003$;
- $Pr[\text{class } (2, 2)] \approx 0.938$;

Right column is sign of $a_{S, T}$. 
How the quantum walk works:

1. Start from uniform superposition.
2. Repeat $\approx 0.6 \cdot 2^n/r$ times:
   - Negate $a_{S,T}$ if $S$ contains collision.
3. Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
   - For each $T$: Diffuse $a_{S,T}$ across all $S$.
   - For each $S$: Diffuse $a_{S,T}$ across all $T$.
   - With probability $2^n/\sqrt{r}$. Optimize: $2^{2n/3}$.

Classify $(S, T)$ according to $(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after 27 negations and 1242 diffusions:

- $\Pr[\text{class } (0, 0)] \approx 0.011$; $-$
- $\Pr[\text{class } (0, 1)] \approx 0.007$; $+$
- $\Pr[\text{class } (1, 0)] \approx 0.007$; $-$
- $\Pr[\text{class } (1, 1)] \approx 0.001$; $-$
- $\Pr[\text{class } (1, 2)] \approx 0.034$; $+$
- $\Pr[\text{class } (2, 1)] \approx 0.003$; $+$
- $\Pr[\text{class } (2, 2)] \approx 0.938$; $+$

Right column is sign of $a_{S,T}$.

Data structures

Moving from $D(S)$ to $D(T)$:
Computation dominated by $O(1)$ evaluations of $f$ if $f$ is extremely slow.
But usually $f$ is not so slow.
How the quantum walk works:
Start from uniform superposition.
Repeat \( \approx 0 \times 6 \cdot 2^n = r \) times:
Negate \( a_{S,T} \) if \( S \) contains collision.
Repeat \( \approx 0 \times \sqrt{r} \) times:
For each \( T \): Diffuse \( a_{S,T} \) across all \( S \).
For each \( S \): Diffuse \( a_{S,T} \) across all \( T \).
Now high probability that \( T \) contains collision.
Cost \( r + 2^n = \sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to \((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \( a \) to low-dim vector.
Analyze evolution of this vector.

e.g. \( n = 15, r = 1024 \), after 27 negations and 1242 diffusions:

\[
\begin{align*}
\Pr[\text{class (0, 0)}] &\approx 0.011; - \\
\Pr[\text{class (0, 1)}] &\approx 0.007; + \\
\Pr[\text{class (1, 0)}] &\approx 0.007; - \\
\Pr[\text{class (1, 1)}] &\approx 0.001; - \\
\Pr[\text{class (1, 2)}] &\approx 0.034; + \\
\Pr[\text{class (2, 1)}] &\approx 0.003; + \\
\Pr[\text{class (2, 2)}] &\approx 0.938; +
\end{align*}
\]

Right column is sign of \( a_{S,T} \).

Data structures
Moving from \( D(S) \) to \( D(T) \):
dominated by \( O(1) \) evaluations of \( f \) if \( f \) is extremely slow.
But usually \( f \) is not so slow.
How the quantum walk works:

Start from uniform superposition.

Repeat \( \approx 0 \)

\[ 6 \cdot 2^n = r \]

Negate \( a_{S,T} \) if \( S \) contains collision.

Repeat \( \approx 0 \)

\[ 7 \cdot \sqrt{r} \]

For each \( T \):

Diffuse \( a_{S,T} \) across all \( S \).

For each \( S \):

Diffuse \( a_{S,T} \) across all \( T \).

Now high probability that \( T \) contains collision.

Cost \( r + 2^n = \sqrt{r} \). Optimize: \( 2^{2n/3} \).

Classify \((S, T)\) according to \((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);

reduce \( a \) to low-dim vector.

Analyze evolution of this vector.

e.g. \( n = 15, r = 1024 \), after 27 negations and 1242 diffusions:

\[
\begin{align*}
\Pr[\text{class }(0, 0)] &\approx 0.011; - \\
\Pr[\text{class }(0, 1)] &\approx 0.007; + \\
\Pr[\text{class }(1, 0)] &\approx 0.007; - \\
\Pr[\text{class }(1, 1)] &\approx 0.001; - \\
\Pr[\text{class }(1, 2)] &\approx 0.034; + \\
\Pr[\text{class }(2, 1)] &\approx 0.003; + \\
\Pr[\text{class }(2, 2)] &\approx 0.938; +
\end{align*}
\]

Right column is sign of \( a_{S,T} \).

Data structures

Moving from \( D(S) \) to \( D(T) \):

dominated by \( O(1) \) evaluations of \( f \) if \( f \) is extremely slow.

But usually \( f \) is not so slow.
Classify $(S, T)$ according to
$$(\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))$$;
reduce $a$ to low-dim vector.
Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after
27 negations and 1242 diffusions:

$$
\begin{align*}
\Pr[\text{class } (0, 0)] & \approx 0.011; - \\
\Pr[\text{class } (0, 1)] & \approx 0.007; + \\
\Pr[\text{class } (1, 0)] & \approx 0.007; - \\
\Pr[\text{class } (1, 1)] & \approx 0.001; - \\
\Pr[\text{class } (1, 2)] & \approx 0.034; + \\
\Pr[\text{class } (2, 1)] & \approx 0.003; + \\
\Pr[\text{class } (2, 2)] & \approx 0.938; + 
\end{align*}
$$

Right column is sign of $a_{S,T}$.

Data structures
Moving from $D(S)$ to $D(T)$:
dominated by $O(1)$ evaluations of $f$ if $f$ is extremely slow.

But usually $f$ is not so slow.
Classify \((S, T)\) according to 
\((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \(a\) to low-dim vector.
Analyze evolution of this vector.

\[\text{e.g. } n = 15, \quad r = 1024, \quad \text{after} \]
27 negations and 1242 diffusions:

\[
\begin{align*}
\Pr[\text{class } (0, 0)] & \approx 0.011; - \\
\Pr[\text{class } (0, 1)] & \approx 0.007; + \\
\Pr[\text{class } (1, 0)] & \approx 0.007; - \\
\Pr[\text{class } (1, 1)] & \approx 0.001; - \\
\Pr[\text{class } (1, 2)] & \approx 0.034; + \\
\Pr[\text{class } (2, 1)] & \approx 0.003; + \\
\Pr[\text{class } (2, 2)] & \approx 0.938; + \\
\end{align*}
\]

Right column is sign of \(a_{S, T}\).

Data structures
Moving from \(D(S)\) to \(D(T)\):
dominated by \(O(1)\) evaluations
of \(f\) if \(f\) is extremely slow.

But usually \(f\) is not so slow.
Store set \(S\) and multiset \(f(S)\)
in, e.g., hash tables?
Classify \((S, T)\) according to 
\((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \(a\) to low-dim vector.
Analyze evolution of this vector.

\[e.g. \ n = 15, \ r = 1024, \ \text{after} \ 27 \ \text{negations and} \ 1242 \ \text{diffusions:}\]

\[\Pr[\text{class } (0, 0)] \approx 0.011; -\]
\[\Pr[\text{class } (0, 1)] \approx 0.007; +\]
\[\Pr[\text{class } (1, 0)] \approx 0.007; -\]
\[\Pr[\text{class } (1, 1)] \approx 0.001; -\]
\[\Pr[\text{class } (1, 2)] \approx 0.034; +\]
\[\Pr[\text{class } (2, 1)] \approx 0.003; +\]
\[\Pr[\text{class } (2, 2)] \approx 0.938; +\]

Right column is sign of \(a_{S, T}\).

---

**Data structures**

Moving from \(D(S)\) to \(D(T)\): dominated by \(O(1)\) evaluations of \(f\) if \(f\) is extremely slow.

But usually \(f\) is not so slow.
Store set \(S\) and multiset \(f(S)\) in, e.g., hash tables?

Minor problem: time to hash \(S\) is huge for some sets \(S\).
Classify \((S, T)\) according to 
\((\#(S \cap \{p, q\}), \#(T \cap \{p, q\}))\);
reduce \(a\) to low-dim vector.
Analyze evolution of this vector.

e.g. \(n = 15, r = 1024\), after
27 negations and 1242 diffusions:

\[
\begin{align*}
\Pr[\text{class (0, 0)}] & \approx 0.011; - \\
\Pr[\text{class (0, 1)}] & \approx 0.007; + \\
\Pr[\text{class (1, 0)}] & \approx 0.007; - \\
\Pr[\text{class (1, 1)}] & \approx 0.001; - \\
\Pr[\text{class (1, 2)}] & \approx 0.034; + \\
\Pr[\text{class (2, 1)}] & \approx 0.003; + \\
\Pr[\text{class (2, 2)}] & \approx 0.938; + \\
\end{align*}
\]

Right column is sign of \(a_{S,T}\).

Data structures
Moving from \(D(S)\) to \(D(T)\):
dominated by \(O(1)\) evaluations
of \(f\) if \(f\) is extremely slow.

But usually \(f\) is not so slow.
Store set \(S\) and multiset \(f(S)\)
in, e.g., hash tables?

Minor problem: time to hash \(S\)
is huge for some sets \(S\).

Fix: randomize hash function
(1979 Carter–Wegman),
and specify big enough time for
whole algorithm to be reliable.
12
Classify \((S, T)\) according to 
\(#(S \cap \{p, q\})\), 
\(#(T \cap \{p, q\})\);
reduce \(a\) to low-dim vector.
Analyze evolution of this vector.

15, \(r = 1024\), after 
27 negations and 1242 diffusions:
\[
\begin{align*}
(0, 0) & \approx 0.011; - \\
(0, 1) & \approx 0.007; + \\
(1, 0) & \approx 0.007; - \\
(1, 1) & \approx 0.001; - \\
(1, 2) & \approx 0.034; + \\
(2, 1) & \approx 0.003; + \\
(2, 2) & \approx 0.938; + 
\end{align*}
\]
Right column is sign of \(a_{S,T}\).

13
Data structures
Moving from \(D(S)\) to \(D(T)\):
dominated by \(O(1)\) evaluations
of \(f\) if \(f\) is extremely slow.
But usually \(f\) is not so slow.
Store set \(S\) and multiset \(f(S)\)
in, e.g., hash tables?
Minor problem: time to hash \(S\)
is huge for some sets \(S\).
Fix: randomize hash function
(1979 Carter–Wegman),
and specify big enough time for
whole algorithm to be reliable.

14
Major problem: hash table
depends on history, not just on
\(S\). Algorithm fails horribly.
Need history-independent \(D(S)\).
Classify $(S; T)$ according to $(\#(S \cap \{p, q\}); \#(T \cap \{p, q\}))$; reduce a to low-dim vector.

Analyze evolution of this vector. e.g. $n = 15$, $r = 1024$, after 27 negations and 1242 diffusions:

Pr[class (0; 0)] $\approx 0 : 011$; 
Pr[class (0; 1)] $\approx 0 : 007$; +
Pr[class (1; 0)] $\approx 0 : 007$; −
Pr[class (1; 1)] $\approx 0 : 001$; −
Pr[class (2; 1)] $\approx 0 : 034$; +
Pr[class (2; 2)] $\approx 0 : 938$; +

Right column is sign of $a_{S,T}$.

Data structures
Moving from $D(S)$ to $D(T)$: dominated by $O(1)$ evaluations of $f$ if $f$ is extremely slow.

But usually $f$ is not so slow. Store set $S$ and multiset $f(S)$ in, e.g., hash tables?

Minor problem: time to hash $S$ is huge for some sets $S$.

Fix: randomize hash function (1979 Carter–Wegman), and specify big enough time for whole algorithm to be reliable.

Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$. 
Classify \((S;T)\) according to \((#(S \cap \{p;q\}); #((T \cap \{p;q\}))\)); reduce \(a\) to low-dim vector.

Analyze evolution of this vector. e.g. \(n = 15, r = 1024, \) after 27 negations and 1242 diffusions:

\[
\begin{align*}
\Pr[\text{class (0; 0)}] & \approx 0.011; \\
\Pr[\text{class (0; 1)}] & \approx 0.007; \\
\Pr[\text{class (1; 0)}] & \approx 0.007; \\
\Pr[\text{class (1; 1)}] & \approx 0.001; \\
\Pr[\text{class (1; 2)}] & \approx 0.034; \\
\Pr[\text{class (2; 1)}] & \approx 0.003; \\
\Pr[\text{class (2; 2)}] & \approx 0.938; \\
\end{align*}
\]

Right column is sign of \(a\).
Data structures

Moving from $D(S)$ to $D(T)$: dominated by $O(1)$ evaluations of $f$ if $f$ is extremely slow.

But usually $f$ is not so slow.

Store set $S$ and multiset $f(S)$ in, e.g., hash tables?

Minor problem: time to hash $S$ is huge for some sets $S$.

Fix: randomize hash function (1979 Carter–Wegman), and specify big enough time for whole algorithm to be reliable.

Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$. 
Data structures

Moving from $D(S)$ to $D(T)$: dominated by $O(1)$ evaluations of $f$ if $f$ is extremely slow.

But usually $f$ is not so slow.

Store set $S$ and multiset $f(S)$ in, e.g., hash tables?

Minor problem: time to hash $S$ is huge for some sets $S$.

Fix: randomize hash function (1979 Carter–Wegman), and specify big enough time for whole algorithm to be reliable.

Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$.

2003 Ambainis: “combination of a hash table and a skip list”. Several pages of analysis.
Data structures

Moving from $D(S)$ to $D(T)$: dominated by $O(1)$ evaluations of $f$ if $f$ is extremely slow.

But usually $f$ is not so slow.

Store set $S$ and multiset $f(S)$ in, e.g., hash tables?

Minor problem: time to hash $S$ is huge for some sets $S$.

Fix: randomize hash function (1979 Carter–Wegman), and specify big enough time for whole algorithm to be reliable.

Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$.

2003 Ambainis: “combination of a hash table and a skip list”.

Several pages of analysis.


Simplest radix tree: Left subtree stores $\{x : (0, x) \in S\}$ if nonempty. Right subtree stores $\{x : (1, x) \in S\}$ if nonempty.
Data structures

Moving from $D(S)$ to $D(T)$: dominated by $O(1)$ evaluations of $f$ if $f$ is extremely slow.

But usually $f$ is not so slow. Store set $S$ and multiset $f(S)$ in, e.g., hash tables?

Major problem: time to hash $S$ is huge for some sets $S$.

Fix: randomize hash function (1979 Carter–Wegman), and specify big enough time for whole algorithm to be reliable.

Caveats

The $2^{2n} = 3$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.
Data structures
Moving from $D(S)$ to $D(T)$:
- Dominated by $O(1)$ evaluations of $f$ if $f$ is extremely slow.
- But usually $f$ is not so slow.
- Store set $S$ and multiset $f(S)$ in, e.g., hash tables?
- Minor problem: time to hash $S$ is huge for some sets $S$.
- Fix: randomize hash function (1979 Carter–Wegman), and specify big enough time for whole algorithm to be reliable.

Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.
- Need history-independent $D(S)$.
- 2003 Ambainis: “combination of a hash table and a skip list”.
- Several pages of analysis.
- Simplest radix tree: Left subtree stores $\{x : (0, x) \in S\}$ if nonempty. Right subtree stores $\{x : (1, x) \in S\}$ if nonempty.

Caveats
The $2^{2n/3}$ analysis assumes cheap random access to memory.
Justified by simplicity, not realism.
Data structures

Moving from $D(S)$ to $D(T)$: dominated by $O(1)$ evaluations of $f$ if $f$ is extremely slow.

But usually $f$ is not so slow.

Store set $S$ and multiset $f(S)$ in, e.g., hash tables?

Minor problem: time to hash $S$ is huge for some sets $S$.

Fix: randomize hash function (1979 Carter–Wegman), and specify big enough time for whole algorithm to be reliable.

Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$.

2003 Ambainis: “combination of a hash table and a skip list”. Several pages of analysis.


Simplest radix tree: Left subtree stores $\{x : (0,x) \in S\}$ if nonempty. Right subtree stores $\{x : (1,x) \in S\}$ if nonempty.

Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.
Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$.

2003 Ambainis: “combination of a hash table and a skip list”. Several pages of analysis.


Simplest radix tree: Left subtree stores $\{x : (0, x) \in S\}$ if nonempty. Right subtree stores $\{x : (1, x) \in S\}$ if nonempty.

Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.
Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$.

2003 Ambainis: “combination of a hash table and a skip list”. Several pages of analysis.


Simplest radix tree: Left subtree stores $\{x : (0,x) \in S\}$ if nonempty. Right subtree stores $\{x : (1,x) \in S\}$ if nonempty.

Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved?
Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$.

2003 Ambainis: “combination of a hash table and a skip list”. Several pages of analysis.


Simplest radix tree: Left subtree stores $\{x : (0, x) \in S\}$ if nonempty. Right subtree stores $\{x : (1, x) \in S\}$ if nonempty.

Caveats
The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved? 2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm.
Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$.

2003 Ambainis: “combination of a hash table and a skip list”. Several pages of analysis.


Simplest radix tree: Left subtree stores $\{x : (0, x) \in S\}$ if nonempty. Right subtree stores $\{x : (1, x) \in S\}$ if nonempty.

Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved? 

2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm. Lasers spread. Fibers lose. etc.
Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$.

2003 Ambainis: “combination of a hash table and a skip list”. Several pages of analysis.


Simplest radix tree: Left subtree stores $\{x : (0, x) \in S\}$ if nonempty. Right subtree stores $\{x : (1, x) \in S\}$ if nonempty.

Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved? 2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm. Lasers spread. Fibers lose. etc.

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.
Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly.

Need history-independent $D(S)$.

Ambainis: “combination of a hash table and a skip list”. Several pages of analysis.

2003 Ambainis: “combination of a hash table and a skip list”. Several pages of analysis.


Simplest radix tree: Left subtree stores $\{x : (0, x) \in S\}$ if nonempty. Right subtree stores $\{x : (1, x) \in S\}$ if nonempty.

Caveats
The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved? 2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm. Lasers spread. Fibers lose. etc.

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don’t seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.
Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly. Need history-independent $D(S)$.

2003 Ambainis: “combination of a hash table and a skip list”. Several pages of analysis.

2013 Bernstein–Jeffery–Lange–Meurer: radix tree. Simplest radix tree: Left subtree stores \[ \{x : (0; x) \in S\} \] if nonempty. Right subtree stores \[ \{x : (1; x) \in S\} \] if nonempty.

Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved? 2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm. Lasers spread. Fibers lose. etc.

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don’t seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.
Major problem: hash table depends on history, not just on $S$. Algorithm fails horribly. Need history-independent $D(S)$. 2003 Ambainis: “combination of a hash table and a skip list”. Several pages of analysis. 2013 Bernstein–Jeffery–Lange–Meurer: radix tree. Simplest radix tree: Left subtree stores $\{x : (0;x) \in S\}$ if nonempty. Right subtree stores $\{x : (1;x) \in S\}$ if nonempty.

Caveats
The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism. Can we move data using energy sublinear in distance moved? 2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm. Lasers spread. Fibers lose. etc. I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don’t seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.
Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved? 2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm. Lasers spread. Fibers lose. etc.

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don’t seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.
Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved?  
**2015 Intel presentation** says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm. Lasers spread. Fibers lose. etc.

I recommend algorithm analysis on 2-dim mesh of tiny processors: 
e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don’t seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:
• Parallelization reduces speedup.  
  $D \times$ speedup needs depth $D$. 

Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved? The 2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm. Lasers spread. Fibers lose. etc.

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don’t seem to exist in this model. E.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D \times$ speedup needs depth $D$.
- Reversibility is expensive.
Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved? 2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm. Lasers spread. Fibers lose. etc.

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don’t seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D \times$ speedup needs depth $D$.
- Reversibility is expensive.
- Quantum ops are expensive.
Caveats

The $2^{2n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved?  
2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm. Lasers spread. Fibers lose. etc.

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don’t seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D \times$ speedup needs depth $D$.

- Reversibility is expensive.

- Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.
The $2^{n/3}$ analysis assumes cheap random access to memory. Justified by simplicity, not realism.

Can we move data using energy sublinear in distance moved?

A 2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm.

Lasers spread. Fibers lose. etc.

I recommend algorithm analysis on 2-dim mesh of tiny processors:

e.g. $O_{472}$ for MQ (vs. $O_{462}$)

from 2017 Bernstein–Yang.

Many claimed quantum speedups don’t seem to exist in this model.

e.g. 2009 Bernstein analysis:

fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D \times$ speedup needs depth $D$.
- Reversibility is expensive.
- Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.
Many claimed quantum speedups don’t seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. \( D \times \text{ speedup needs depth } D \).
- Reversibility is expensive.
- Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.
Caveats

The $2^n = 3$ analysis assumes cheap random access to memory. Justified by simplicity, not realism. Can we move data using energy sublinear in distance moved? 2015 Intel presentation says that moving 8 bytes on wire at 22nm costs 11.20 pJ per 5mm.

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. $0 : 472$ for MQ (vs. $0 : 462$) from 2017 Bernstein–Yang.

Many claimed quantum speedups don’t seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:• Parallelization reduces speedup. $D \times$ speedup needs depth $D$.
• Reversibility is expensive.
• Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.

Background slides ...
Many claimed quantum speedups don’t seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. \( D \times \) speedup needs depth \( D \).
- Reversibility is expensive.
- Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.
Many claimed quantum speedups don’t seem to exist in this model.

e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

• Parallelization reduces speedup. Speedup needs depth $D$.

• Reversibility is expensive.

• Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.

“Quantum algorithm” means an algorithm that a quantum computer can run.

i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?
Many claimed quantum speedups don’t seem to exist in this model. For example, the Bernstein analysis of 2009 showed that the fastest algorithm known for random-collision search is the 1994 van Oorschot–Wiener algorithm.

Further obstacles to Grover’s algorithm:

- Parallelization reduces speedup. 
- Reversibility is expensive. 
- Quantum operations are expensive.

Grover’s algorithm poses a much smaller risk to cryptography than Shor’s algorithm.

What do quantum computers do?

“Quantum algorithm” means an algorithm that a quantum computer can run. i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?
Many claimed quantum speedups don't seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener. Further obstacles to Grover:

- Parallelization reduces speedup. $D \times$ speedup needs depth $D$.
- Reversibility is expensive.
- Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.

What do quantum computers do? “Quantum algorithm” means an algorithm that a quantum computer can run. i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?
What do quantum computers do?

“Quantum algorithm” means an algorithm that a quantum computer can run. i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?
What do quantum computers do?

“Quantum algorithm” means an algorithm that a quantum computer can run. i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?
What do quantum computers do?

“Quantum algorithm” means an algorithm that a quantum computer can run. i.e. a sequence of instructions, where each instruction is in a quantum computer's supported instruction set.

Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”.

How do we know which instructions a quantum computer will support?
What do quantum computers do?

“Quantum algorithm” means an algorithm that a quantum computer can run.

i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1) contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “T gate”. 
What do quantum computers do?

“Quantum algorithm” means an algorithm that a quantum computer can run.

i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “T gate”.
What do quantum computers do?

“Quantum algorithm” means an algorithm that a quantum computer can run. i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “$T$ gate”.

Making these instructions work is the main goal of quantum-computer engineering.
What do quantum computers do?

“Quantum algorithm” means an algorithm that a quantum computer can run.

i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “$T$ gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions to compute “Toffoli gate”; . . . “Simon’s algorithm”; . . . “Shor’s algorithm”; etc.
What do quantum computers do?

“Quantum algorithm” means an algorithm that a quantum computer can run.

i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “$T$ gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions to compute “Toffoli gate”; … “Simon’s algorithm”; … “Shor’s algorithm”; etc.

General belief: Traditional CPU isn’t QC1; e.g. can’t factor quickly.
What do quantum computers do?

Quantum algorithm means an algorithm that a quantum computer can run. i.e. a sequence of instructions, where each instruction is in a quantum computer’s supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1):
contains many “qubits”;
can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “$T$ gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions to compute “Toffoli gate”; … “Simon’s algorithm”; … “Shor’s algorithm”; etc.

General belief: Traditional CPU isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.
Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “T gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions to compute “Toffoli gate”; … “Simon’s algorithm”; … “Shor’s algorithm”; etc.

General belief: Traditional CPU isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.
Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “T gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions to compute “Toffoli gate”; 
... “Simon’s algorithm”; 
... “Shor’s algorithm”; etc.

General belief: Traditional CPU isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.

Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “$T$ gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions to compute “Toffoli gate”; . . . “Simon’s algorithm”; . . . “Shor’s algorithm”; etc.

General belief: Traditional CPU isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.
Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “T gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions to compute “Toffoli gate”; … “Simon’s algorithm”; … “Shor’s algorithm”; etc.

General belief: Traditional CPU isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.

General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill “Quantum algorithms for quantum field theories”.
Quantum computer type 1 (QC1):
contains many “qubits”;
can efficiently perform
“NOT gate”, “Hadamard gate”,
controlled NOT gate”, “T gate”.
Making these instructions work
is the main goal of quantum-
computer engineering.

Combine these instructions
to compute “Toffoli gate”;
::: “Simon’s algorithm”;
::: “Shor’s algorithm”; etc.

General belief: Traditional CPU
isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of
quantum computers introduced
by 1982 Feynman “Simulating
physics with computers”.

General belief: any QC1 is a QC2.
Partial proof: see, e.g.,
2011 Jordan–Lee–Preskill
“Quantum algorithms for
quantum field theories”.

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.
Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “\(T\) gate”. Making these instructions work is the main goal of quantum-computer engineering. Combine these instructions to compute “Toffoli gate”; ::: “Simon’s algorithm”; ::: “Shor’s algorithm”; etc. General belief: Traditional CPU isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired. This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”. General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill “Quantum algorithms for quantum field theories”.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.
Quantum computer type 1 (QC1):
contains many “qubits”;
can efficiently perform
“NOT gate”, “Hadamard gate”,
“controlled NOT gate”, “T gate”.
Making these instructions work
is the main goal of quantum-
computer engineering.
Combine these instructions
to compute “Toffoli gate”;
::: “Simon’s algorithm”;
::: “Shor’s algorithm”; etc.
General belief: Traditional CPU
isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.
This is the original concept of
quantum computers introduced
by 1982 Feynman “Simulating
physics with computers”.
General belief: any QC1 is a QC2.
Partial proof: see, e.g.,
2011 Jordan–Lee–Preskill
“Quantum algorithms for
quantum field theories”.

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.
Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.

General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill “Quantum algorithms for quantum field theories”.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.
Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.

General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill “Quantum algorithms for quantum field theories”.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.
Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.

General belief: any QC1 is a QC2.
Partial proof: see, e.g., 2011 Jordan–Lee–Preskill “Quantum algorithms for quantum field theories”.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief: look, we’re building a QC1.
Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired. This is the original concept of quantum computers introduced by Feynman “Simulating physics with computers”.

General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill “Quantum algorithms for quantum field theories”.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we’re building a QC1.

A note on D-Wave
Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.
Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired. This is the original concept of quantum computers introduced by Feynman in 1982 “Simulating physics with computers”.

General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill “Quantum algorithms for quantum field theories”.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we’re building a QC1.

A note on D-Wave: Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.
Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired. This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers.”

General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill “Quantum algorithms for quantum field theories.”

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we’re building a QC1.

A note on D-Wave
Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.
Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we’re building a QC1.

A note on D-Wave
Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.
Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we’re building a QC1.

A note on D-Wave

Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
  • collecting venture capital;
Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we’re building a QC1.

A note on D-Wave

Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
- collecting venture capital;
- selling some machines;
Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief:
any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief:
look, we’re building a QC1.

A note on D-Wave
Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
• collecting venture capital;
• selling some machines;
• collecting possibly useful engineering expertise;
Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we’re building a QC1.

A note on D-Wave
Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
• collecting venture capital;
• selling some machines;
• collecting possibly useful engineering expertise;
• not being punished for deceiving people.
Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we’re building a QC1.

A note on D-Wave

Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

• collecting venture capital;
• selling some machines;
• collecting possibly useful engineering expertise;
• not being punished for deceiving people.

Is D-Wave a bad investment?
Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

- General belief: any QC2 is a QC3.
  - Argument for belief: any physical computer must follow the laws of quantum so a QC2 can efficiently simulate any physical computer.
- General belief: any QC3 is a QC1.
  - Argument for belief: we’re building a QC1.

A note on D-Wave

Apparent scientific consensus: current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data (“state”) stored in 3 bits: a list of 3 elements of \{0, 1\}.

e.g.: (0, 0, 0).
Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we’re building a QC1.

A note on D-Wave

Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
• collecting venture capital;
• selling some machines;
• collecting possibly useful engineering expertise;
• not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data (“state”) stored in 3 bits:
a list of 3 elements of \{0, 1\}.

E.g.: (0, 0, 0).
Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently. General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data (“state”) stored in 3 bits: a list of 3 elements of \{0, 1\}. e.g.: (0, 0, 0).
A note on D-Wave

Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

• collecting venture capital;
• selling some machines;
• collecting possibly useful engineering expertise;
• not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data (“state”) stored in 3 bits: a list of 3 elements of \{0, 1\}. e.g.: (0, 0, 0).
A note on D-Wave

Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

• collecting venture capital;
• selling some machines;
• collecting possibly useful engineering expertise;
• not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data (“state”) stored in 3 bits: a list of 3 elements of \{0, 1\}.

e.g.: (0, 0, 0).
e.g.: (1, 1, 1).
A note on D-Wave

Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
• collecting venture capital;
• selling some machines;
• collecting possibly useful engineering expertise;
• not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data (“state”) stored in 3 bits: a list of 3 elements of \{0, 1\}.
e.g.: (0, 0, 0).
e.g.: (1, 1, 1).
e.g.: (0, 1, 1).
A note on D-Wave

Apparent scientific consensus: Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
• collecting venture capital;
• selling some machines;
• collecting possibly useful engineering expertise;
• not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data (“state”) stored in 3 bits: a list of 3 elements of \{0, 1\}.

- e.g.: (0, 0, 0).
- e.g.: (1, 1, 1).
- e.g.: (0, 1, 1).

Data stored in 64 bits: a list of 64 elements of \{0, 1\}.
A note on D-Wave
Apparent scientific consensus:
Current “quantum computers” from D-Wave are useless—
can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
• collecting venture capital;
• selling some machines;
• collecting possibly useful engineering expertise;
• not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer
Data (“state”) stored in 3 bits:
a list of 3 elements of \{0, 1\}.
e.g.: (0, 0, 0).
e.g.: (1, 1, 1).
e.g.: (0, 1, 1).

Data stored in 64 bits:
a list of 64 elements of \{0, 1\}.
e.g.: (1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1).
A note on D-Wave

Apparent scientific consensus:

Current “quantum computers” from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

D-Wave is
• collecting venture capital;
• selling some machines;
• collecting possibly useful engineering expertise;
• not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of \(\{0, 1\}\).
e.g.: \((0, 0, 0)\).
e.g.: \((1, 1, 1)\).
e.g.: \((0, 1, 1)\).

Data stored in 64 bits:
a list of 64 elements of \(\{0, 1\}\).
e.g.: \((1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1)\).
A note on D-Wave
Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is
• collecting venture capital;
• selling some machines;
• collecting possibly useful
engineering expertise;
• not being punished
for deceiving people.

Is D-Wave a bad investment?

The state of a computer
Data ("state") stored in 3 bits:
a list of 3 elements of \{0, 1\}.
e.g.: (0, 0, 0).
e.g.: (1, 1, 1).
e.g.: (0, 1, 1).

Data stored in 64 bits:
a list of 64 elements of \{0, 1\}.
e.g.: (1, 1, 1, 1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
1, 0, 1, 0, 0, 1, 0, 0, 0, 1,
0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1).

The state of a quantum computer
Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
A note on D-Wave

Apparent scientific consensus:
Current "quantum computers" from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of \{0, 1\}.

- e.g.: (0, 0, 0).
- e.g.: (1, 1, 1).
- e.g.: (0, 1, 1).

Data stored in 64 bits:
a list of 64 elements of \{0, 1\}.

- e.g.: (1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1).

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.

- e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of \( \{0, 1\} \).
e.g.: (0, 0, 0).
e.g.: (1, 1, 1).
e.g.: (0, 1, 1).

Data stored in 64 bits:
a list of 64 elements of \( \{0, 1\} \).
e.g.: (1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1).

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of \{0, 1\}.
e.g.: (0, 0, 0).
e.g.: (1, 1, 1).
e.g.: (0, 1, 1).

Data stored in 64 bits:
a list of 64 elements of \{0, 1\}.
e.g.: (1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1).

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
e.g.: (−2, 7, −1, 8, 1, −8, −2, 8).
The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of \{0, 1\}.
e.g.: (0, 0, 0).
e.g.: (1, 1, 1).
e.g.: (0, 1, 1).

Data stored in 64 bits:
a list of 64 elements of \{0, 1\}.
e.g.: (1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,} 23

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
e.g.: (−2, 7, −1, 8, 1, −8, −2, 8).
e.g.: (0, 0, 0, 0, 0, 1, 0, 0).
The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of \{0, 1\}.
e.g.: (0, 0, 0).
e.g.: (1, 1, 1).
e.g.: (0, 1, 1).

Data stored in 64 bits:
a list of 64 elements of \{0, 1\}.
e.g.: (1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1).

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
e.g.: (−2, 7, −1, 8, 1, −8, −2, 8).
e.g.: (0, 0, 0, 0, 0, 1, 0, 0).

Data stored in 4 qubits: a list of
16 numbers, not all zero. e.g.:
(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3).
The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of \{0, 1\}.
e.g.: (0, 0, 0).
e.g.: (1, 1, 1).
e.g.: (0, 1, 1).

Data stored in 64 bits:
a list of 64 elements of \{0, 1\}.
e.g.: (1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1).

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
e.g.: (−2, 7, −1, 8, 1, −8, −2, 8).
e.g.: (0, 0, 0, 0, 0, 1, 0, 0).

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3).

Data stored in 64 qubits:
a list of \(2^{64}\) numbers, not all zero.
Data stored in 3 bits: a list of 3 elements of \{0, 1\}.

- e.g.: (0, 0, 1).
- e.g.: (1, 1, 1).
- e.g.: (0, 1, 1).

Data stored in 64 bits: a list of 64 elements of \{0, 1\}.

- e.g.: (1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1).

Data stored in 3 qubits: a list of 8 numbers, not all zero.

- e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
- e.g.: (1, 1, 1).
- e.g.: (0, 1, 1).

Data stored in 4 qubits: a list of 16 numbers, not all zero.

- e.g.: (3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7).
- e.g.: (0, 0, 0, 0, 0, 0, 0).
- e.g.: (0, 0, 0, 0, 0, 0, 0).

Data stored in 64 qubits: a list of 264 numbers, not all zero.

Data stored in 1000 qubits: a list of 21000 numbers, not all zero.
The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of \{0, 1\}.
e.g.: (0; 0; 0).
e.g.: (1; 1).
e.g.: (1; 1).

Data stored in 64 bits:
a list of 64 elements of \{0, 1\}.
e.g.: (1; 1; 1; 1; 0; 0; 0; 1,
0; 0; 1; 1; 0; 0; 0; 1,
0; 0; 1; 1; 0; 0; 0; 0).

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
e.g.: (−2, 7, −1, 8, 1, −8, −2, 8).
e.g.: (0, 0, 0, 0, 0, 1, 0, 0).

Data stored in 4 qubits: a list of
16 numbers, not all zero. e.g.:
(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3).

Data stored in 64 qubits:
a list of \(2^{64}\) numbers, not all zero.

Data stored in 1000 qubits: a list
of \(2^{1000}\) numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list
of numbers stored in \(n\) qubits.

The state of a computer

Data stored in 3 bits: a list of \{0, 1\}.
e.g.: (0; 0; 0; 0; 1; 0; 0; 0; 1).

Data stored in 64 bits: a list of 64 elements of \{0, 1\}.
e.g.: (0; 0; 0; 0; 1; 0; 0; 0; 1).

The state of a quantum computer

Data stored in 3 qubits: a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
e.g.: (−2, 7, −1, 8, 1, −8, −2, 8).
e.g.: (0, 0, 0, 0, 1, 0, 0).

Data stored in 4 qubits: a list of 16 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3).

Data stored in 64 qubits: a list of \(2^{64}\) numbers, not all zero.

Data stored in 1000 qubits: a list of \(2^{1000}\) numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in \(n\) qubits.
The state of a computer

Data ("state") stored in 3 bits:
- a list of 3 elements of \{0, 1\}.
- e.g.: (0, 0, 0).
- e.g.: (1, 1, 1).
- e.g.: (0, 1, 1).

Data stored in 64 bits:
- a list of 64 elements of \{0, 1\}.
- e.g.: (1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0).

The state of a quantum computer

Data stored in 3 qubits:
- a list of 8 numbers, not all zero.
- e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
- e.g.: (−2, 7, −1, 8, 1, −8, −2, 8).
- e.g.: (0, 0, 0, 0, 0, 1, 0, 0).

Data stored in 4 qubits: a list of 16 numbers, not all zero.
- e.g.: (3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3).

Data stored in 64 qubits:
- a list of $2^{64}$ numbers, not all zero.

Data stored in 1000 qubits: a list of $2^{1000}$ numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.

Cannot simply look at the list of numbers stored in $n$ qubits.
The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: \((3, 1, 4, 1, 5, 9, 2, 6)\).
e.g.: \((-2, 7, -1, 8, 1, -8, -2, 8)\).
e.g.: \((0, 0, 0, 0, 0, 1, 0, 0)\).

Data stored in 4 qubits: a list of 16 numbers, not all zero.
e.g.: \((3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)\).

Data stored in 64 qubits:
a list of \(2^{64}\) numbers, not all zero.

Data stored in 1000 qubits: a list of \(2^{1000}\) numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in \(n\) qubits.
The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
e.g.: (−2, 7, −1, 8, 1, −8, −2, 8).
e.g.: (0, 0, 0, 0, 0, 1, 0, 0).

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3).

Data stored in 64 qubits:
a list of $2^{64}$ numbers, not all zero.

Data stored in 1000 qubits: a list of $2^{1000}$ numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in $n$ qubits.

Measuring $n$ qubits
• produces $n$ bits and
• destroys the state.
The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
e.g.: (−2, 7, −1, 8, 1, −8, −2, 8).
e.g.: (0, 0, 0, 0, 1, 0, 0).

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.: (3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3).

Data stored in 64 qubits:
a list of $2^{64}$ numbers, not all zero.

Data stored in 1000 qubits: a list of $2^{1000}$ numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in $n$ qubits.

Measuring $n$ qubits
• produces $n$ bits and
• destroys the state.

If $n$ qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces $q$ with probability $|a_q|^2 / \sum_r |a_r|^2$. 
The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: (3, 1, 4, 1, 5, 9, 2, 6).
e.g.: (−2, 7, −1, 8, 1, −8, −2, 8).
e.g.: (0, 0, 0, 0, 1, 0, 0).

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3).

Data stored in 64 qubits: a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.

Cannot simply look at the list of numbers stored in \( n \) qubits.

**Measuring** \( n \) qubits

• produces \( n \) bits and
• destroys the state.

If \( n \) qubits have state \((a_0, a_1, \ldots, a_{2^n−1})\) then measurement produces \( q \) with probability \( |a_q|^2 / \sum_r |a_r|^2 \).

State is then all zeros except 1 at position \( q \).
The state of a quantum computer

Data stored in 3 qubits:

- a list of 8 numbers, not all zero.
  - e.g.: (3; 1; 4; 1; 5; 9; 2; 6).
  - (2; 7; −1; 8; 1; −8; −2; 8).
  - (0; 0; 0; 0; 1; 0; 0).

Data stored in 4 qubits: a list of 16 numbers, not all zero.
- e.g.: (3; 1; 4; 1; ...)
- (0; 0; 0; 0; 1; 0; 0; 0).

Data stored in 64 qubits: a list of 2^64 numbers, not all zero.

Data stored in 1000 qubits: a list of 2^1000 numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state (a_0, a_1, ..., a_{2^n-1}) then measurement produces q with probability |a_q|^2 / \sum_r |a_r|^2.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have state (1; 1; 1; 1; 1; 1; 1; 1).
The state of a quantum computer

Data stored in 3 qubits:
- a list of 8 numbers, not all zero.
  - e.g.: (3; 1; 4; 1; ...)

Data stored in 4 qubits:
- a list of 16 numbers, not all zero.
  - e.g.: (3; 5; 1; 7; 4; 1; 5; 9; ...)

Data stored in 64 qubits:
- a list of $2^{64}$ numbers, not all zero.

Data stored in 1000 qubits:
- a list of $2^{1000}$ numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.

Cannot simply look at the list of numbers stored in $n$ qubits.

Measuring $n$ qubits
- produces $n$ bits and
- destroys the state.

If $n$ qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces $q$ with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position $q$. 

e.g.: Say 3 qubits have state (1, 1, 1, 1, 1, 1, 1, 1).
Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in $n$ qubits.

**Measuring** $n$ qubits

- produces $n$ bits and
- destroys the state.

If $n$ qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces $q$ with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position $q$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$. 

24

25
Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in $n$ qubits.

**Measuring** $n$ qubits

- produces $n$ bits and
- destroys the state.

If $n$ qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces $q$ with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position $q$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$. 
Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in $n$ qubits.

**Measuring** $n$ qubits

- produces $n$ bits and
- destroys the state.

If $n$ qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces $q$ with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position $q$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces

- $000 = 0$ with probability $1/8$;
- $001 = 1$ with probability $1/8$;
- $010 = 2$ with probability $1/8$;
- $011 = 3$ with probability $1/8$;
- $100 = 4$ with probability $1/8$;
- $101 = 5$ with probability $1/8$;
- $110 = 6$ with probability $1/8$;
- $111 = 7$ with probability $1/8$.
Measuring a quantum computer
Can simply look at a bit.
Cannot simply look at the list of numbers stored in \( n \) qubits.

**Measuring** \( n \) qubits
- produces \( n \) bits and
- destroys the state.

If \( n \) qubits have state
\((a_0, a_1, \ldots, a_{2^n-1})\)
then measurement produces \( q \) with probability \( |a_q|^2 / \sum_r |a_r|^2 \).
State is then all zeros except 1 at position \( q \).

e.g.: Say 3 qubits have state \((1, 1, 1, 1, 1, 1, 1, 1)\).
Measurement produces
\(000 = 0 \text{ with probability } 1/8;\)
\(001 = 1 \text{ with probability } 1/8;\)
\(010 = 2 \text{ with probability } 1/8;\)
\(011 = 3 \text{ with probability } 1/8;\)
\(100 = 4 \text{ with probability } 1/8;\)
\(101 = 5 \text{ with probability } 1/8;\)
\(110 = 6 \text{ with probability } 1/8;\)
\(111 = 7 \text{ with probability } 1/8.\)

“Quantum RNG.”
Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in $n$ qubits.

**Measuring** $n$ qubits

- produces $n$ bits and
- destroys the state.

If $n$ qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces $q$ with probability $|a_q|^2/\sum_r |a_r|^2$.

State is then all zeros except 1 at position $q$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces

- $000 = 0$ with probability $1/8$;
- $001 = 1$ with probability $1/8$;
- $010 = 2$ with probability $1/8$;
- $011 = 3$ with probability $1/8$;
- $100 = 4$ with probability $1/8$;
- $101 = 5$ with probability $1/8$;
- $110 = 6$ with probability $1/8$;
- $111 = 7$ with probability $1/8$.

“Quantum RNG.”

Warning: Quantum RNGs sold today are measurably biased.
Measuring a quantum computer

can simply look at a bit.
simply look at the list
of numbers stored in \( n \) qubits.

**Measuring** \( n \) qubits

produces \( n \) bits and

destroys the state.

If \( n \) qubits have state

\((a_0; a_1; \ldots; a_{2^n-1})\)

then measurement produces \( q \)

with probability

\[ |a_q|^2 / \sum_r |a_r|^2. \]

then all zeros
at position \( q \).

**e.g.:** Say 3 qubits have state

\((1, 1, 1, 1, 1, 1, 1, 1)\).

Measurement produces

000 = 0 with probability \( 1/8 \);

001 = 1 with probability \( 1/8 \);

010 = 2 with probability \( 1/8 \);

011 = 3 with probability \( 1/8 \);

100 = 4 with probability \( 1/8 \);

101 = 5 with probability \( 1/8 \);

110 = 6 with probability \( 1/8 \);

111 = 7 with probability \( 1/8 \).

"Quantum RNG."

Warning: Quantum RNGs sold
today are measurably biased.
Measuring a quantum computer can simply look at a bit. Cannot simply look at the list of numbers stored in \( n \) qubits. Measuring \( n \) qubits produces \( n \) bits and destroys the state. If \( n \) qubits have state \((a_0; a_1; \ldots; a_{2^n-1})\) then measurement produces \( q \) with probability \(|a_q|^2 / \sum_r |a_r|^2\). State is then all zeros except 1 at position \( q \).

e.g.: Say 3 qubits have state \((1, 1, 1, 1, 1, 1, 1, 1)\).

Measurement produces
000 = 0 with probability 1/8;
001 = 1 with probability 1/8;
010 = 2 with probability 1/8;
011 = 3 with probability 1/8;
100 = 4 with probability 1/8;
101 = 5 with probability 1/8;
110 = 6 with probability 1/8;
111 = 7 with probability 1/8.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say 3 qubits have state \((3, 1, 4, 1, 5, 9, 2, 6)\).
Measuring a quantum computer
Can simply look at a bit.
Cannot simply look at the list
of numbers stored in \( n \) qubits.

Measuring \( n \) qubits
• produces \( n \) bits and
• destroys the state.

If \( n \) qubits have state 
\( (a_0; a_1; \ldots; a_{2^n-1}) \) then

measurement produces
\( q \) with probability
\( |a_q|^2 \).

State is then all zeros
except 1 at position \( q \).

e.g.: Say 3 qubits have state
\((1, 1, 1, 1, 1, 1, 1, 1)\).

Measurement produces
\(000 = 0 \) with probability \( 1/8 \);
\(001 = 1 \) with probability \( 1/8 \);
\(010 = 2 \) with probability \( 1/8 \);
\(011 = 3 \) with probability \( 1/8 \);
\(100 = 4 \) with probability \( 1/8 \);
\(101 = 5 \) with probability \( 1/8 \);
\(110 = 6 \) with probability \( 1/8 \);
\(111 = 7 \) with probability \( 1/8 \).

“How Quantum RNG.”

Warning: Quantum RNGs sold
today are measurably biased.
e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
000 = 0 with probability 1/8;
001 = 1 with probability 1/8;
010 = 2 with probability 1/8;
011 = 3 with probability 1/8;
100 = 4 with probability 1/8;
101 = 5 with probability 1/8;
110 = 6 with probability 1/8;
111 = 7 with probability 1/8.

“Quantum RNG.”

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$. 
e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
000 = 0 with probability $1/8$;
001 = 1 with probability $1/8$;
010 = 2 with probability $1/8$;
011 = 3 with probability $1/8$;
100 = 4 with probability $1/8$;
101 = 5 with probability $1/8$;
110 = 6 with probability $1/8$;
111 = 7 with probability $1/8$.

“Quantum RNG.”

Warning: Quantum RNGs sold today are measurably biased.

---

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
000 = 0 with probability $9/173$;
001 = 1 with probability $1/173$;
010 = 2 with probability $16/173$;
011 = 3 with probability $1/173$;
100 = 4 with probability $25/173$;
101 = 5 with probability $81/173$;
110 = 6 with probability $4/173$;
111 = 7 with probability $36/173$. 
e.g.: Say 3 qubits have state $(1,1,1,1,1,1,1,1)$. Measurement produces
000 = 0 with probability $1/8$;
001 = 1 with probability $1/8$;
010 = 2 with probability $1/8$;
011 = 3 with probability $1/8$;
100 = 4 with probability $1/8$;
101 = 5 with probability $1/8$;
110 = 6 with probability $1/8$;
111 = 7 with probability $1/8$.

“Quantum RNG.”

Warning: Quantum RNGs sold today are measurably biased.

---

e.g.: Say 3 qubits have state $(3,1,4,1,5,9,2,6)$. Measurement produces
000 = 0 with probability $9/173$;
001 = 1 with probability $1/173$;
010 = 2 with probability $16/173$;
011 = 3 with probability $1/173$;
100 = 4 with probability $25/173$;
101 = 5 with probability $81/173$;
110 = 6 with probability $4/173$;
111 = 7 with probability $36/173$.

5 is most likely outcome.
Say 3 qubits have state \((1, 1, 1, 1, 1, 1, 1, 1)\).

Measurement produces
- \(000 = 0\) with probability \(1/8\);
- \(001 = 1\) with probability \(1/8\);
- \(010 = 2\) with probability \(1/8\);
- \(011 = 3\) with probability \(1/8\);
- \(100 = 4\) with probability \(1/8\);
- \(101 = 5\) with probability \(1/8\);
- \(110 = 6\) with probability \(1/8\);
- \(111 = 7\) with probability \(1/8\).

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

---

e.g.: Say 3 qubits have state \((3, 1, 4, 1, 5, 9, 2, 6)\).

Measurement produces
- \(000 = 0\) with probability \(9/173\);
- \(001 = 1\) with probability \(1/173\);
- \(010 = 2\) with probability \(16/173\);
- \(011 = 3\) with probability \(1/173\);
- \(100 = 4\) with probability \(25/173\);
- \(101 = 5\) with probability \(81/173\);
- \(110 = 6\) with probability \(4/173\);
- \(111 = 7\) with probability \(36/173\).

5 is most likely outcome.

---

e.g.: Say 3 qubits have state \((0, 0, 0, 0, 0, 1, 0, 0)\).
e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$. Measurement produces
000 = 0 with probability $1/8$;  
001 = 1 with probability $1/8$;  
010 = 2 with probability $1/8$;  
011 = 3 with probability $1/8$;  
100 = 4 with probability $1/8$;  
101 = 5 with probability $1/8$;  
110 = 6 with probability $1/8$;  
111 = 7 with probability $1/8$.

"Quantum RNG." Warning: Quantum RNGs sold today are measurably biased.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$. Measurement produces
000 = 0 with probability $9/173$;  
001 = 1 with probability $1/173$;  
010 = 2 with probability $16/173$;  
011 = 3 with probability $1/173$;  
100 = 4 with probability $25/173$;  
101 = 5 with probability $81/173$;  
110 = 6 with probability $4/173$;  
111 = 7 with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.
e.g.: Say 3 qubits have state
(1, 1, 1, 1, 1, 1, 1, 1).
Measurement produces
000 = 0 with probability 1 = 8;
001 = 1 with probability 1 = 8;
010 = 2 with probability 1 = 8;
011 = 3 with probability 1 = 8;
100 = 4 with probability 1 = 8;
101 = 5 with probability 1 = 8;
110 = 6 with probability 1 = 8;
111 = 7 with probability 1 = 8.

“Quantum RNG.”

Warning: Quantum RNGs sold
today are measurably biased.

e.g.: Say 3 qubits have state
(3, 1, 4, 1, 5, 9, 2, 6).
Measurement produces
000 = 0 with probability 9/173;
001 = 1 with probability 1/173;
010 = 2 with probability 16/173;
011 = 3 with probability 1/173;
100 = 4 with probability 25/173;
101 = 5 with probability 81/173;
110 = 6 with probability 4/173;
111 = 7 with probability 36/173.

5 is most likely outcome.

e.g.: Say 3 qubits have state
(0, 0, 0, 0, 0, 0, 1, 0, 0).
e.g.: Say 3 qubits have state 
\((3, 1, 4, 1, 5, 9, 2, 6)\).

Measurement produces 
\(000 = 0 \text{ with probability } \frac{9}{173};\) 
\(001 = 1 \text{ with probability } \frac{1}{173};\) 
\(010 = 2 \text{ with probability } \frac{16}{173};\) 
\(011 = 3 \text{ with probability } \frac{1}{173};\) 
\(100 = 4 \text{ with probability } \frac{25}{173};\) 
\(101 = 5 \text{ with probability } \frac{81}{173};\) 
\(110 = 6 \text{ with probability } \frac{4}{173};\) 
\(111 = 7 \text{ with probability } \frac{36}{173}.\)

5 is most likely outcome.

---

e.g.: Say 3 qubits have state 
\((0, 0, 0, 0, 0, 1, 0, 0).\)
e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability $0$;
$001 = 1$ with probability $0$;
$010 = 2$ with probability $0$;
$011 = 3$ with probability $0$;
$100 = 4$ with probability $0$;
$101 = 5$ with probability $1$;
$110 = 6$ with probability $0$;
$111 = 7$ with probability $0$. 
e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces

- $000 = 0$ with probability $9/173$;
- $001 = 1$ with probability $1/173$;
- $010 = 2$ with probability $16/173$;
- $011 = 3$ with probability $1/173$;
- $100 = 4$ with probability $25/173$;
- $101 = 5$ with probability $81/173$;
- $110 = 6$ with probability $4/173$;
- $111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces

- $000 = 0$ with probability $0$;
- $001 = 1$ with probability $0$;
- $010 = 2$ with probability $0$;
- $011 = 3$ with probability $0$;
- $100 = 4$ with probability $0$;
- $101 = 5$ with probability $1$;
- $110 = 6$ with probability $0$;
- $111 = 7$ with probability $0$.

5 is guaranteed outcome.
Say 3 qubits have state $(1, 5, 9, 2, 6)$.

Measurement produces
- $000 = 0$ with probability $\frac{9}{173}$;
- $001 = 1$ with probability $\frac{1}{173}$;
- $010 = 2$ with probability $\frac{16}{173}$;
- $011 = 3$ with probability $\frac{1}{173}$;
- $100 = 4$ with probability $\frac{25}{173}$;
- $101 = 5$ with probability $\frac{81}{173}$;
- $110 = 6$ with probability $\frac{4}{173}$;
- $111 = 7$ with probability $\frac{36}{173}$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
- $000 = 0$ with probability $0$;
- $001 = 1$ with probability $0$;
- $010 = 2$ with probability $0$;
- $011 = 3$ with probability $0$;
- $100 = 4$ with probability $0$;
- $101 = 5$ with probability $1$;
- $110 = 6$ with probability $0$;
- $111 = 7$ with probability $0$.

5 is guaranteed outcome.

NOT gates

NOT$_0$ gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (1, 3, 1, 4, 9, 5, 6, 2)$. 

5 is guaranteed outcome.
e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
- $000 = 0$ with probability $9/173$;
- $001 = 1$ with probability $1/173$;
- $010 = 2$ with probability $16/173$;
- $011 = 3$ with probability $1/173$;
- $100 = 4$ with probability $25/173$;
- $101 = 5$ with probability $81/173$;
- $110 = 6$ with probability $4/173$;
- $111 = 7$ with probability $36/173$.

5 is most likely outcome.

NOT gates

$NOT_0$ gate on 3 qubits:

$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (1, 3, 1, 4, 9, 5, 6, 2)$.
e.g.: Say 3 qubits have state $(0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability $0$
$001 = 1$ with probability $0$
$010 = 2$ with probability $0$
$011 = 3$ with probability $0$
$100 = 4$ with probability $0$
$101 = 5$ with probability $1$
$110 = 6$ with probability $0$
$111 = 7$ with probability $0$.

5 is guaranteed outcome.

NOT gates

NOT$_0$ gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (1, 3, 1, 4, 9, 5, 6, 2)$. 
e.g.: Say 3 qubits have state \( (0, 0, 0, 0, 0, 1, 0, 0) \).

Measurement produces
\[
\begin{align*}
000 &= 0 \text{ with probability } 0; \\
001 &= 1 \text{ with probability } 0; \\
010 &= 2 \text{ with probability } 0; \\
011 &= 3 \text{ with probability } 0; \\
100 &= 4 \text{ with probability } 0; \\
101 &= 5 \text{ with probability } 1; \\
110 &= 6 \text{ with probability } 0; \\
111 &= 7 \text{ with probability } 0.
\end{align*}
\]

5 is guaranteed outcome.

\textbf{NOT gates}

\text{NOT}_0 \text{ gate on 3 qubits:}
\[
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (1, 3, 1, 4, 9, 5, 6, 2).
\]
e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

**NOT gates**

**NOT**\textsubscript{0} gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (1, 3, 1, 4, 9, 5, 6, 2)$.

**NOT**\textsubscript{0} gate on 4 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3) \mapsto (1, 3, 1, 4, 9, 5, 6, 2, 3, 5, 8, 5, 7, 9, 3, 9)$. 
e.g.: Say 3 qubits have state 
(0, 0, 0, 0, 0, 1, 0, 0).

Measurement produces
000 = 0 with probability 0;
001 = 1 with probability 0;
010 = 2 with probability 0;
011 = 3 with probability 0;
100 = 4 with probability 0;
101 = 5 with probability 1;
110 = 6 with probability 0;
111 = 7 with probability 0.

5 is guaranteed outcome.

\textbf{NOT gates}

\textbf{NOT}_0 \text{ gate on 3 qubits:}
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto
(1, 3, 1, 4, 9, 5, 6, 2).

\textbf{NOT}_0 \text{ gate on 4 qubits:}
(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3) \mapsto
(1, 3, 1, 4, 9, 5, 6, 2, 3, 5, 8, 5, 7, 9, 3, 9).

\textbf{NOT}_1 \text{ gate on 3 qubits:}
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto
(4, 1, 3, 1, 2, 6, 5, 9).
e.g.: Say 3 qubits have state (0, 0, 0, 0, 0, 1, 0, 0).

Measurement produces
000 = 0 with probability 0;
001 = 1 with probability 0;
010 = 2 with probability 0;
011 = 3 with probability 0;
100 = 4 with probability 0;
101 = 5 with probability 1;
110 = 6 with probability 0;
111 = 7 with probability 0.

5 is guaranteed outcome.

NOT gates

$\text{NOT}_0$ gate on 3 qubits:
$\begin{pmatrix} 3, 1, 4, 1, 5, 9, 2, 6 \end{pmatrix} \mapsto \begin{pmatrix} 1, 3, 1, 4, 9, 5, 6, 2 \end{pmatrix}$.

$\text{NOT}_0$ gate on 4 qubits:
$\begin{pmatrix} 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3 \end{pmatrix} \mapsto \begin{pmatrix} 1, 3, 1, 4, 9, 5, 6, 2, 3, 5, 8, 5, 7, 9, 3, 9 \end{pmatrix}$.

$\text{NOT}_1$ gate on 3 qubits:
$\begin{pmatrix} 3, 1, 4, 1, 5, 9, 2, 6 \end{pmatrix} \mapsto \begin{pmatrix} 4, 1, 3, 1, 2, 6, 5, 9 \end{pmatrix}$.

$\text{NOT}_2$ gate on 3 qubits:
$\begin{pmatrix} 3, 1, 4, 1, 5, 9, 2, 6 \end{pmatrix} \mapsto \begin{pmatrix} 5, 9, 2, 6, 3, 1, 4, 1 \end{pmatrix}$.
3 qubits have state
(0, 0, 1, 0, 0).

Measurement produces
000 = 0 with probability 0;
001 = 1 with probability 0;
010 = 2 with probability 0;
011 = 3 with probability 0;
100 = 4 with probability 0;
101 = 5 with probability 1;
110 = 6 with probability 0;
111 = 7 with probability 0.

5 is guaranteed outcome.

**NOT gates**

**NOT**₀ gate on 3 qubits:
(3, 1, 4, 1, 5, 9, 2, 6) \[\rightarrow\]
(1, 3, 1, 4, 9, 5, 6, 2).

**NOT**₀ gate on 4 qubits:
(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3) \[\rightarrow\]
(1, 3, 1, 4, 9, 5, 6, 2, 3, 5, 8, 5, 7, 9, 3, 9).

**NOT**₁ gate on 3 qubits:
(3, 1, 4, 1, 5, 9, 2, 6) \[\rightarrow\]
(4, 1, 3, 1, 2, 6, 5, 9).

**NOT**₂ gate on 3 qubits:
(3, 1, 4, 1, 5, 9, 2, 6) \[\rightarrow\]
(5, 9, 2, 6, 3, 1, 4, 1).

State measurement:
(1, 0, 0, 0, 0, 0, 0, 0) 000
(0, 1, 0, 0, 0, 0, 0, 0) 001
(0, 0, 1, 0, 0, 0, 0, 0) 010
(0, 0, 0, 1, 0, 0, 0, 0) 011
(0, 0, 0, 0, 1, 0, 0, 0) 100
(0, 0, 0, 0, 0, 1, 0, 0) 101
(0, 0, 0, 0, 0, 0, 1, 0) 110
(0, 0, 0, 0, 0, 0, 0, 1) 111

Operation on quantum state:
NOT₀, swapping pairs.

Operation after measurement:
flipping bit 0 of result.

Flip: output is not input.
NOT gates

NOT₀ gate on 3 qubits:
(3, 1, 4, 1, 5, 9, 2, 6) \rightarrow (1, 3, 1, 4, 9, 5, 6, 2).

NOT₀ gate on 4 qubits:
(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3) \rightarrow (1, 3, 1, 4, 9, 5, 6, 2, 3, 5, 8, 5, 7, 9, 3, 9).

NOT₁ gate on 3 qubits:
(3, 1, 4, 1, 5, 9, 2, 6) \rightarrow (4, 1, 3, 1, 2, 6, 5, 9).

NOT₂ gate on 3 qubits:
(3, 1, 4, 1, 5, 9, 2, 6) \rightarrow (5, 9, 2, 6, 3, 1, 4, 1).

state

(1, 0, 0, 0, 0, 0, 0, 0) \rightarrow (0, 1, 0, 0, 0, 0, 0, 0).
(0, 0, 1, 0, 0, 0, 0, 0) \rightarrow (0, 0, 0, 1, 0, 0, 0, 0).
(0, 0, 0, 0, 1, 0, 0, 0) \rightarrow (0, 0, 0, 0, 0, 1, 0, 0).
(0, 0, 0, 0, 0, 1, 0, 0) \rightarrow (0, 0, 0, 0, 0, 0, 1, 0).

Operation on quantum state:
NOT₀, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.
NOT gates

NOT\(_0\) gate on 3 qubits:
(3, 1, 4, 1, 5, 9, 2, 6) \(\mapsto\)
(1, 3, 1, 4, 9, 5, 6, 2).

NOT\(_0\) gate on 4 qubits:
(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3) \(\mapsto\)
(1, 3, 1, 4, 9, 5, 6, 2, 3, 5, 8, 5, 7, 9, 3, 9).

NOT\(_1\) gate on 3 qubits:
(3, 1, 4, 1, 5, 9, 2, 6) \(\mapsto\)
(4, 1, 3, 1, 2, 6, 5, 9).

NOT\(_2\) gate on 3 qubits:
(3, 1, 4, 1, 5, 9, 2, 6) \(\mapsto\)
(5, 9, 2, 6, 3, 1, 4, 1).

Operation on quantum state:
NOT\(_0\), swapping pairs.

Operation after measurement:
flipping bit 0 of result.

Flip: output is not input.
NOT gates

\( \text{NOT}_0 \) gate on 3 qubits:
\[
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (1, 3, 1, 4, 9, 5, 6, 2).
\]

\( \text{NOT}_0 \) gate on 4 qubits:
\[
(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3) \mapsto (1, 3, 1, 4, 9, 5, 6, 2, 3, 5, 8, 5, 7, 9, 3, 9).
\]

\( \text{NOT}_1 \) gate on 3 qubits:
\[
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (4, 1, 3, 1, 2, 6, 5, 9).
\]

\( \text{NOT}_2 \) gate on 3 qubits:
\[
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (5, 9, 2, 6, 3, 1, 4, 1).
\]

State measurement:

\[
\begin{align*}
(1, 0, 0, 0, 0, 0, 0, 0) & \rightarrow 000 \\
(0, 1, 0, 0, 0, 0, 0, 0) & \rightarrow 001 \\
(0, 0, 1, 0, 0, 0, 0, 0) & \rightarrow 010 \\
(0, 0, 0, 1, 0, 0, 0, 0) & \rightarrow 011 \\
(0, 0, 0, 0, 1, 0, 0, 0) & \rightarrow 100 \\
(0, 0, 0, 0, 0, 1, 0, 0) & \rightarrow 101 \\
(0, 0, 0, 0, 0, 0, 1, 0) & \rightarrow 110 \\
(0, 0, 0, 0, 0, 0, 0, 1) & \rightarrow 111
\end{align*}
\]

Operation on quantum state:

\( \text{NOT}_0 \), swapping pairs.

Operation after measurement:

flipping bit 0 of result.

Flip: output is not input.
NOT gates

NOT 0 gate on 3 qubits:
\((3 ; 1 ; 4 ; 1 ; 5 ; 9 ; 2 ; 6) \mapsto (1 ; 3 ; 1 ; 4 ; 9 ; 5 ; 6 ; 2).\)

NOT 0 gate on 4 qubits:
\((3 ; 1 ; 4 ; 1 ; 5 ; 9 ; 2 ; 6 ; 5 ; 3 ; 1 ; 2 ; 6 ; 5 ; 9 ; 3 ; 9) \mapsto (3 ; 1 ; 4 ; 1 ; 5 ; 9 ; 2 ; 6 ; 5 ; 3 ; 1 ; 2 ; 6 ; 5 ; 9 ; 3 ; 9).\)

NOT 1 gate on 3 qubits:
\((3 ; 1 ; 4 ; 1 ; 5 ; 9 ; 2 ; 6) \mapsto (4 ; 1 ; 3 ; 1 ; 2 ; 6 ; 5 ; 9).\)

NOT 2 gate on 3 qubits:
\((3 ; 1 ; 4 ; 1 ; 5 ; 9 ; 2 ; 6) \mapsto (5 ; 9 ; 2 ; 6 ; 3 ; 1 ; 4 ; 1).\)

state measurement

\begin{align*}
\text{state} & \quad \text{measurement} \\
(1, 0, 0, 0, 0, 0, 0, 0) & \quad 000 \\
(0, 1, 0, 0, 0, 0, 0, 0) & \quad 001 \\
(0, 0, 1, 0, 0, 0, 0, 0) & \quad 010 \\
(0, 0, 0, 1, 0, 0, 0, 0) & \quad 011 \\
(0, 0, 0, 0, 1, 0, 0, 0) & \quad 100 \\
(0, 0, 0, 0, 0, 1, 0, 0) & \quad 101 \\
(0, 0, 0, 0, 0, 0, 1, 0) & \quad 110 \\
(0, 0, 0, 0, 0, 0, 0, 1) & \quad 111
\end{align*}

Operation on quantum state:
\(\text{NOT}_0,\) swapping pairs.

Operation after measurement:
flipping bit 0 of result.

Flip: output is not input.

Controlled-NOT gates

e.g. \(\text{CNOT}\) \(1 ; 0 :\)
\((3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 1, 4, 5, 9, 6, 2).\)

\[(3, 1, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 1, 4, 5, 9, 6, 2).\]
NOT gates

**NOT 0 gate on 3 qubits:**
\[(3; 1; 4; 1; 5; 9; 2; 6) \mapsto (1; 3; 1; 4; 9; 5; 6; 2).\]

**NOT 0 gate on 4 qubits:**
\[(3; 1; 4; 1; 5; 9; 2; 6; 5; 3; 5; 8; 9; 7; 9) \mapsto (1; 3; 1; 4; 9; 5; 6; 2; 3; 5; 8; 5; 7; 9; 3).\]

**NOT 1 gate on 3 qubits:**
\[(3; 1; 4; 1; 5; 9; 2; 6) \mapsto (4; 1; 3; 1; 2; 6; 5; 9).\]

**NOT 2 gate on 3 qubits:**
\[(3; 1; 4; 1; 5; 9; 2; 6) \mapsto (5; 9; 2; 6; 3; 1; 4; 1).\]

---

**State measurement**
\[(1; 0; 0; 0; 0; 0; 0; 0) \leftrightarrow 000\]
\[(0; 1; 0; 0; 0; 0; 0; 0) \leftrightarrow 001\]
\[(0; 0; 1; 0; 0; 0; 0; 0) \leftrightarrow 010\]
\[(0; 0; 0; 1; 0; 0; 0; 0) \leftrightarrow 011\]
\[(0; 0; 0; 0; 1; 0; 0; 0) \leftrightarrow 100\]
\[(0; 0; 0; 0; 0; 1; 0; 0) \leftrightarrow 101\]
\[(0; 0; 0; 0; 0; 0; 1; 0) \leftrightarrow 110\]
\[(0; 0; 0; 0; 0; 0; 0; 1) \leftrightarrow 111\]

**Operation on quantum state:**
NOT_0, swapping pairs.

**Operation after measurement:**
flipping bit 0 of result.

**Flip:** output is not input.

---

**Controlled-NOT gates**

*e.g. CNOT_{1,0}:
\[(3; 1; 4; 1; 5; 9; 2; 6) \mapsto (3; 1; 1; 4; 5; 9; 6; 2).\]*
### NOT gates

NOT 0 gate on 3 qubits:

\[(3,1,4,1,5,9,2,6) \mapsto (1,3,1,4,9,5,6,2).\]

NOT 0 gate on 4 qubits:

\[(3,1,4,1,5,9,2,6;5,3,5,8,9,7,9,3) \mapsto (1,3,1,4,9,5,6,2;3,5,8,5,7,9,3,9).\]

NOT 1 gate on 3 qubits:

\[(3,1,4,1,5,9,2,6) \mapsto (4,1,3,1,2,6,5,9).\]

NOT 2 gate on 3 qubits:

\[(3,1,4,1,5,9,2,6) \mapsto (5,9,2,6,3,1,4,1).\]

### State Measurement

<table>
<thead>
<tr>
<th>State</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,0,0,0,0,0,0,0)</td>
<td>000</td>
</tr>
<tr>
<td>(0,1,0,0,0,0,0,0)</td>
<td>001</td>
</tr>
<tr>
<td>(0,0,1,0,0,0,0,0)</td>
<td>010</td>
</tr>
<tr>
<td>(0,0,0,1,0,0,0,0)</td>
<td>011</td>
</tr>
<tr>
<td>(0,0,0,0,1,0,0,0)</td>
<td>100</td>
</tr>
<tr>
<td>(0,0,0,0,0,1,0,0)</td>
<td>101</td>
</tr>
<tr>
<td>(0,0,0,0,0,0,1,0)</td>
<td>110</td>
</tr>
<tr>
<td>(0,0,0,0,0,0,0,1)</td>
<td>111</td>
</tr>
</tbody>
</table>

Operation on quantum state:

NOT 0, swapping pairs.

Operation after measurement:

flipping bit 0 of result.

Flip: output is not input.

### Controlled-NOT Gates

e.g. CNOT\(_{1,0}\):

\[(3,1,4,1,5,9,2,6) \mapsto (3,1,1,4,5,9,6,2).\]
### State Measurement

<table>
<thead>
<tr>
<th>State</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0, 0, 0, 0, 0, 0, 0)</td>
<td>000</td>
</tr>
<tr>
<td>(0, 1, 0, 0, 0, 0, 0, 0)</td>
<td>001</td>
</tr>
<tr>
<td>(0, 0, 1, 0, 0, 0, 0, 0)</td>
<td>010</td>
</tr>
<tr>
<td>(0, 0, 0, 1, 0, 0, 0, 0)</td>
<td>011</td>
</tr>
<tr>
<td>(0, 0, 0, 0, 1, 0, 0, 0)</td>
<td>100</td>
</tr>
<tr>
<td>(0, 0, 0, 0, 0, 1, 0, 0)</td>
<td>101</td>
</tr>
<tr>
<td>(0, 0, 0, 0, 0, 0, 1, 0)</td>
<td>110</td>
</tr>
<tr>
<td>(0, 0, 0, 0, 0, 0, 0, 1)</td>
<td>111</td>
</tr>
</tbody>
</table>

**Operation on quantum state:**

- **NOT**₀, swapping pairs.

**Operation after measurement:**

- Flipping bit 0 of result.

**Flip:** output is not input.

---

### Controlled-NOT Gates

- **e.g.** CNOT₁₀:

  - (3, 1, 4, 1, 5, 9, 2, 6) → (3, 1, 1, 4, 5, 9, 6, 2).

---

### Controlled-NOT Gates (Examples)

- E.g., CNOT₁₀:

  - (3, 1, 4, 1, 5, 9, 2, 6) → (3, 1, 1, 4, 5, 9, 6, 2).
state measurement

(1, 0, 0, 0, 0, 0, 0, 0) 000
(0, 1, 0, 0, 0, 0, 0, 0) 001
(0, 0, 1, 0, 0, 0, 0, 0) 010
(0, 0, 0, 1, 0, 0, 0, 0) 011
(0, 0, 0, 0, 1, 0, 0, 0) 100
(0, 0, 0, 0, 0, 1, 0, 0) 101
(0, 0, 0, 0, 0, 0, 1, 0) 110
(0, 0, 0, 0, 0, 0, 0, 1) 111

Operation on quantum state:
NOT₀, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

Controlled-NOT gates
e.g. CNOT₁,₀:
(3, 1, 4, 1, 5, 9, 2, 6) ↦→
(3, 1, 1, 4, 5, 9, 6, 2).

Operation after measurement:
flipping bit 0 if bit 1 is set; i.e.,
(q₂, q₁, q₀) ↦→ (q₂, q₁, q₀ ⊕ q₁).
### Controlled-NOT gates

<table>
<thead>
<tr>
<th>State</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1, 0, 0, 0, 0, 0, 0, 0))</td>
<td>000</td>
</tr>
<tr>
<td>((0, 1, 0, 0, 0, 0, 0, 0))</td>
<td>001</td>
</tr>
<tr>
<td>((0, 0, 1, 0, 0, 0, 0, 0))</td>
<td>010</td>
</tr>
<tr>
<td>((0, 0, 0, 1, 0, 0, 0, 0))</td>
<td>011</td>
</tr>
<tr>
<td>((0, 0, 0, 0, 1, 0, 0, 0))</td>
<td>100</td>
</tr>
<tr>
<td>((0, 0, 0, 0, 0, 1, 0, 0))</td>
<td>101</td>
</tr>
<tr>
<td>((0, 0, 0, 0, 0, 0, 1, 0))</td>
<td>110</td>
</tr>
<tr>
<td>((0, 0, 0, 0, 0, 0, 0, 1))</td>
<td>111</td>
</tr>
</tbody>
</table>

Operation on quantum state:
NOT\(_0\), swapping pairs.

Operation after measurement:
flipping bit 0 if bit 1 is set; i.e.,
\((q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)\).

E.g. CNOT\(_{1,0}\):
\((3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 1, 4, 5, 9, 6, 2)\).

E.g. CNOT\(_{2,0}\):
\((3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 9, 5, 6, 2)\).
Operation on quantum state:
\( \text{NOT}_0 \), swapping pairs.

Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

Controlled-NOT gates

\( \text{CNOT}_{1,0} \):

\[
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 1, 4, 5, 9, 6, 2).
\]

Operation after measurement:
flipping bit 0 if bit 1 is set; i.e.,
\[ (q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1). \]

\( \text{CNOT}_{2,0} \):

\[
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 9, 5, 6, 2).
\]

\( \text{CNOT}_{0,2} \):

\[
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 9, 4, 6, 5, 1, 2, 1).
\]
Operation on quantum state:
swapping pairs.

Operation after measurement:
flipping bit 0 of result.

Flip: output is not input.

Controlled-NOT gates

e.g. CNOT_{1,0}:
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 1, 4, 5, 9, 6, 2).

Operation after measurement:
flipping bit 0 if bit 1 is set; i.e.,
(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1).

e.g. CNOT_{2,0}:
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 9, 5, 6, 2).

e.g. CNOT_{0,2}:
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 9, 4, 6, 5, 1, 2, 1).

Toffoli gates

Also known as controlled-controlled-NOT gates.

e.g. CCNOT_{2,1,0}:
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 5, 9, 6, 2).
Quantum state:

<table>
<thead>
<tr>
<th>Measurement</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation on quantum state:

- **NOT 0**, swapping pairs.
- **Operation after measurement:** flipping bit 0 of result.
- **Flip:** output is not input.

**Controlled-NOT gates**

e.g. **CNOT\(_{1,0}\):**

\[(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 1, 4, 5, 9, 6, 2)\].

Operation after measurement:

flipping bit 0 *if* bit 1 is set; i.e.,

\[(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)\].

e.g. **CNOT\(_{2,0}\):**

\[(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 9, 5, 6, 2)\].

\[q_2, q_1, q_0\]⇒\[q_2, q_1, q_0 ⊕ q_1\]

e.g. **CNOT\(_{0,2}\):**

\[(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 9, 4, 6, 5, 1, 2, 1)\].

**Toffoli gates**

Also known as controlled-controlled-NOT gates.

e.g. **CCNOT\(_{2,1,0}\):**

\[(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 5, 9, 2, 6)\].
Controller-NOT gates

e.g. CNOT\textsubscript{1,0}:
\[(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 1, 4, 5, 9, 6, 2)\].

Operation after measurement:
flipping bit 0 if bit 1 is set; i.e.,
\[(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)\].

e.g. CNOT\textsubscript{2,0}:
\[(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 9, 5, 6, 2)\].

e.g. CNOT\textsubscript{0,2}:
\[(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 9, 4, 6, 5, 1, 2, 1)\].

Toffoli gates

Also known as controlled-controlled-NOT gates.

e.g. CCNOT\textsubscript{2,1,0}:
\[(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 5, 9, 6, 2)\].
Controlled-NOT gates

e.g. $\text{CNOT}_{1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 if bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. $\text{CNOT}_{2,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. $\text{CNOT}_{0,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli gates

Also known as controlled-controlled-NOT gates.

e.g. $\text{CCNOT}_{2,1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 5, 9, 6, 2)$. 
**Controlled-NOT gates**

e.g. $\text{CNOT}_{1,0}$:
$$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 1, 4, 5, 9, 6, 2).$$

Operation after measurement:
flipping bit 0 if bit 1 is set; i.e.,
$$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1).$$

e.g. $\text{CNOT}_{2,0}$:
$$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 9, 5, 6, 2).$$

e.g. $\text{CNOT}_{0,2}$:
$$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 9, 4, 6, 5, 1, 2, 1).$$

**Toffoli gates**

Also known as
controlled-controlled-NOT gates.

e.g. $\text{CCNOT}_{2,1,0}$:
$$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 5, 9, 6, 2).$$

Operation after measurement:
$$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2).$$
Controlled-NOT gates

e.g. CNOT\(_{1,0}\):
\((3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 1, 4, 5, 9, 6, 2)\).

Operation after measurement:
flipping bit 0 if bit 1 is set; i.e.,
\((q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)\).

e.g. CNOT\(_{2,0}\):
\((3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 9, 5, 6, 2)\).

e.g. CNOT\(_{0,2}\):
\((3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 9, 4, 6, 5, 1, 2, 1)\).

Toffoli gates

Also known as
controlled-controlled-NOT gates.

e.g. CCNOT\(_{2,1,0}\):
\((3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 5, 9, 6, 2)\).

Operation after measurement:
\((q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)\).

e.g. CCNOT\(_{0,1,2}\):
\((3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 6, 5, 9, 2, 1)\).
### Controlled-NOT gates

\( \text{CNOT}_{1,0}: \)

\( (3, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 5, 9, 2, 6). \)

**Operation after measurement:**

- bit 0 if bit 1 is set; i.e., \( q_0 \mapsto (q_2, q_1, q_0 \oplus q_1). \)

\( \text{CNOT}_{2,0}: \)

\( (3, 1, 5, 9, 2, 6) \mapsto (3, 1, 9, 5, 6, 2). \)

\( \text{CNOT}_{0,2}: \)

\( (3, 1, 5, 9, 2, 6) \mapsto (3, 5, 1, 2, 1). \)

### Toffoli gates

Also known as controlled-controlled-NOT gates.

e.g. \( \text{CCNOT}_{2,1,0}: \)

\( (3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 5, 9, 6, 2). \)

**Operation after measurement:**

\( (q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2). \)

e.g. \( \text{CCNOT}_{0,1,2}: \)

\( (3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 6, 5, 9, 2, 1). \)

### More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.
Controlled-NOT gates

Also known as controlled-controlled-NOT gates.

e.g. CCNOT_2,1,0:
(3, 1, 4, 1, 5, 9, 2, 6) \rightarrow
(3, 1, 4, 1, 5, 9, 6, 2).

Operation after measurement:

(q_2, q_1, q_0) \rightarrow (q_2, q_1, q_0 \oplus q_1 q_2).

e.g. CCNOT_0,1,2:
(3, 1, 4, 1, 5, 9, 2, 6) \rightarrow
(3, 1, 4, 6, 5, 9, 2, 1).

More shuffling
Combine NOT, CNOT, Toffoli to build other permutations.
Controlled-NOT gates

Also known as controlled-controlled-NOT gates.

e.g. CCNOT_{2,1,0}:
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto
(3, 1, 4, 1, 5, 9, 2, 6).

Operation after measurement:
(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2).

e.g. CCNOT_{0,1,2}:
(3, 1, 4, 1, 5, 9, 2, 6) \mapsto
(3, 1, 4, 6, 5, 9, 2, 1).

More shuffling

Combine NOT, CNOT, Toffoli gates to build other permutations.

(i.e., (3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 6, 5, 9, 2, 1).)
Toffoli gates

Also known as controlled-controlled-NOT gates.

e.g. $\text{CCNOT}_{2,1,0}$:

$$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 5, 9, 6, 2).$$

Operation after measurement:

$$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2).$$

e.g. $\text{CCNOT}_{0,1,2}$:

$$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 6, 5, 9, 2, 1).$$

More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.
Toffoli gates

Also known as controlled-controlled-NOT gates.

e.g. CCNOT\(_{2,1,0}\):
\[(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 1, 5, 9, 6, 2)\].

Operation after measurement:
\[(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)\].

e.g. CCNOT\(_{0,1,2}\):
\[(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (3, 1, 4, 6, 5, 9, 2, 1)\].

More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

e.g. series of gates to rotate 8 positions by distance 1:

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 6 & 5 & 9 & 2 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
3 & 6 & 4 & 1 & 5 & 1 & 2 & 9 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
6 & 3 & 1 & 4 & 1 & 5 & 9 & 2 \\
\end{array}
\]
Toffoli gates
Also known as controlled-controlled-NOT gates.
e.g. CCNOT 2 ; 1 ; 0 :
(3 ; 1 ; 4 ; 1 ; 5 ; 9 ; 2 ; 6) \rightarrow (3 ; 1 ; 4 ; 1 ; 5 ; 9 ; 2 ; 6).

Operation after measurement:
(q_2 ; q_1 ; q_0) \rightarrow (q_2 ; q_1 ; q_0 \oplus q_1 q_2).

More shuffling
Combine NOT, CNOT, Toffoli to build other permutations.
e.g. series of gates to rotate 8 positions by distance 1:

Hadamard gates
Hadamard 0 :
(a, b) \rightarrow (a + b, a - b).

Hadamard gate
(a, b) \rightarrow (3, 1, 4, 1, 5, 9, 2, 6)

3
4
4
4
4
1

5
4
4
4
4
9

2
4
4
4
4
6

4 2 5 3 1

4 8

4

2

3

1

3

1

4

1

5

9

2

6

3

1

4

1

5

9

2

1

3

6

4

1

5

1

2

9

6

3

1

4

1

5

9

2

NOT

6

3

1

4

1

5

9

2
Toffoli gates
Also known as controlled-controlled-NOT gates.

\[ \text{e.g. CCNOT } 2 ; 1 ; 0 : \]
\[ (3 ; 1 ; 4 ; 1 ; 5 ; 9 ; 2 ; 6) \mapsto (3 ; 1 ; 4 ; 1 ; 5 ; 9 ; 2 ; 6). \]

Operation after measurement:
\[ (q_2 ; q_1 ; q_0) \mapsto (q_2 ; q_1 ; q_0 \oplus q_1 q_2). \]

More shuffling
Combine NOT, CNOT, Toffoli to build other permutations.

\[ \text{e.g. series of gates to rotate 8 positions by distance 1:} \]
\[ \begin{array}{ccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2
\end{array} \]
\[ \begin{array}{ccccccc}
CCNOT_{0,1,2}
\end{array} \]
\[ \begin{array}{ccccccc}
3 & 1 & 4 & 6 & 5 & 9 & 2 & 1
\end{array} \]
\[ \begin{array}{ccccccc}
\text{CNOT}_{0,1}
\end{array} \]
\[ \begin{array}{ccccccc}
3 & 6 & 4 & 1 & 5 & 1 & 2 & 9
\end{array} \]
\[ \begin{array}{ccccccc}
\text{NOT}_0
\end{array} \]
\[ \begin{array}{ccccccc}
6 & 3 & 1 & 4 & 1 & 5 & 9 & 2
\end{array} \]

Hadamard gates
Hadamard \( 0 \):
\[ (a, b) \mapsto (a + b, a - b). \]
More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

e.g. series of gates to
rotate 8 positions by distance 1:

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>1</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>9</th>
<th>2</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCNOT_{0,1,2}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CNOT_{0,1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT_{0}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hadamard gates

Hadamard_0:

\[(a, b) \mapsto (a + b, a - b).\]

\[
\begin{pmatrix}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & 7
\end{pmatrix}
\]
More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

E.g. series of gates to rotate 8 positions by distance 1:

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 6 & 5 & 9 & 2 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
3 & 6 & 4 & 1 & 5 & 1 & 2 & 9 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
6 & 3 & 1 & 4 & 1 & 5 & 9 & 2 \\
\end{array}
\]

Hadamard gates

Hadamard \(_0\):

\[(a, b) \mapsto (a + b, a - b).\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
| & | & | & | & & & \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4 \\
\end{array}
\]
More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

e.g. series of gates to rotate 8 positions by distance 1:

\[
\begin{array}{ccccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\end{array}
\]

\begin{array}{ccccccccc}
3 & 1 & 4 & 6 & 5 & 9 & 2 & 1 \\
\end{array}

\begin{array}{ccccccccc}
3 & 6 & 4 & 1 & 5 & 1 & 2 & 9 \\
\end{array}

\begin{array}{ccccccccc}
6 & 3 & 1 & 4 & 1 & 5 & 9 & 2 \\
\end{array}

Hadamard gates

Hadamard_0:
\((a, b) \mapsto (a + b, a - b)\).

\[
\begin{array}{ccccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\end{array}
\]

\begin{array}{ccccccccc}
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4 \\
\end{array}

Hadamard_1:
\((a, b, c, d) \mapsto (a + c, b + d, a - c, b - d)\).

\[
\begin{array}{ccccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\end{array}
\]

\begin{array}{ccccccccc}
7 & 2 & -1 & 0 & 7 & 15 & 3 & 3 \\
\end{array}
More shuffling

Use NOT, CNOT, Toffoli gates to build other permutations.

For example, a series of gates to rotate 8 positions by distance 1:

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4 \\
7 & 2 & -1 & 0 & 15 & 3 & 3 \\
\end{array}
\]

**Hadamard gates**

Hadamard\(_0\):

\[(a, b) \mapsto (a + b, a - b).\]

Hadamard\(_1\):

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

**Simon's algorithm**

Step 1. Set up pure zero state:

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
More shuffling
Combine NOT, CNOT, Toffoli to build other permutations.
e.g. series of gates to rotate 8 positions by distance 1:

Hadamard gates
Hadamard₀:
\[(a, b) \mapsto (a + b, a - b).\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \times & \times \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4
\end{array}
\]

Hadamard₁:
\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \times & \times \\
7 & 2 & -1 & 0 & 7 & 15 & 3 & 3
\end{array}
\]

Simon’s algorithm
Step 1. Set up pure zero state:

1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Hadamard gates
Hadamard₀:
\[(a, b) \mapsto (a + b, a - b).\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \times & \times \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4
\end{array}
\]

Hadamard₁:
\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \times & \times \\
7 & 2 & -1 & 0 & 7 & 15 & 3 & 3
\end{array}
\]
More shuffling
Combine NOT, CNOT, Toffoli to build other permutations.

e.g. series of gates to rotate 8 positions by distance 1:

```
3 1 4 1
L L L L
5 9 2 6
```

CCNOT 0, 1, 2:

```
3 1
9 9 9 9
4 6
```

CNOT 0, 1:

```
3 1
9 9 9 9
4 6
```

NOT 0:

```
3 1
9 9 9 9
4 6
```

Hadamard gates
Hadamard 0:

\[(a, b) \mapsto (a + b, a - b).\]

```
3 1 4 1 5 9 2 6
| | | | | | | |
4 2 5 3 1 4 0 0
```

Hadamard 1:

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

```
3 1 4 1 5 9 2 6
| | | | | | | |
4 2 5 3 1 4 0 0
```

Simon’s algorithm
Step 1. Set up pure zero state:

```
1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
```

```
7 2 -1 0
7 15 3 3
```
Hadamard gates

Hadamard$_0$:

$$(a, b) \mapsto (a + b, a - b).$$

Hadamard$_1$:

$$(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).$$

Simon’s algorithm

Step 1. Set up pure zero state:

$$1, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0.$$

\[3\times 7\]
Hadamard gates

Hadamard\(_0\):

\[(a, b) \mapsto (a + b, a - b).\]

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>9</th>
<th>2</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4 2 5 3 14 -4 8 -4

Hadamard\(_1\):

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>9</th>
<th>2</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7 2 -1 0 7 15 3 3

Simon’s algorithm

Step 2. Hadamard\(_0\):

1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Hadamard gates

Hadamard$_0$:

$$(a, b) \mapsto (a + b, a - b).$$

Hadamard$_1$:

$$(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).$$

Simon’s algorithm

Step 3. Hadamard$_1$:

$$1, 1, 1, 1, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0.$$
**Hadamard gates**

**Hadamard** subscripts:

- **Hadamard**\(_0\): (\(a, b\)) \(\mapsto (a + b, a - b)\).

- **Hadamard**\(_1\): (\(a, b, c, d\)) \(\mapsto (a + c, b + d, a - c, b - d)\).

- Each column is a parallel universe.

**Simon’s algorithm**

**Step 4. Hadamard**\(_2\): 

\[
\begin{align*}
1, 1, 1, 1, 1, 1, 1, 1, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0.
\end{align*}
\]

Each column is a parallel universe.
Hadamard gates

Hadamard $0$:

$$(a, b) \mapsto (a + b, a - b).$$

Hadamard $1$:

$$(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).$$

Simon’s algorithm

Step 5. $\text{CNOT}_{0,3}$:

$$
\begin{array}{cccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
$$

Each column is a parallel universe performing its own computations.
Hadamard gates

Hadamard$_0$:

$$(a, b) \mapsto (a + b, a - b).$$

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4 \\
\end{array}
\]

Hadamard$_1$:

$$(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).$$

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \\
7 & 2 & -1 & 0 & 7 & 15 & 3 & 3 \\
\end{array}
\]

Simon’s algorithm

Step 5b. More shuffling:

\[
\begin{array}{cccccccc}
1, 0, 0, 0, 1, 0, 0, 0, \\
0, 1, 0, 0, 0, 1, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 1, 0, 0, 0, 0, 1, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0. \\
\end{array}
\]

Each column is a parallel universe performing its own computations.
Hadamard gates

Hadamard_0:

\[(a, b) \mapsto (a + b, a - b).\]

```
3  1  4  1  5  9  2  6
|X|  |X|  |X|  |X|
4  2  5  3  14 -4  8 -4
```

Hadamard_1:

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

```
3  1  4  1  5  9  2  6
|X|  |X|  |X|  |X|
7  2 -1  0  7  15  3  3
```

Simon’s algorithm

Step 5c. More shuffling:

```
1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1.
```

Each column is a parallel universe performing its own computations.
Hadamard gates

Hadamard$_0$:

\[(a, b) \mapsto (a + b, a - b).\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \times & \times \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4 \\
\end{array}
\]

Hadamard$_1$:

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \times & \times \\
7 & 2 & -1 & 0 & 7 & 15 & 3 & 3 \\
\end{array}
\]
Hadamard gates

Hadamard_0:

\[(a, b) \mapsto (a + b, a - b).\]

<table>
<thead>
<tr>
<th>3 1 4 1</th>
<th>5 9 2 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4 2 5 3</td>
<td>14 -4 8</td>
</tr>
</tbody>
</table>

Hadamard_1:

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

<table>
<thead>
<tr>
<th>3 1 4 1</th>
<th>5 9 2 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>7 2 -1 0</td>
<td>7 15 3</td>
</tr>
</tbody>
</table>

Simon’s algorithm

Step 5e. More shuffling:

\[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.\]

Each column is a parallel universe performing its own computations.
Hadamard gates

Hadamard\textsubscript{0}:

\[(a, b) \mapsto (a + b, a - b).\]

\[
\begin{array}{llllllll}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \times & \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4
\end{array}
\]

Hadamard\textsubscript{1}:

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

\[
\begin{array}{llllllll}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \times & \\
7 & 2 & -1 & 0 & 7 & 15 & 3 & 3
\end{array}
\]

Simon’s algorithm

Step 5f. More shuffling:

\[0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0.\]

Each column is a parallel universe performing its own computations.
Hadamard gates

Hadamard\(_0\):

\((a, b) \mapsto (a + b, a - b)\).

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4
\end{array}
\]

Hadamard\(_1\):

\((a, b, c, d) \mapsto (a + c, b + d, a - c, b - d)\).

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline \\
7 & 2 & -1 & 0 & 7 & 15 & 3 & 3
\end{array}
\]

Simon’s algorithm

Step 5g. More shuffling:

\[
\begin{array}{cccccccc}
0, & 1, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 1, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 1, & 0, & 0, \\
0, & 0, & 0, & 0, & 1, & 0, & 0, & 1, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 1, & 0, & 0, & 0, & 0, & 1
\end{array}
\]

Each column is a parallel universe performing its own computations.
Hadamard gates

Hadamard\textsubscript{0}:

\[(a, b) \mapsto (a + b, a - b).\]

\[
\begin{array}{ccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
| & | & | & | & | & | & |
\end{array}
\]

3 4 2 5 3 14 -4 8 -4

Hadamard\textsubscript{1}:

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

\[
\begin{array}{ccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
| & | & | & | & | & | & |
\end{array}
\]

7 2 -1 0 7 15 3 3

Simon’s algorithm

Step 5h. More shuffling:

0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.
Hadamard gates

Hadamard$_0$:

$$(a, b) \mapsto (a + b, a - b).$$

Hadamard$_1$:

$$(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).$$

Simon’s algorithm

Step 5i. More shuffling:

$\begin{array}{cccccccc}
0, 0, 0, 0, 0, 0, 1, 0, \\
0, 0, 0, 1, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 1, 0, 0, 0, 0, 0, 0, \\
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, \\
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0.
\end{array}$

Each column is a parallel universe performing its own computations.
### Hadamard gates

**Hadamard**

**Hadamard**

**0:**

\[(a, b) \mapsto (a + b, a - b).\]

**Hadamard**

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

### Simon’s algorithm

**Step 5j. Final shuffling:**

\[
\begin{align*}
0, & 0, 0, 0, 0, 0, 0, 0, \\
0, & 0, 0, 1, 0, 0, 1, 0, \\
0, & 0, 0, 0, 0, 0, 0, 0, \\
0, & 0, 1, 0, 0, 0, 0, 1, \\
0, & 1, 0, 0, 1, 0, 0, 0, \\
0, & 0, 0, 0, 0, 0, 0, 1, \\
0, & 0, 0, 0, 0, 0, 0, 0, \\
& 1, 0, 0, 0, 0, 1, 0, 0.
\end{align*}
\]

Each column is a parallel universe performing its own computations.
Hadamard gates

Hadamard_0:

$$(a, b) \mapsto (a + b, a - b).$$

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4
\end{array}
\]

Hadamard_1:

$$(a, b, c, d) \mapsto \quad (a + c, b + d, a - c, b - d).$$

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\hline
\times & \times & \times & \times & \times & \times & \\
7 & 2 & -1 & 0 & 7 & 15 & 3 & 3
\end{array}
\]

Simon’s algorithm

Step 5j. Final shuffling:

\[
\begin{array}{cccccccc}
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 1, & 0, & 0, & 1, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 1, & 0, & 0, & 0, & 0, & 1, \\
0, & 1, & 0, & 0, & 1, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
1, & 0, & 0, & 0, & 0, & 1, & 0, & 0.
\end{array}
\]

Each column is a parallel universe performing its own computations. Surprise: \(u\) and \(u \oplus 101\) match.
Hadamard gates

Hadamard$_0$:

$$(a, b) \mapsto (a + b, a - b).$$

$\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\times & \times & \times & \times & \times & \times & \times & \times \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4 \\
\end{array}$

Hadamard$_1$:

$$(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).$$

$\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\times & \times & \times & \times & \times & \times & \times & \times \\
7 & 2 & -1 & 0 & 7 & 15 & 3 & 3 \\
\end{array}$

Simon's algorithm

Step 6. Hadamard$_0$:

$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$,

$0, 0, 1, 1, 0, 0, 0, 1, 1$,

$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$,

$0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0$,

$1, 1, 0, 0, 1, 1$.

$1, 1, 0, 0, 1, 1, 0, 0$. 
Hadamard gates

Hadamard_0:

\[(a, b) \mapsto (a + b, a - b).\]

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>14</td>
<td>-4</td>
<td>8</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Hadamard_1:

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   | X | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 7 | 2 | -1| 0 | 7 | 15| 3 | 3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Simon’s algorithm

Step 7. Hadamard_1:

\[0, 0, 0, 0, 0, 0, 0, 0, 1, \overline{1}, \overline{1}, 1, 1, 1, \overline{1}, \overline{1}, \overline{1}, \overline{1}, 0, 0, 0, 0, 0, 0, 1, 1, \overline{1}, \overline{1}, 1, \overline{1}, 1, 1, \overline{1}, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, \overline{1}, \overline{1}, \overline{1}.\]
Hadamard gates

Hadamard$_0$:

$$(a, b) \mapsto (a + b, a - b).$$

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>9</th>
<th>2</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>14</td>
<td>-4</td>
<td>8</td>
<td>-4</td>
</tr>
</tbody>
</table>

Hadamard$_1$:

$$(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).$$

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>9</th>
<th>2</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>-1</td>
<td>0</td>
<td>7</td>
<td>15</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Simon’s algorithm

Step 8. Hadamard$_2$:

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, 2, 0, \overline{2},$

$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2},$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, \overline{2},$

$2, 0, 2, 0, 0, 2, 0, 2.$
Hadamard gates

**Hadamard**$_0$: 

\[(a, b) \mapsto (a + b, a - b).\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\times & \times & \times & \times & \times & \times & \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4
\end{array}
\]

**Hadamard**$_1$: 

\[(a, b, c, d) \mapsto (a + c, b + d, a - c, b - d).\]

\[
\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\times & \times & \times & \times & \times & \times & \\
7 & 2 & -1 & 0 & 7 & 15 & 3 & 3
\end{array}
\]

Simon’s algorithm

**Step 8. Hadamard**$_2$: 

\[
\begin{array}{cccccccccccc}
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
2, & 0, & \overline{2}, & 0, & 0, & \overline{2}, & 0, & 2, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
2, & 0, & \overline{2}, & 0, & 0, & \overline{2}, & 0, & \overline{2}, \\
2, & 0, & \overline{2}, & 0, & 0, & \overline{2}, & 0, & \overline{2}, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
2, & 0, & \overline{2}, & 0, & 0, & \overline{2}, & 0, & 2.
\end{array}
\]

**Step 9:** Measure. Obtain some information about the surprise: a random vector orthogonal to 101.