Introduction to quantum algorithms

Daniel J. Bernstein
University of Illinois at Chicago & Technische Universiteit Eindhoven
Data ("state") stored in $n$ bits: an element of $\{0, 1\}^n$, often viewed as representing an element of $\{0, 1, \ldots, 2^n - 1\}$. 
Data ("state") stored in $n$ bits:
an element of $\{0, 1\}^n$,
often viewed as representing
an element of $\{0, 1, \ldots, 2^n - 1\}$.

State stored in $n$ qubits:
a nonzero element of $\mathbb{C}^{2^n}$.
Retrieving this vector is tough!
Data ("state") stored in \( n \) bits: an element of \( \{0, 1\}^n \), often viewed as representing an element of \( \{0, 1, \ldots, 2^n - 1\} \).

State stored in \( n \) qubits: a nonzero element of \( \mathbb{C}^{2^n} \). Retrieving this vector is tough!

If \( n \) qubits have state \((a_0, a_1, \ldots, a_{2^n-1})\) then **measuring** the qubits produces an element of \( \{0, 1, \ldots, 2^n - 1\} \) and destroys the state. Measurement produces element \( q \) with probability \( |a_q|^2 / \sum_r |a_r|^2 \).
Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is “$|0\rangle$” in standard notation. Measurement produces 0.
Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is
“$|0\rangle$” in standard notation.
Measurement produces 0.

$(0, 0, 0, 0, 0, 0, 1, 0)$ is
“$|6\rangle$” in standard notation.
Measurement produces 6.
Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is “$|0\rangle$” in standard notation. Measurement produces 0.

$(0, 0, 0, 0, 0, 0, 1, 0)$ is “$|6\rangle$” in standard notation. Measurement produces 6.

$(0, 0, 0, 0, 0, 0, -7i, 0) = -7i |6\rangle$: Measurement produces 6.
Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is “$|0\rangle$” in standard notation. Measurement produces 0.

$(0, 0, 0, 0, 0, 0, 0, 1)$ is “$|6\rangle$” in standard notation. Measurement produces 6.

$(0, 0, 0, 0, 0, 0, -7i, 0) = -7i|6\rangle$: Measurement produces 6.

$(0, 0, 4, 0, 0, 0, 8, 0) = 4|2\rangle + 8|6\rangle$: Measurement produces 2 with probability 20%, 6 with probability 80%.
Fast quantum operations, part 1

\[(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)\]
is complementing index bit 0, hence “complementing qubit 0”. 
Fast quantum operations, part 1

\((a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)\)
is complementing index bit 0, hence “complementing qubit 0”.

\((a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)\)
is measured as \((q_0, q_1, q_2)\), representing \(q = q_0 + 2q_1 + 4q_2\), with probability \(|a_q|^2 / \sum_r |a_r|^2\).

\((a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)\)
is measured as \((q_0 \oplus 1, q_1, q_2)\), representing \(q \oplus 1\), with probability \(|a_q|^2 / \sum_r |a_r|^2\).
\[(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)\]

is “complementing qubit 2”:

\[(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1).\]
is "complementing qubit 2": 
\((q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)\).

is "swapping qubits 0 and 2": 
\((q_0, q_1, q_2) \mapsto (q_2, q_1, q_0)\).
\((a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)\)

is “complementing qubit 2”:
\((q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)\).

\((a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)\)

is “swapping qubits 0 and 2”:
\((q_0, q_1, q_2) \mapsto (q_2, q_1, q_0)\).

Complementing qubit 2
\[=\text{swapping qubits 0 and 2}\]
  \[\circ\text{complementing qubit 0}\]
  \[\circ\text{swapping qubits 0 and 2}.\]

Similarly: swapping qubits \(i, j\).
\((a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)\)
is a “reversible XOR gate” =
“controlled NOT gate”:
\((q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)\).

Example with more qubits:
\((a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23}, a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31}) \mapsto (a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6, a_8, a_9, a_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14}, a_{16}, a_{17}, a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22}, a_{24}, a_{25}, a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})\).
\((a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)\)

is a “Toffoli gate” = “controlled controlled NOT gate”:
\((q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)\).

Example with more qubits:
\((a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23}, a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31}) \mapsto (a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6, a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14}, a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22}, a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})\).
Reversible computation

Say $p$ is a permutation of $\{0, 1, \ldots, 2^n - 1\}$.

General strategy to compose these fast quantum operations to obtain index permutation

$$(a_p(0), a_p(1), \ldots, a_p(2^n - 1)) \mapsto (a_0, a_1, \ldots, a_{2^n - 1})$$
Reversible computation

Say $p$ is a permutation of $\{0, 1, \ldots, 2^n - 1\}$.

General strategy to compose these fast quantum operations to obtain index permutation

$$(a_p(0), a_p(1), \ldots, a_p(2^n-1)) \quad \mapsto \quad (a_0, a_1, \ldots, a_{2^n-1})$$

1. Build a traditional circuit to compute $j \mapsto p(j)$ using NOT/XOR/AND gates.

2. Convert into reversible gates: e.g., convert AND into Toffoli.
Example: Let’s compute
\[(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)\];
permutation \( q \mapsto q + 1 \mod 8 \).

1. Build a traditional circuit
to compute \( q \mapsto q + 1 \mod 8 \).
2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$: 

$$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3).$$
2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
\[(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3).\]

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
\[(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto (a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5).\]
2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:

$$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3).$$

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:

$$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto (a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5).$$

NOT for $q_0 \leftarrow q_0 \oplus 1$:

$$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5) \mapsto (a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6).$$
This permutation example was deceptively easy.

It didn’t need many operations.

For large $n$, most permutations $p$ need many operations $\Rightarrow$ slow. Really want fast circuits.
This permutation example was deceptively easy.

It didn’t need many operations.

For large $n$, most permutations $p$ need many operations $\Rightarrow$ slow. Really want fast circuits.

Also, it didn’t need extra storage: circuit operated “in place” after computation $c_1 \leftarrow q_1 q_0$ was merged into $q_2 \leftarrow q_2 \oplus c_1$.

Typical circuits aren’t in-place.
Start from any circuit:
inputs $b_1, b_2, \ldots, b_i$;

\[ b_{i+1} = 1 \oplus b_f(i+1) b_g(i+1); \]

\[ b_{i+2} = 1 \oplus b_f(i+2) b_g(i+2); \]

\ldots

\[ b_T = 1 \oplus b_f(T) b_g(T); \]

specified outputs.
Start from any circuit:

inputs $b_1, b_2, \ldots, b_i$;

$$b_{i+1} = 1 \oplus b_f(i+1) b_g(i+1);$$
$$b_{i+2} = 1 \oplus b_f(i+2) b_g(i+2);$$

\ldots

$$b_T = 1 \oplus b_f(T) b_g(T);$$

specified outputs.

Reversible but dirty:

inputs $b_1, b_2, \ldots, b_T$;

$$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_f(i+1) b_g(i+1);$$
$$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_f(i+2) b_g(i+2);$$

\ldots

$$b_T \leftarrow 1 \oplus b_T \oplus b_f(T) b_g(T).$$

Same outputs if all of $b_{i+1}, \ldots, b_T$ started as 0.
Reversible and clean: after finishing dirty computation, set non-outputs back to 0, by repeating same operations on non-outputs in reverse order.

Original computation:
(inputs) $\mapsto$ (inputs, dirt, outputs).

Dirty reversible computation:
(inputs, zeros, zeros) $\mapsto$ (inputs, dirt, outputs).

Clean reversible computation:
(inputs, zeros, zeros) $\mapsto$ (inputs, zeros, outputs).
Given fast circuit for $p$ and fast circuit for $p^{-1}$, build fast reversible circuit for 
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$. 
Given fast circuit for $p$ and fast circuit for $p^{-1}$, build fast reversible circuit for 
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Replace reversible bit operations with Toffoli gates etc.
permuting $\mathbb{C}^{2^{n+z}} \rightarrow \mathbb{C}^{2^{n+z}}$.

Permutation on first $2^n$ entries is 
$(a_{p(0)}, a_{p(1)}, \ldots, a_{p(2^n-1)})$
$\mapsto (a_0, a_1, \ldots, a_{2^n-1})$.

Typically prepare vectors supported on first $2^n$ entries 
so don’t care how permutation acts on last $2^{n+z} - 2^n$ entries.
Warning: Number of qubits $\approx$ number of bit operations in original $p, p^{-1}$ circuits.

This can be much larger than number of bits stored in the original circuits.

Many useful techniques to compress into fewer qubits, but often these lose time. Many subtle tradeoffs.

Crude “poly-time” analyses don’t care about this, but serious cryptanalysis is much more precise.
Fast quantum operations, part 2

“Hadamard”:

\[(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1).\]
“Hadamard”:

\[(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1).\]

\[(a_0, a_1, a_2, a_3) \mapsto (a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3).\]
Fast quantum operations, part 2

“Hadamard”:

\[(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)\].

\[(a_0, a_1, a_2, a_3) \mapsto (a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)\].

Same for qubit 1:

\[(a_0, a_1, a_2, a_3) \mapsto (a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)\].
Fast quantum operations, part 2

“Hadamard”:

$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto
(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Same for qubit 1:

$(a_0, a_1, a_2, a_3) \mapsto
(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

Qubit 0 and then qubit 1:

$(a_0, a_1, a_2, a_3) \mapsto
(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3) \mapsto
(a_0 + a_1 + a_2 + a_3, a_0 - a_1 + a_2 - a_3,
 a_0 + a_1 - a_2 - a_3, a_0 - a_1 - a_2 + a_3)$. 
Repeat \( n \) times: e.g.,

\((1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1)\).

Measuring \((1, 0, 0, \ldots, 0)\) always produces 0.

Measuring \((1, 1, 1, \ldots, 1)\) can produce any output:

\[ \Pr[\text{output} = q] = 1/2^n. \]
Repeat $n$ times: e.g.,

$$(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1).$$

Measuring $(1, 0, 0, \ldots, 0)$ always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$ can produce any output:

$$\Pr[\text{output} = q] = 1/2^n.$$ 

Aside from “normalization” (irrelevant to measurement), have Hadamard $= \text{Hadamard}^{-1}$, so easily work backwards from “uniform superposition” $(1, 1, 1, \ldots, 1)$ to “pure state” $(1, 0, 0, \ldots, 0)$. 

Simon’s algorithm

Assume: nonzero $s \in \{0, 1\}^n$ satisfies $f(x) = f(x \oplus s)$ for every $x \in \{0, 1\}^n$. Can we find this period $s$, given a fast circuit for $f$?
Simon’s algorithm

Assume: nonzero $s \in \{0, 1\}^n$
satisfies $f(x) = f(x \oplus s)$
for every $x \in \{0, 1\}^n$.
Can we find this period $s$,
given a fast circuit for $f$?

We don’t have enough data
if $f$ has many periods.
Assume: only periods are 0, s.
Simon’s algorithm

Assume: nonzero $s \in \{0, 1\}^n$ satisfies $f(x) = f(x \oplus s)$ for every $x \in \{0, 1\}^n$.
Can we find this period $s$, given a fast circuit for $f$?

We don’t have enough data if $f$ has many periods.
Assume: only periods are 0, $s$.

Traditional solution:
Compute $f$ for many inputs, sort, analyze collisions.
Success probability is very low until $\#\text{inputs}$ approaches $2^{n/2}$.
Simon’s algorithm is much, much, much faster.

Say \( f \) maps \( n \) bits to \( m \) bits, using \( z \) “ancilla” bits for reversibility.

Prepare \( n + m + z \) qubits in pure zero state: vector \((1, 0, 0, \ldots)\).

Use \( n \)-fold Hadamard to move first \( n \) qubits into uniform superposition: \((1, 1, 1, \ldots, 1, 0, 0, \ldots)\) with \(2^n\) entries 1, others 0.
Apply fast vector permutation for reversible $f$ computation:
1 in position $(q, 0, 0)$
moves to position $(q, f(q), 0)$.

Note symmetry between
1 at $(q, f(q), 0)$ and
1 at $(q \oplus s, f(q), 0)$.

Apply $n$-fold Hadamard.

Measure. By symmetry,
output is orthogonal to $s$.

Repeat $n + 10$ times.
Use Gaussian elimination to (probably) find $s$. 
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 beats traditional algorithm.
Start from uniform superposition over all \( n \)-bit strings \( q \).

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 steps 1 and 2 about \( 0.58 \cdot 2^{0.5n} \) times.

Measure the \( n \) qubits.
With high probability this finds \( s \).
Graph of $q \mapsto a_q$
for an example with $n = 12$
after 0 steps:
Graph of $q \mapsto a_q$
for an example with $n = 12$
after Step 1:
Graph of $q \mapsto a_q$
for an example with $n = 12$
after Step 1 + Step 2:
Graph of $q \mapsto a_q$
for an example with $n = 12$
after Step 1 $+$ Step 2 $+$ Step 1:
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $2 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $3 \times (\text{Step 1 + Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $4 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $5 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $6 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $7 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $8 \times (\text{Step 1} + \text{Step 2})$: 

![Graph of q ↦ a_q](image-url)
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $9 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $10 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $11 \times (\text{Step 1} + \text{Step 2})$:
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $12 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $13 \times (\text{Step 1} + \text{Step 2})$:
Graph of \( q \mapsto a_q \)
for an example with \( n = 12 \)
after \( 14 \times (\text{Step 1 + Step 2}) \):
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $15 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $16 \times (\text{Step 1 + Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $17 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $18 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $19 \times (\text{Step 1} + \text{Step 2})$:
Graph of $q \mapsto a_q$

for an example with $n = 12$

after $20 \times (\text{Step 1 + Step 2})$:
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $25 \times (\text{Step 1 + Step 2})$: 

![Graph](image-url)
Graph of $q \mapsto a_q$

for an example with $n = 12$

after $30 \times (\text{Step 1} + \text{Step 2})$:
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.
Graph of $q \mapsto a_q$

for an example with $n = 12$

after $40 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $45 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $50 \times (\text{Step 1} + \text{Step 2})$:

Traditional stopping point.
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $60 \times (\text{Step 1 + Step 2})$:  

![Graph](image-url)
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $70 \times (\text{Step 1} + \text{Step 2})$: 
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $80 \times (\text{Step 1} + \text{Step 2})$:
Graph of $q \mapsto a_q$
for an example with $n = 12$
after $90 \times (\text{Step 1} + \text{Step 2})$:
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.

\[ \Rightarrow \text{Probability is } \approx 1 \text{ after } \approx (\pi/4)2^{0.5n} \text{ iterations.} \]
Notes on provability

Textbook algorithm analysis:

Proof of correctness

New algorithm

Proof of run time

Mislead students into thinking that best algorithm = best *proven* algorithm.
Reality: state-of-the-art cryptanalytic algorithms are almost never proven.
Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: “Work harder, find proofs!”
Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: “Work harder, find proofs!”

Consensus of the experts: proofs probably do not exist for most of these algorithms. So demanding proofs is silly.
Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: “Work harder, find proofs!”

Consensus of the experts: proofs probably do not exist for most of these algorithms. So demanding proofs is silly.

Without proofs, how do we analyze correctness + speed?
Answer: Real algorithm analysis relies critically on heuristics and computer experiments.
What about quantum algorithms? Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.
What about quantum algorithms? Want to analyze, optimize quantum algorithms \textit{today} to figure out safe crypto against \textit{future} quantum attack.

1. Simulate \textit{tiny} q. computer? \Rightarrow Huge extrapolation errors.
What about quantum algorithms? Want to analyze, optimize quantum algorithms today to figure out safe crypto against future quantum attack.

1. Simulate tiny q. computer?
   $\Rightarrow$ Huge extrapolation errors.

What about quantum algorithms? Want to analyze, optimize quantum algorithms \textit{today} to figure out safe crypto against \textit{future} quantum attack.

1. Simulate tiny q. computer? \(\Rightarrow\) Huge extrapolation errors.


3. Fast \textbf{trapdoor simulation}. Simulator (like prover) knows more than the algorithm does. Tung Chou has implemented this, found errors in two publications.